xmlprague

a conference on XML

Lesser Town Campus
Prague, Czech Republic
25th June 2005

ROOT.CZ

největší server

o Linuxu

a Open source

v České republice

> denně aktuální zpravodajství

> odborné analýzy, návody, zkušenosti z praxe

> jedinečné diskusní fórum s možností hodnocení kvality příspěvků

> centrum české linuxové komunity

> založen v roce 1999

www.root.cz

Contents

General Information

Date

Saturday, June 25th, 2005

Location

Lesser Town Campus, Lecture Hall S5
Malostranské náměstí 25, 110 00 Prague 1, Czech Republic

Speakers

Michael Kay, *Saxonica Limited*
Robin Berjon, *Expway*
Eric van der Vlist, *Dyomedea*
Miloslav Nič, *ICT Prague*
James Fuller, *WebComposite s.r.o*
Petr Pajas, *Charles University, Prague*
Václav Trojan, *Syntea software group a.s.*

Organizing Comitee

Jaroslav Nešetřil, *Charles University, Prague*
Tomáš Kaiser, *University of West Bohemia, Pilsen*
Petr Cimprich, *Ginger Alliance s.r.o.*

Proceedings Typesetting

Vít Janota, *Ginger Alliance s.r.o.*

Preface

This publication contains papers to be presented at XML Prague 2005.

The conference is hosted at the Lesser Town Campus of the Faculty of Mathematics and Physics, Charles University, Prague. XML Prague 2005 is jointly organized by the Institute for Theoretical Computer Science and Ginger Alliance s.r.o.

XML Prague 2005 is focused on providing a broad yet detailed enough analysis of applied and theoretical XML topics. These topics are presented by avowed XML authorities over the course of a single-day; talks have been chosen across the spectrum of XML and related technologies.

We have strived to keep the conference academic in nature by keeping commercial sponsorship to a minimum. The organizers hope XML Prague 2005 shall be enjoyable for both audience and participants alike, providing a dynamic interchange and environment to discuss XML technologies.

Ginger Alliance

Open Minded Software

XML

Since 1999, Ginger Alliance develops XML-based products and solutions, and provides services. The company is in touch with the latest technologies, pioneering and innovating in many areas. Ginger Alliance combines both traditional environments such as web or local networks with emerging wireless technologies, pushing forward the notion of modern communication and data processing.

Technologies

Products developed by Ginger Alliance include Sablotron, the world's first open C++ XSLT and DOM processor, and PerlRDF, a complete toolkit to process RDF data. Ginger Alliance relies on open-source software. We have completed many successful projects using stable and proved building stones such as Linux, Apache, PostgreSQL or Mozilla. As an active contributor to the Open Source community, Ginger Alliance has released most of its products, including Sablotron and PerlRDF, under OS licenses.

Mobile

In 2002, Ginger Alliance has entered the rapidly growing market of mobile communications with GA GinDjin, its new flag-ship product. GinDjin is a digital content download platform and a gateway to mobile communication channels such as SMS, MMS or WAP. Its extensible and flexible design makes GinDjin the optimal choice for mobile operators and third-party content providers and aggregators.

http://www.gingerall.com

Program

9:00	*Opening*
9:10	XML Schema Languages Compared, *Eric van der Vlist*
10:10	*Coffee Break*
10:30	XML Processing Moves Forward: XSLT 2.0 and XQuery 1.0, *Michael Kay*
11:35	EXSLT: An Example of Facilitating the Standards Process, *James Fuller*
12:00	*Lunch Break (on your own)*
13:30	Binary XML and its Characterization, *Robin Berjon*
14:35	XML Experiments in Science and Publishing, *Miloslav Nič*
15:05	*Coffee Break*
15:25	Schema-aware XSLT processing, *Michael Kay*
16:10	Introduction to XML Editing Shell, *Petr Pajas*
16:45	Xdefinitions, *Václav Trojan*
16:45	Panel Discussion
17:30	*End of Day*

Produced by

Institute for Theoretical Computer Science
(`http://iti.mff.cuni.cz/`)

Ginger Alliance (`http://www.gingerall.com/`)

Sponsored by

DIMATIA (`http://dimatia.mff.cuni.cz/`)

Media Partners

ROOT.cz (`http://root.cz/`)

CHIP (`http://chip.cz/`)

Bulgaria WebDev Magazine (`http://spisanie.com/`)

OXYGEN xml editor (`http://www.oxygenxml.com/`)

XML Schema Languages Compared

Eric van der Vlist (Dyomedea)

What's a XML schema language?

Not what you're expecting!

The name "XML schema" language is misleading and they are not what you'd expect them in plain English.

Not always a representation of the class instances
In plain English, a schema is a representation of an object, however, some XML schema languages do not try to represent XML instance documents.

Not simpler than the documents
In plain English, a schema is a simpler representation of an object, however, XML schemas are usually more complicated than the XML instance documents.

XML schemas are:

If they're not schemas, what can they be?

Filters or firewalls
They can be considered as filters or firewalls that protect applications from the wide diversity of XML documents.

Transformations producing a validation report and/or an augmented infoset
XML schema languages can be seen as transformations taking instance documents as their input and producing a validation report and, in some cases, an infoset augmented with the information (type information, default values, ...) gathered during the validation.

Open and closed schemas

Two different approaches

Firewalls too can be either open or closed.

Everything which is not forbidden is allowed
For open schemas (or firewalls), everything which is not forbidden is allowed. An open schema is thus a list of rules defining what's forbidden.

Or everything which is not allowed is forbidden
For closed schemas, on the contrary, everything which is not allowed is forbidden. A closed schema is thus a list of rules defining what's allowed.

In practice, both are needed

In real world applications, we need to find a balance between openness and closeness. This is often done by opening closed schemas to add some of the extensibility which gives its name to XML, the eXtensible Markup Language.

One XML document out of ten contains at least one error

Figure given by two independent sources

This figure is frightening and it comes from two different authors working in two different domains.

Social security in Belgium
Reference: Isabelle Boydens *Informatique, normes et temps.* Bruxelles: Bruylant, 1999.

Banking systems (UK)
Reference: Simon Riggs, XML Europe 2003.

Can we afford this error rate?

Probably not!

Validation is not optional
If validation can decrease this rate we can't afford to ignore it and must do our best to improve its efficiency.

What kind of validation?

Decreasing the error rate in XML documents isn't an easy job and the nature of the tests to perform is very diverse.

Structure (imbrication of elements and attributes)

A first type of validation consists in checking the structure, i.e. the imbrication of elements and attributes. This validation acts at markup level and do not test the content of the text nodes or attributes.

Datatypes

A second category of validation consists in checking the content of text nodes and attributes independently of each other. With the exception of qualified names (QNames) in the content, an unfortunate practice that creates a dependency between the markup and the content, datatype validation ignores the markup and tests only the content of the document.

Integrity constraints

A third category of validation consists in checking identifiers to verify that they are unique and references to check that they refer to existing identifiers. Integrity constraints may be performed internally to a document or between documents (link checking).

Business rules

What's left after these three categories is often called business rules. Business rules can be as simple as checking that a date of death is, when it exists, greater than the date of birth or as complex as spell checking.

Short (and incomplete) history of schema languages

A long list starting before XML was invented...

Schema languages do not start with XML and, even though the clean distinction between features that are specific to validation and other features is relatively new, SGML had already its own schema languages.

DTD family

This first family takes its roots from SGML.

W3C XML Schema family

XML-Data (W3C note, January 98, by people from Microsoft, DataChannel, Arbortext, Inso Corporation and University of Edinburgh)

Includes (most of) the basic concepts developed by W3C XML Schema:

- internal and external entity definitions.
- the mapping with RDF and object oriented structures.

XML-Data-Reduced (XDR) (W3C note, July 98, by editors from Microsoft and University of Edinburgh)

Goal: "refine and subset those ideas down to a more manageable size in order to allow faster progress toward adopting a new schema language for

XML".
Implemented by Microsoft (Biztalk).

Document Content Description (DCD) (W3C note, July 98, by editors from Textuality, Microsoft and IBM)
"Subset of the XML-Data Submission; expresses it in a way which is consistent with the ongoing W3C RDF (Resource Description Framework) [RDF] effort".
No mapping but consistency with RDF.

Schema for Object-Oriented XML (SOX) (W3C Note, September 98 (2nd vers. July 99), by Veo Systems/Commerce One)
Very influenced by object oriented design (concepts of interface and implementation).
Influenced by the DTDs (parameters).
Widely used by Commerce One.

Document Definition Markup Language (DDML or Xschema) (W3C Note, January 99, XML-DEV edited by Ronald Bourret, John Cowan, Ingo Macherius and Simon St.Laurent)
"Encodes the logical (as opposed to physical) content of DTDs in an XML document".
Great attention paid to back and forward conversions back between DTD.
"Experimental chapter proposing Inline DDML elements".
Clear distinction between structures and data (left apart).

W3C XML Schema (W3C Recommendation, May 2001)
Acknowledges the influence of DCD, DDML, SOX, XML-Data and XDR.
Picked pieces from each of these proposals (compromise).
Proponents of living ancestors have all announced their support.
Should become the only living member of this family.

Relax family

RELAX (REgular LAnguage description for XML) (Published in March 2000 as a Japanese ISO Standard by MURATA Makoto)
Simple ("Tired of complicated specifications? You just RELAX!").
Solid (adaptation of the hedge automaton theory to XML trees).
Approved as an ISO/IEC Technical Report in May 2001.

XDuce (pron. "transduce") (First announced in March 2000)
Not a schema but a programming language.

Its typing system has influenced schema languages.

TREX (Tree Regular Expressions for XML) (James Clark, January 2001)
"Basically the type system of XDuce with an XML syntax and with a bunch of additional features".
A TREX schema reads almost as plain English.
Consistent treatment between elements and attributes.

RELAX NG (May 2001-December 2001, OASIS TC specification)
Now an ISO/IEC Standard (DSDL Part 2).
Merge between RELAX and TREX.
Coedited by James Clark and MURATA Makoto.

SCHEMATRON (PROPOSED IN SEPTEMBER 1999 BY RICK JELLIFFE)

Validation rules using XPath expressions.

EXAMPLOTRON (PROPOSED IN MARCH 2001 BY THE AUTHOR OF THIS PRESENTATION)

Example of instance structures used as schemas.
Borrows to Schematron and RELAX NG.

DOCUMENT SCHEMA DEFINITION LANGUAGES (DSDL)

TOO COMPLEX FOR A SINGLE LANGUAGE

Standing for "Document Schema Definition Languages", DSDL is a recognition that the validation of XML documents is a subject too wide and complex to be covered by a single language and that the industry needs a set of simple and dedicated languages to perform different validation tasks and a framework in which these languages may be used together.

Part 1: Overview
This is a kind of roadmap describing DSDL itself and introducing each of the parts.

Part 2: Regular-grammar-based Validation
This part is RELAX NG itself. It is a rewriting of the Relax NG Oasis Technical Committee specification to meet the requirements of ISO publications. Its wording is more formal than the Oasis specification but the features of the language is the same and any RELAX NG implementation conform the one of these two documents should also be conform to the other.

DSDL Part 2 is now a “Final Draft International Standard” (FDIS), i.e. an official ISO standard.

Part 3: Rule-based Validation
This part will describe the next release of the rule based schema language known as Schematron. Schematron has been defined by Rick Jelliffe and other contributors and its home page is

`http://www.ascc.net/xml/schematron/` .

Part 4: Selection of Validation Candidates
Although Relax NG provides a way to write and combine modular schemas, it is often the case that you need to validate a composite document against existing schemas which can be written using different languages: you may want for instance to validate XHTML documents with embedded RDF statements. In this case, you need to split your documents into pieces and validate each of these pieces against its own schema.

Part 5: Datatypes
The goal of this part is to define a set of primitive datatypes with their constraining facets and the mechanisms to derive new datatypes from this set and it is fair to say that it’s probably the least advanced and more complex part of DSDL.

Part 6: Path-based Integrity Constraints
The goal of this part is basically to define a feature covering integrity constraints.

Part 7: Character Repertoire Validation
This part will allow to specify which characters may be used in specific elements and attributes or within entire XML documents.

Part 8: Declarative Document Architectures
This part is still the most mysterious to me. The idea here is to allow to add information to documents (such as default values) depending on the structure of the document and the only input considered for Part 8 so far is known as “Architectural Forms”, an old promising-but-never-used-that-much technology.

Part 9: Namespace and Datatype-aware DTDs
There were plenty of good things in DTDs, especially in SGML DTDs and many people are still using them and do challenge the need to put them to trash and define new schema languages to support namespaces and datatypes. DSDL Part 9 is for these people who would like to rely on

years of usage of DTDs without loosing all of the goodies of newer schema languages. Despite a burst of discussion in April 2002, this part hasn't really advanced yet.

Part 10: Validation Management
Least but not last, Part 10 (formerly known as Part 1: Interoperability Framework) is the cement which will let you use together the different parts from DSDL together with external tools such as XSLT, W3C XML Schema or your favourite spell checker to come back to an example given in the introduction to this chapter.

Sample document

The following example will be used during the presentation:

```
<?xml version="1.0"?>
<library>
  <book id="_0836217462">
    <isbn>
      0836217462
    </isbn>
    <title>
      Being a Dog Is a Full-Time Job
    </title>
    <author-ref id="Charles-M.-Schulz"/>
    <character-ref id="Peppermint-Patty"/>
    <character-ref id="Snoopy"/>
    <character-ref id="Schroeder"/>
    <character-ref id="Lucy"/>
  </book>
  <book id="_0805033106">
    <isbn>
      0805033106
    </isbn>
    <title>
      Peanuts Every Sunday
    </title>
    <author-ref id="Charles-M.-Schulz"/>
    <character-ref id="Sally-Brown"/>
    <character-ref id="Snoopy"/>
    <character-ref id="Linus"/>
    <character-ref id="Snoopy"/>
  </book>
  <author id="Charles-M.-Schulz">
    <name>
      Charles M. Schulz
    </name>
    <nickName>
      SPARKY
    </nickName>
    <born>
      November 26, 1922
    </born>
    <dead>
      February 12, 2000
    </dead>
  </author>
  <character id="Peppermint-Patty">
    <name>
      Peppermint Patty
    </name>
    <since>
      Aug.  22, 1966
    </since>
    <qualification>
      bold, brash and tomboyish
    </qualification>
  </character>
  <character id="Snoopy">
    <name>
      Snoopy
    </name>
    <since>
      October 4, 1950
    </since>
    <qualification>
      extroverted beagle
    </qualification>
  </character>
  <character id="Schroeder">
    <name>
      Schroeder
    </name>
    <since>
      May 30, 1951
    </since>
    <qualification>
      brought classical music to the Peanuts strip
    </qualification>
  </character>
  <character id="Lucy">
    <name>
      Lucy
    </name>
    <since>
      March 3, 1952
    </since>
    <qualification>
      bossy, crabby and selfish
    </qualification>
  </character>
```

```
  <character id="Sally-Brown">
    <name>
      Sally Brown
    </name>
    <since>
      Aug, 22, 1960
    </since>
    <qualification>
      always looks for the easy way out
    </qualification>
  </character>
  <character id="Linus">
    <name>
      Linus
    </name>
    <since>
      Sept.  19, 1952
    </since>
    <qualification>
      the intellectual of the gang
    </qualification>
  </character>
</library>
```

Fact sheet: DTD

Basic facts

Author: W3C.
Status: Recommendation (XML 1.0).
Page: `http://www.w3.org/TR/REC-xml`
PSVI: yes.
Structure: yes.
Datatype: yes (weak).
Integrity: local with restrictions (ID/IDREF).
Other rules: no.
Vendor support: excellent.

Notes

- specific syntax.
- no support for namespaces.
- no support for content-sensitive content models.
- include other features (entities).
- reference to the DTD needs to be included in the instance document.

Sample DTD

DTD for our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT author (name, nickName, born, dead)>
<!ATTLIST author
  id ID #REQUIRED
>
<!ELEMENT author-ref EMPTY>
<!ATTLIST author-ref
  id IDREF #REQUIRED
>
<!ELEMENT book (isbn, title, author-ref*, character-ref*)>
<!ATTLIST book
  id ID #REQUIRED
>
<!ELEMENT born (#PCDATA)>
<!ELEMENT character (name, since, qualification)>
<!ATTLIST character
  id ID #REQUIRED
>
<!ELEMENT character-ref EMPTY>
```

```
<!ATTLIST character-ref
  id IDREF #REQUIRED
>
<!ELEMENT dead (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT library (book+, author*, character*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nickName (#PCDATA)>
<!ELEMENT qualification (#PCDATA)>
<!ELEMENT since (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Fact sheet: W3C XML Schema

Basic facts

Author: W3C.
Status: Recommendation.
Page: `http://www.w3.org/TR/xmlschema-0/`
PSVI: yes.
Structure: yes.
Datatype: yes.
Integrity: local (ID/IDREF, key, keyref).
Other rules: no.
Vendor support: potentially excellent, still immature.

Notes

- paranoiac about determinism.
- borrows ideas from OO design.
- considered complex.
- presented as a foundation of XML.
- controversial way of dereferencing namespace URIs.

Sample W3C XML Schema

W3C XML Schema for our example:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <xs:element name="library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"/>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="author-ref" minOccurs="0"
               maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:IDREF" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="character-ref" minOccurs="0"
               maxOccurs="unbounded">
                <xs:complexType>
```

```
                  <xs:attribute name="id" type="xs:IDREF" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="author" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="name"/>
              <xs:element name="nickName" type="xs:string"/>
              <xs:element name="born" type="xs:string"/>
              <xs:element name="dead" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="name"/>
              <xs:element name="since" type="xs:string"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
</xs:schema>
```

Fact sheet: RELAX NG

Basic facts

Author: OASIS RELAX NG TC & ISO DSDL WG.
Status: OASIS Specification, ISO/IEC Standard.
Page: `http://relaxng.org/`
PSVI: no.
Structure: yes.
Datatype: pluggable.
Integrity: through keys and key references.
Other rules: no.
Vendor support: improving.

Notes

- merge between RELAX and TREX: TREX based syntax including the full semantics of RELAX.
- two syntaxes are available

Sample RELAX NG (XML Syntax)

RELAX NG Schema for our example:

```
<?xml version="1.0"?>
<grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
 xmlns:="http://relaxng.org/ns/structure/1.0"
 xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <start>
    <ref name="library"/>
  </start>
  <define name="library">
    <element name="library">
      <oneOrMore>
        <ref name="book"/>
      </oneOrMore>
      <zeroOrMore>
        <ref name="author"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="character"/>
      </zeroOrMore>
    </element>
  </define>
  <define name="author">
    <element name="author">
      <attribute name="id">
```

```
        <data type="ID"/>
      </attribute>
      <element name="name">
        <text/>
      </element>
      <element name="nickName">
        <text/>
      </element>
      <element name="born">
        <text/>
      </element>
      <element name="dead">
        <text/>
      </element>
    </element>
  </define>
  <define name="book">
    <element name="book">
      <ref name="id-attribute"/>
      <ref name="isbn"/>
      <ref name="title"/>
      <zeroOrMore>
        <element name="author-ref">
          <attribute name="id">
            <data type="IDREF"/>
          </attribute>
          <empty/>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="character-ref">
          <attribute name="id">
            <data type="IDREF"/>
          </attribute>
          <empty/>
        </element>
      </zeroOrMore>
    </element>
  </define>
  <define name="id-attribute">
    <attribute name="id">
      <data type="ID"/>
    </attribute>
  </define>
  <define name="character">
    <element name="character">
      <ref name="id-attribute"/>
      <ref name="name"/>
      <ref name="since"/>
      <ref name="qualification"/>
    </element>
  </define>
  <define name="isbn">
    <element name="isbn">
      <text/>
    </element>
  </define>
  <define name="name">
    <element name="name">
      <text/>
    </element>
  </define>
  <define name="nickName">
    <element name="nickName">
      <text/>
    </element>
  </define>
  <define name="qualification">
    <element name="qualification">
      <text/>
    </element>
  </define>
  <define name="since">
    <element name="since">
      <data type="date"/>
    </element>
  </define>
  <define name="title">
    <element name="title">
      <text/>
    </element>
  </define>
</grammar>
```

Sample RELAX NG (Compact Syntax)

The same RELAX NG Schema using the compact syntax:

```
start = library
library = element library { book+, author*, character* }
author =
  element author {
    attribute id { xsd:ID },
    element name { text },
    element nickName { text },
    element born { text },
    element dead { text }
  }
book =
  element book {
    id-attribute,
    isbn,
    title,
    element author-ref {
      attribute id { xsd:IDREF },
      empty
    }*,
    element character-ref {
      attribute id { xsd:IDREF },
      empty
    }*
  }
id-attribute = attribute id { xsd:ID }
character =
  element character { id-attribute, name, since, qualification }
isbn = element isbn { text }
name = element name { text }
nickName = element nickName { text }
qualification = element qualification { text }
since = element since { xsd:date }
title = element title { text }
```

Fact sheet: Schematron

Basic facts

Author: Rick Jelliffe and other contributors.
Status: unofficial, will become ISO DSDL Part 3.
Page: `http://www.ascc.net/xml/schematron/`
PSVI: sort of (validation report can be customized).
Structure: not directly.
Datatype: not directly.
Integrity: not directly.
Other rules: yes.
Vendor support: improving.

Notes

- rule expressions only.

Sample Schematron

Schematron Schema for our example (partial):

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron"
 xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <sch:title>
    Schematron Schema for library
  </sch:title>
  <sch:pattern>
    <sch:rule context="/">
      <sch:assert test="library">
        The document element should be "library".
      </sch:assert>
    </sch:rule>
    <sch:rule context="/library">
      <sch:assert test="book">
        There should be at least a book!
      </sch:assert>
      <sch:assert test="not(@*)">
        No attribute for library, please!
      </sch:assert>
    </sch:rule>
    <sch:rule context="/library/book">
      <sch:assert test="not(following-sibling::book/@id=@id)">
        Duplicated ID for this book.
      </sch:assert>
      <sch:assert test="@id=concat('_', isbn)">
        The id should be derived from the ISBN.
      </sch:assert>
    </sch:rule>
    <sch:rule context="/library/*">
      <sch:assert test="name()='book' or name()='author' or
       name()='character'">
        This element shouldn't be here...
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Embedded languages

Schematron is a good candidate to be embedded in other languages
Schematron patterns can be included in `xs:appinfo` W3C XML Schema.
An "experimental validator" does the same with RELAX NG.
Schematron is a good fit since it doesn't overlap that much with the other languages.

Tool support

All these languages come with open source implementations that can be integrated in your applications.

Integrated support by tools and framework is (today):

Best: DTD.
Most promising: W3C XML Schema.
Challenger: RELAX, RELAX NG.
Niche: TREX, Schematron, Examplotron.

Features

Features supported by the schema languages:

XML Structure: DTD, W3C XML Schema, RELAX, TREX, RELAX NG, Examplotron.
Datatypes: DTD (weak), W3C XML Schema.
Integrity: DTD, W3C XML Schema, RELAX, TREX, RELAX NG.
Rules: Schematron, Examplotron.

Expressiveness

Ability to describe a wide range of structures

The ranking by expressiveness or flexibility (i.e. ability to describe a wide range of structures):

Most flexible: Schematron (but the authors need to define all the rules one by one).
Most flexible structural languages: TREX, RELAX NG, RELAX.
Challenger: Examplotron.
Behind: W3C XML Schema.
Less flexible: DTD (no support for namespaces).

Conclusion

There is no perfect XML Schema language, choose the best fitted for the job you have to do!

XML Processing Moves Forward: XSLT 2.0 and XQuery 1.0

Michael Kay (Saxonica Limited)

Abstract

We've been waiting for a long time for new raft of XML processing standards to be finished – XSLT 2.0, XPath 2.0, and XQuery 1.0. But they're now very close, and many people are using them even before they're finished. This talk is not so much a survey of what's in the standards, more a demonstration of their power and potential. It will also provide a comparison of the roles of each of the languages. But the key message is that these languages enable you to do a vast range of things that would previously have been tackled in low-level languages such as Java and C#. Together with other XML-based tools such as XForms and pipeline processing languages, these technologies allow you to build entire distributed applications: not just the presentation, not just the database access but the whole thing.

Introduction

On 4th April 2005 the second round of "last-call drafts" for XSLT 2.0, XPath 2.0 XQuery 1.0 and the rest of the family were published. The last call period has ended, and the number of comments appears to be manageable, which gives a good chance that the specs will become Candidate Recommendations this year, and full Recommendations in 2006.

This is a long gestation period: XSLT 1.0 and XPath 1.0 were published in 1999, as were the first working drafts of what eventually became XQuery. And as one might expect, the result represents a step-function in terms of the XML processing capability that's now available.

Most of the work between now and final Recommendations will be testing, to demonstrate that the specifications are implementable and that implementations are interoperable. W3C attaches increasing importance to this aspect of quality control.

In this talk I want to concentrate not so much on the language features that are present in the new specifications, as on the possibilities that they open up. This means also looking at how these specifications fit into the wider picture of what's going on in the XML world.

Much of that world is shaped by XML Schema and its competitors, and it's therefore good scheduling by the programme committee that I'm following on after Eric van der Vlist who I'm sure will have provided an excellent survey of this area. Despite the knocking that W3C XML Schema has received from some quarters, and the excellent alternatives available, W3C remains firmly committed to the specification, as do the big vendors such as Oracle, Microsoft and IBM who are providing most of the resource that drives the XSLT and XQuery efforts. One of the reasons for this is that XML Schema is the only one of the validation languages that attempts not just to classify a document as valid or invalid, but to attach type information to the document to make subsequent processing easier – and it's one of the aims of the new XSLT and XQuery specifications to exploit this information. I'll come back to say more about schema-aware processing in due course.

What's in the new raft of specifications?

The document set being published by W3C contains a bewildering number of modules, so it's as well to have a bird's eye view so you know what is defined where (I'll only give URLs for the first three, since the others are easily found by following cross-references):

- XSLT 2.0 (`http://www.w3.org/TR/xslt20/`)
  This is the new version of XSLT itself. The language has probably doubled in size, based on counting elements and attributes.
- XPath 2.0 (`http://www.w3.org/TR/xpath20/`)
  The new version of XPath. Like XSLT, it's probably twice the size of its predecessor.
- XQuery 1.0 (`http://www.w3.org/TR/xquery/`)
  The main language specification for XQuery. XQuery is in fact a superset of XPath 2.0, so this document contains a lot of material that duplicates the XPath 2.0 spec. In fact, the two documents are generated from a common source (using XSLT, naturally).
- XPath Functions and Operators

XQuery, XSLT and XPath share the same library of functions. This has grown from about 25 functions in XPath 1.0 to about 120 in XPath 2.0. While this seems a large increase, it's still quite a modest function library compared with other languages such as SQL or Java. Rather confusingly for many readers, this specification includes not only the user-visible functions, but also specifications of imaginary functions that exist only to define the meaning of the operators in the language such as "+", "=", and "|". A disproportionately large number of the functions seem to be concerned with handling of dates, times, and durations.

- Data Model
  XQuery, XPath and XSLT share a common data model and type system. Because this is common to all the languages, it has been extracted into a specification of its own. The data model for representing XML as trees of nodes hasn't changed all that much since XPath 1.0, except for the fact that nodes can now be typed: for example, an attribute that has been through schema validation can be annotated as an `xs:date`, which means that operations like sorting will work in the expected way for this data type. The other significant change in the data model is that the number of atomic types has increased: instead of the three types number, boolean, and string, the model now supports all the simple types of XML Schema, and sequences of atomic values as well as singleton values. This adds a great deal of power to the languages (though some people would have liked even more, notably sequences of sequences).

- XQueryX
  Some people don't like the fact that XQuery queries aren't XML documents. This makes it tricky, for example, to embed queries in an XML-based pipeline language. XQueryX provides an answer to this objection, by providing an XML-based encoding of the XQuery syntax. It's pretty indigestible, but some people think it's useful.

- Serialization
  The serialization part of XSLT (the description of `xsl:output` and its effect) has been taken out of the XSLT specification into a separate document, so that it can be re-used by other specs; this also helps to emphasize that serialization is not something that's properly considered part of XSLT transformation, it's a separate operation that happens afterwards.

- XQuery Formal Semantics
  The academics among you will be interested in the formal semantics; the rest of you almost certainly won't.
- Use Cases
  Provided for tutorial purposes
- XSLT/XQuery/XPath Requirements
  These documents were produced at the start of the process, and they are taken quite seriously when assessing whether the spec delivers what it promised.

There are also working drafts of facilities that won't make it into this round of standardisation but are under active development, namely an update capability for XQuery, and syntax for full-text retrieval.

Schema-Aware Processing

One of the big ideas that pervades these specifications is that of schema-aware processing. The idea is that a schema can be used as the type system for the language, and that stronger type-checking can be used to achieve two important benefits:

- query optimization.
- better error handling.

These ideas remain controversial. One reason for this is that schema-aware products haven't yet received wide exposure. The only schema-aware XSLT processor is the commercial version of my Saxon product, and because it's the commercial version, there are few people playing with it and fewer people talking about it than about the open-source, non-schema aware version. There are several schema-aware XQuery processors but again, in many cases they are commercial products rather than open-source. Another reason, of course, is that XML Schema itself is less than universally popular.

So far in Saxon I'm not using schema information for optimization, though I can think of some ways in which it might be possible. But I am using it for better error checking both at compile time and at run-time. At this stage I will show an example of this in action: since this is necessarily interactive, I'm afraid that those of you who missed the talk and are reading this in the proceedings will have to exercise your imagination. However, the key point is that many simple stylesheet errors which in the past might have been quite hard to debug, because they caused incorrect output to be produced with no error message, now result in either a compile-time error message or an explicit run-time message telling you where the incorrect instruction in your stylesheet is (so in a debugger, the editor can take you straight to the spot where you need to fix your code).

The specs have been carefully written so that schema-aware processing is optional. It's recognized that not all documents will have schemas, and that the weakly-typed approach of XPath 1.0 is still appropriate in many circumstances. What's more, the two styles can even be mixed, as befits a processing language for semi-structured data. I think that in practice, 90% of XSLT processing will continue to be untyped, but in the XQuery world, especially the world of XML databases, schema-based processing is likely to be very important. Basically, no-one knows how to search large XML databases in reasonable time without a schema to guide the process.

New capabilities in XSLT 2.0

There are two ways of looking at the collection of new facilities in XSLT 2.0. At one level, it's simply a collection of separate enhancements designed to respond to the problems where XSLT 1.0 users had most difficulties. However, I think one can look beyond that. There's a statement I like to quote from the start of the XSLT 1.0 specification:

XSLT is not intended as a completely general-purpose XML transformation language.

This statement is not interesting for what it says about XSLT, but for what it says about the intentions of the XSL Working Group at the time. I wasn't a member at that stage, but I know enough about the way this and other Working Groups operate to know that such a sentence indicates that a philosophical debate took place about the role and purpose of the language and this was the outcome. To my mind the outcome is less interesting than the fact that there was a debate: there were clearly some members trying to make XSLT general-purpose, and others who felt it should be designed primarily for the needs of rendering applications. The same tension can be seen in the provision of two synonyms for the top-level element, `<xsl:stylesheet>` and `<xsl:transform>`.

I think it's fair to say that in XSLT 2.0 the "general-purpose transformation" direction has won the day (and without a great deal of further debate). Many of the new features (grouping, regular expressions, functions, multiple output files, temporary trees) can be seen as supporting this view of the role of the language. There are new rendition features as well, for example the `format-date()` family of functions, but they are secondary.

This reflects the fact that people are using XSLT to do far more than rendition of XML documents, and they are hitting limitations in XSLT 1.0 when they do this. In fact, even in applications concerned with presentation there are limitations. One thing that was clear to me from the start, coming as I do from a database background, was that XSLT never benefited from any experience of report-writing languages. It comes from the SGML world of document rendition, not from the SQL world of data visualisation. In practice, people have done amazing things with XSLT, such as generation of SVG graphics to display raw data, and these have stretched the language severely.

Probably the feature that wins over most new users is grouping. Grouping is really hard in XSLT 1.0, and it becomes really easy in XSLT 2.0. Another big winner is regular expressions, though not for everyone, because some people find regular expressions very cryptic, and even for experts it's hard to get them correct. Together with the simple `unparsed-text()` function that allow a plain text file to be read, these facilities taken together make XSLT a powerful solution for most up-conversion applications, by which I mean applications that convert implicit structure in a document into explicit XML markup, by various forms of pattern recognition. For an example that illustrates this, see the paper I gave at XML 2004:

`http://www.idealliance.org/proceedings/xml04/papers/111/mhk-paper.html` .

Many people writing complex stylesheets soon find that the most powerful new feature is the stylesheet function. Stylesheet functions are very similar to named templates, except that they are called from XPath expressions rather than by means of a special `call-template` instruction. Although this might appear to be syntactic sugar, in practice it makes it much more feasible to wrap common bits of code into a function rather than repeating it. For example, I often use a function to express a relationship (get all the orders for this customer), and it's then possible to use this function as a virtual axis (`sum($cust/get-orders(.)/value)`) which means there's only one place in the code that knows how to navigate from a customer to the relevant orders.

Where does XQuery fit in?

If XSLT can do so much, why do we need XQuery as well?

You'll get different answers to that question from different people. Some would put it the other way round: they see XQuery as the answer to everything, and see no need for XSLT.

XQuery is in fact a functional subset of XSLT: there is nothing you can do in the XQuery 1.0 language that you can't do in XSLT 2.0. There are some things you can do more concisely, and that's the appeal for some people, but that would not be enough on its own to justify having a separate language.

The reason XQuery is needed is that XSLT is not suitable as a database query language. XSLT is simply far too powerful for that job. A database query language needs to be simple enough that the properties of a query can be analysed at compile time; you can't afford to simply let it loose on the database, locking shared data for hours while it computes a five-table join the hard way.

While the designers of XSLT focused originally on the requirements of rendering documents, the designers of XQuery focused on retrieving data. They were database people, and database people know how to do that; and when you're designing a language, you concentrate on solving the problems you understand. As a result, the FLWOR expression, which one can see as a re-engineered SQL SELECT expression, forms the heart and soul of the language. Many people in fact seem to write every query as a FLWOR expression, failing to realise that a construct such as

```
for $x in $input//customer where $x/location="UK" return $x
```

can be expressed rather more concisely in XPath 1.0 as

```
$input//customer[location="UK"]
```

XQuery is very often more concise than XSLT when expressing simple data extractions on a source document. Therefore, I don't think its only role will be in conjunction with XML databases. (If I thought that, I wouldn't have developed the XQuery front-end to Saxon.) For example, I think it often works well to extract the data needed by a Java application: XSLT can be over-the-top for that job. By contrast, there are many 10,000 lines stylesheets around to process complex XML vocabularies, and I find it hard to imagine we equivalents for these applications written in XQuery alone. Apart from anything else, XQuery lacks the mechanisms for polymorphism and inheritance exhibited by XSLT's template rules and import precedence, which become increasingly important as the size of the application increases.

The success of XQuery depends on the future direction of XML databases. At present, all the big three database companies are adding XQuery front-ends to their established relational products. I have to say that personally I find this slightly depressing; I think that information is naturally a network of relationships rather than a set of tables, and forcing everything into a model with flat tables at its heart feels to me like trying to extend the life of punched card tabulators. XML, to my mind, is an opportunity to break free of this model. I've seen many applications that can naturally be modelled in terms of a document workflow (where the document is something like an insurance claim) and it seems artificial to force the document into a relational view. By contrast, there are many applications that don't have a natural concept of a document, and in these cases a relational database seems perfectly adequate for the job without adding an XML layer.

Putting the application together

XSLT and XQuery are both component technologies: they only do one part of the job. Neither can be used to write an entire application. How, for example, do you capture information entered by the user in the browser?

Even in conjunction, XSLT and XQuery only do part of the job. I think there will be many applications that extract data from a database using XQuery, and then use XSLT to process it either for display or for sending to another application. But there's still something missing.

The two missing parts of the jigsaw, in my view, are forms processing and pipeline processing. For forms processing we have a specification in XForms, but relatively little experience of deployment. For pipeline processing there have been several proposals, and there are some good products (one of my clients achieves great things with Orbeon) but the ideas are not yet well entrenched.

In my view this is the answer to "what happens next?". With a good pipeline processing language to provide the binding middleware, and a forms engine to capture the user input, we have the basis of an XML fourth-generation-language that allows entire applications to be written, in XML, without any procedural code.

Pipeline processing is possible without a special language, and it's a design pattern I would highly recommend: break any complex processing task into a sequence of simple steps, each of which takes an XML document as its input and produces another XML document as its output. With a pipeline processing language to control the whole assembly, this becomes much more powerful. For example (and this brings me back full-circle to schema-aware processing) it becomes possible to define declaratively at which stages in the pipeline it is necessary to perform schema validation.

Summary

I've tried in this paper to avoid giving a catalogue of new features in the new set of W3C specifications. That's mainly because it takes too long to do, and partly because I've found it has a tendency to send people to sleep: it's a very long list. Instead, I've tried to look beyond the features at their significance: what do they enable you to do that you couldn't do before.

The direction that we're going in is towards writing entire applications in XML-based languages with no procedural code. We'll be using XML in the database, XML in the application logic layer, and XML in the presentation layer, and we'll be using XML in the middleware to put it all together.

And whether we like it or not, I have a strong feeling that in many cases we'll be using data description languages based on XML Schema to define and validate the data that's passing around these applications. The more complex the applications become, the more central a role the data definitions will come to play.

EXSLT: An Example of Facilitating the Standards Process

James Fuller (WebComposite s.r.o.)

Abstract

Influencing XML technology standards, in today's ***Rock*** *of commercial interests and the* ***Hard*** *place of Standards bodies like Oasis and the W3C is a nigh impossible task. The community based initiative EXSLT demonstrated that a small group of reasonably organised people can influence standards processes by having a realistic purpose; for EXSLT this was addressing deficiencies in the original v1.0 XSLT specification, highlighted by real world usage. By providing independent generation and verification of use cases along with functional definitions with (and sometimes without) solutions, EXSLT became adopted by XSLT processor implementators and end users. When it came to creation of XSLT 2.0, EXSLT adoption served as "long memory" and naturally influenced specifications such as XSLT 2.0.*

I will be giving a short history of the EXSLT effort, highlighting some of is successes and failures, and generally making the case for more of these types of "micro" efforts to facilitate the generation of future XML technology standards.

Background: 2000-2005

During the whole of the year 2000, developers were writing XSLT using processor specific extension functions which opened up a whole host of incompatibility issues. XSLT authors were finding themselves in the unenviable position of porting their code to multiple platforms or checking for which XSLT processor at runtime. Developers were clearly choosing their XSLT processor based on the their offer of extension functionality causing a significant increase in complexity and potential for failure for their stylesheets to work predictably across XSLT processors.

By April 2000 the W3C XSL Working Group recognized this problem, Stephen Meunch sent an email out outlining of the need for addressing extension functionality in XSLT, and vowed that the next future draft of XSLT, would be heavily influenced by developers concerns.

The idea of developing a standardized approach to extension functions was spookily voiced as early as March 2000; which I will equally attribute to Don Park and David Megginson. Don Park had introduced the concept at XTECH2000, whilst David Megginson, of SAX fame, argued via XML-DEV mailing list the need for standardising XSLT extensions as soon as possible.

Perhaps the vigorous debates that raged throughout the year led the W3C to react too quickly resulting in the doomed XSLT 1.1 Working draft, released in December 2000. In section 14.4 Defining Extension Functions, of this draft, it was proposed to include an `xsl:script` element to address embedding external scripting languages. To its merit it went some way in harmonizing extension functionality across XSLT processors through common language bindings but the same interoperability problems remained whereby stylesheets were still being made dependent upon the underlying processor's access to some scripting language. It was clear that additional functionality was needed in XSLT, as soon as possible, without resorting to escaping to other programming languages.

The next salvo in this debate came in the new year with a petition from a group of XSLT users which called for the striking of the onerous section 14.4 Defining Extension Functions from the current XSLT 1.1 Working Draft. This effort was contributed to by many XML-DEV and XSLT-list mailing list, with support initially rallied by the python XSLT implementator Uche Ogbuji with Dr. Jeni Tennison.

During the early part of 2001, there was much discussion on the issue on both the XML-DEV and XSLT mail lists. This is when Dr. Jeni Tennison, well known XSLT-list diva and author of a few great XSLT books, became involved by publishing a series of micro specifications which underpinned a framework of what would eventually be known as "EXSLT". Her knack of distilling many people's thoughts, opinions, and comments into something coherent and lucid was well demonstrated here, presaging her invitation to join the W3C XSL Working Group as an invited expert. The list below commented or gave input to these protean specifications follows:

1. David Carlisle
2. Jarno Elovirta

3. Joe English
4. Clark C. Evans
5. Jim Fuller
6. Dave Gomboc
7. Dave Hartnoll
8. Kevin Jones
9. Yevgeniy (Eugene) Kaganovich
10. Mike Kay
11. Steve Muench
12. Miloslav Nič
13. Francis Norton
14. Dimitre Novatchev
15. Uche Ogbuji
16. David Rosenborg

In March 2001, Dr.Tennison and a small subset (including Uche Ogbuji, Jim Fuller and Dave Pawson) of the above group then went on and tried to faithfully implement the bare bones of such a specification. This meant developing a website, along with creating a markup language to annotate new elements and functions, provide test cases and implementations where possible using XSLT 1.0 templates or a combination of common scripting languages e.g. using Javascript. Let me also say at this point that the guiding hand of Dr. Michael Kay was present; he had already implemented many of the "useful" functions that users needed already in his XSLT Processor SAXON some of which found their way immediately in the EXSLT definitions.

From March to April 2001, hundreds of email's were passed amongst us core EXSLT members, whereby we argued and debated the finer points. I would say that the process was informal, light on heavyweight concepts, and geared towards addressing the most serious problems first. By July 2001, the core EXSLT members had implemented or defined the core modules which defined a suite of functions and elements all of which were determined to make life easier for developers. Perhaps more surprisingly was the almost immediate implementation of many of the Module functions in the following XSLT processors:

1. 06/05/2001: Michael Kay's XSLT Processor SAXON 6.3
2. 08/05/2001: Uche Ogbuji announced the release of 4EXSLT
3. 11/06/2001: Johannes Döbler's jd.xslt XSLT

There were also significant contributions made at this time by Chris Bayes, Craig Stewart, Mike Brown including comments from the wider community

EXSLT was ready, by April 2001, to be discussed and ruminated on at the XSLT UK 2001 conference, the first conference solely dedicated to XSLT. Dr. Tennison even got a chance to give a short presentation of the effort and its goal. At the same conference, the W3C made the stunning announcement that XSLT 1.1 would remain a working draft and all efforts would be subsumed by a future XSLT 2.0 specification.

Over the years 2002 and 2003 EXSLT experienced what I would say low to moderate growth in terms of functional definitions and XSLT implementations, albeit high growth in terms of support amongst XSLT processors.

Since 2003 EXSLT through one route or another has some range of support over all the major XSLT processors. Even though the specification itself has lain fallow since 2003, significant implementations continue to emerge, like EXSLT.NET enabling .NET platform to use EXSLT.

By the date of this talk, myself and Uche Ogbuji will have revamped the entire EXSLT distro, aiming to discover EXSLT's place in the impending new XSLT 2.0 era. This new distro will simply fix all outstanding bugs and provide the foundation for new work.

So to summarise it's short history, EXSLT started as a reaction against the inclusion of an `xsl:script` element within the December 2000 release of the doomed W3C XSLT 1.1 Working Draft, and worries about the potential interoperability issues that could arise.

What is EXSLT?

Currently EXSLT consists of sets of defined functions and elements, segregated into modules, listed below:

- Common: Dramatically simplified stylesheets through providing a `node-set()`, `object-type()`, and `document()` (as per XSLT 1.1) function definitions.
- Date and Time: A full range of date and time handling functions.
- Dynamic: Enables dynamic evaluation of XPATH.
- Sets: Encapsulated functions which make it easy to perform set type operations on XML.
- Functions: Enabled the creation of functions.

- Math: Enables basic math operations within XSLT.
- Random: Added facilities to generate random numbers.
- Regular Expressions: Enabled regular expression handling.
- Strings: Contains string handling functions such as `tokenize()` and `find()` / `replace()`.

EXSLT provides a definition of each function or element, in XML markup, annotating its functional signature, providing example usage and tracking any changes that have occurred through its lifetime defining inputs and outputs. An example of such a definition is included below, which describes the `math:abs()` function from the EXSLT math module.

```
<?xml version="1.0" encoding="utf-8"?>
<exslt:function xmlns:exslt="http://exslt.org/documentation"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:html="http://www.w3.org/1999/xhtml"
 xmlns:vcf="http://www.ietf.org/internet-drafts/
 draft-dawson-vcard-xml-dtd-03.txt"
 version="1" module="math" status="implemented">
  <exslt:name>abs</exslt:name>
  <rdf:Description ID="math:abs">
    <dc:subject>EXSLT</dc:subject>
    <dc:subject>math</dc:subject>
    <dc:subject>abs</dc:subject>
    <exslt:revision>
      <rdf:Description ID="math:abs.1">
        <exslt:version>1</exslt:version>
        <dc:creator rdf:parseType="Resource">
          <vcf:fn>James Fuller</vcf:fn>
          <vcf:email>jim.fuller@ruminate.co.uk</vcf:email>
          <vcf:url>http://www.ruminate.co.uk</vcf:url>
        </dc:creator>
        <dc:date>2001-06-16</dc:date>
        <dc:description>
          <dc:description>
            <html:div>
              Returns the absolute value of the passed argument.
            </html:div>
          </dc:description>
        </dc:description>
      </rdf:Description>
    </exslt:revision>
    <exslt:revision>
      <rdf:Description
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/" ID="math:abs.1.1">
        <exslt:version>1.1</exslt:version>
        <dc:creator email="craig.stewart@nottingham.ac.uk" url="">
          Craig Stewart
        </dc:creator>
        <dc:date>2002-08-21</dc:date>
        <dc:description xmlns="http://www.w3.org/1999/xhtml">
          Added 4XSLT and libxslt implementation to the list.
        </dc:description>
      </rdf:Description>
    </exslt:revision>
    <exslt:revision>
      <rdf:Description
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       ID="math:abs.1.2">
        <exslt:version>1.2</exslt:version>
        <dc:creator email="craig.stewart@nottingham.ac.uk" url="">
          Craig Stewart
        </dc:creator>
        <dc:date>2002-11-12</dc:date>
        <dc:description xmlns="http://www.w3.org/1999/xhtml">
          Updated 4XSLT version to 0.12.0a3.
        </dc:description>
      </rdf:Description>
    </exslt:revision>
  </rdf:Description>
  <exslt:doc>
    <html:div>
      <html:p>
        The <html:code>math:abs</html:code> function
        returns the absolute value of a number.
      </html:p>
    </html:div>
  </exslt:doc>
  <exslt:definition>
    <exslt:return type="number">
      <html:div/>
    </exslt:return>
    <exslt:arg name="value" type="number" default="0" optional="no">
      <html:div/>
    </exslt:arg>
  </exslt:definition>
  <exslt:implementations>
    <exslt:vendor-implementation version="1"
     vendor="4XSLT, from 4Suite."
     vendor-url="http://4Suite.org"
     vendor-version="0.12.0a3" />
    <exslt:vendor-implementation version="1"
     vendor="libxslt from Daniel Veillard et al."
     vendor-url="http://xmlsoft.org/XSLT/"
     vendor-version="1.0.19" />
    <exslt:implementation src="math.abs.js"
```

```
    language="exslt:javascript" version="1"/>
   <exslt:implementation src="math.abs.msxsl.xsl"
    language="exslt:msxsl" version="1"/>
  </exslt:implementations>
  <exslt:use-cases/>
</exslt:function>
```

We first tried to supply a pure XSLT 1.0 solution for each function, in addition to providing language specific solutions which meant that we could try and hide invocation of a function through namespace binding and the use of either `xsl:call-templates` or direct call (e.g. `math:abs()`). This obviated the need for the developer to make their specific stylesheet incompatible, as the functions were hidden behind an XML namespace. To use XSLT 1.0 and language specific implementations all a developer had to do was declare the right namespace, import the specific function, as shown below. Even better is if the XSLT processor being used directly supports the particular EXSLT function, whereby just the namespace requires definition, as illustrated below:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:math="http://exslt.org/math"
 extension-element-prefixes="math">
  <xsl:template select="/">
    print out absolute value of 4.9999:
    <xsl:value-of select="math:abs(4.9999)"/>
  </xsl:template>
</xsl:stylesheet>
```

We hoped that XSLT processor implementators would quickly provide such support of EXSLT. Once a few XSLT processors supported some of the basic (Common and Function) EXSLT modules it was then also possible to start providing pure EXSLT solutions using the `func:func` and `func:result` to create reusable functions.

Some functions have no XSLT 1.0 or language specific implementation. The Dynamic module for example has some support only built into some XSLT processors with `dyn:map()` *having no known implementation.*

EXSLT informed XSLT 2.0

EXSLT went beyond addressing the "scripting" portability issue; module functions were defined in order of their impact on making XSLT lives easier, as well as providing valuable feedback to the XSLT Working Group for shaping XSLT 2.0.

Working with Result Tree Fragments

The `exsl:node-set` function returns a node-set from a result tree fragment, allowing for single stylesheets to perform multiple pass type processing. Many developers were tearing their hair out working around this “limitation” of XSLT 1.0. Specifically an RTF is what gets returned when using `xsl:variable` and since no further operation could be performed on an RTF it was necessary to convert to a first class node-set using third party functionality. The reason for such a limitation on `xsl:variable` was mainly implementator driven who argued that it would make their XSLT Processor development more difficult. Unsurprisingly most major XSLT processors already had their own extension functions for this, albeit with different signatures. EXSLT normalised node-set function usage, making stylesheets portable as well as to refactor complex transformations into sequences of simpler transformations. This approach was so successful that XSLT 2.0 eschews with the need of node-set function, by introducing the concept of sequences.

XSLT is not a programming language

XSLT 1.0 lacked a lot of functionality which one would expect from a programming language, though of course... it was and still is never meant to be a full blown computer language. EXSLT Date and Time and Math modules provided a lot of “missing” functionality for developers. A lot of this common functionality has been refactored out of XSLT 2.0 all together and placed into a separate specification all together, e.g. XQuery 1.0 and XPath 2.0 Functions and Operators; keeping XSLT focused on what it was meant to be, whilst providing developers with much needed functionality.

Abstracted complex XSLT

The EXSLT Set module was one of the few Modules which could be implemented fully using pure XSLT 1.0 templates. The templates themselves are quite arcane and a good example of XSLT that most new users have problems with. XSLT 2.0 does a great job at getting rid of this type of “black magic”. By encapsulating templates into callable functions, developers stylesheets become more readable and easier to understand. Early on in capturing XSLT 2.0 requirements it was obvious the need for simplifying grouping and set manipulation, which has reflected an host of features in XSLT 2.0.

Outputting Multiple Documents

XSLT 1.1 included some useful functionality, for example, in section 16.4 Multiple Output Documents it defined a method of allowing a single stylesheet transform to generate multiple documents. Having the ability to generate several output documents was something developers definitely wanted. Though with the August 2001 release of XSLT 1.1 stating that the W3C no longer supported that draft it was clear that this was a good function to be included into EXSLT. XSLT 2.0 supports this natively with the `xsl:result-document` instruction.

Evaluating XPATH

EXSLT also solved another nagging problem in XSLT 1.0, that of dynamically evaluating XPATH statements. The Dynamic module let developers create XPATH expressions from strings, allowing them to be constructed "on the fly". Controversially, XSLT 2.0 does not include this functionality, though the ability to use more sophisticated FLWR type expressions meant one didn't have to resort to dynamic evaluation techniques quite as often.

String Handling

XSLT 1.0 had anaemic string handling facilities; the EXSLT String and Regular Expression modules went some way to giving tools to developers to perform sophisticated string manipulation operations. XSLT 2.0 directly adopted regular expression handling and a whole spectrum of new string handling functions were defined for use by XSLT 2.0.

As you can see, these and the other EXSLT modules were great for problem solving, as well as imparting better portability.

What we got wrong

First off, let me say that these are my own opinions (not the rest of the EXSLT gang) and do not reflect upon any decisions made as a group.

No one uses Functions?

EXSLT main failure apart from inactivity over the past few years, is the relatively light adoption of using Functions... a main goal for EXSLT. I am unsure if this is related to not following up and implementing every possible EXSLT module fully with an alternate EXSLT `func:func` implementation or if the various EXSLT functions provided the "sweet" spot for what developers really needed. What became clear is that XSLT developers could do a lot more using EXSLT, and continue to use matching or named templates as primary reuse mechanisms.

There are quite a few power users of XSLT that implemented func-

tions using EXSLT, though mainstream developers rarely employ this reuse mechanism

In any event, XSLT 2.0 implemented functions though avoided making functions into first class citizens, that is they cannot be passed about. XSLT 2.0 could also get away without defining a `func:result` element by dint of the ability to associate a return type on the function element itself.

WE DIDN'T FOLLOW OPEN SOURCE CONVENTIONS

Procedurally EXSLT lacks a robust development environment; we have no issue tracking, very little in the way of automated testing and building of the distribution and no publicly available source repository. In hindsight, we would have been better served to have hosted the EXSLT domain somewhere like SourceForge. Currently we are addressing these issues and by the time of this talk will have added a new system providing such functionality.

There were some advantages to being so lightweight, and having an unusually verbose XML format describing both functions and modules, seemed to be enough in terms of tracking versions and changes. Though having just recently gone through the EXSLT email list to identify and track issues, I wish we would have just set things up properly from the beginning.

IMPORTING AND INVOKING FUNCTIONS

Because of the multiple implementations there were slight differences and problems related to importing and invoking functions. For example, developers wishing to use functions implemented an XSLT 1.0 template and not supported in their XSLT 1.0 processors they needed to use the `xsl:call-template` for invocation and `xsl:import` to make the stylesheet available. This was further compounded by having "mixed" examples whereby we would say use `xsl:call-template` and then have a test case directly invoking the function e.g. as illustrated below using the String module `str:tokenize` function:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:str="http://exslt.org/strings"
 extension-element-prefixes="str">
  <xsl:template match="a">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="*">
```

```
    <xsl:value-of select="." /> -
    <xsl:value-of select="str:tokenize(string(.), ' ')" />
    <xsl:value-of select="str:tokenize(string(.), '')" />
    <xsl:for-each select="str:tokenize(string(.), ' ')" >
      <xsl:value-of select="." />
    </xsl:for-each>
    <xsl:apply-templates select = "*" />
  </xsl:template>
</xsl:stylesheet>
```

This would throw an error on a XSLT processor that didn't natively support the String module. Though more importantly even if an `xsl:import` statement referring to the correct XSLT 1.0 implementation (in this case `str.tokenize.template.xsl`) it still wouldn't work because the user would have to change the invocation `str:tokenize()` to use `xsl:call-template`.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:exsl="http://exslt.org/common"
 xmlns:str="http://exslt.org/strings"
 extension-element-prefixes="str">
  <xsl:import href="str.tokenize.template.xsl"/>
  <xsl:template match="a">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="*">
    <xsl:value-of select="." /> -
    <xsl:call-template name="str:tokenize">
      <xsl:with-param name="string" select="string(.)" />
      <xsl:with-param name="delimiters" select="' '" />?
    </xsl:call-template>
    <xsl:call-template name="str:tokenize">
      <xsl:with-param name="string" select="string(.)" />
      <xsl:with-param name="delimiters" select="''" />?
    </xsl:call-template>
    <xsl:variable name="holdresult">
      <xsl:call-template name="str:tokenize">
        <xsl:with-param name="string" select="string(.)"/>
        <xsl:with-param name="delimiters" select="' '"/>?
      </xsl:call-template>
    </xsl:variable>
    <xsl:for-each select="exsl:node-set($holdresult)" >
      <xsl:value-of select="." />
    </xsl:for-each>
    <xsl:apply-templates select = "*" />
  </xsl:template>
</xsl:stylesheet>
```

If this wasn't confusing enough for developers there were many other problems related to invocation and importing. Since we had many different language implementations Module and function level imports would routinely not work because the import stylesheets provided didn't do any checking for the existence of the function and at worst the processor would throw an error because it didn't know how to handle an element. This behavior is demonstrated below, once again using `str:tokenize`. EXSLT `str.tokenize.xsl` attempts to import all XSLT 1.0 and language specific (Javascript and MSXML), in turn the module import `str.xsl` would import this stylesheet.

```
<?xml version="1.0" encoding="utf-8"?>
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:func="http://exslt.org/functions"
 xmlns:str="http://exslt.org/strings"
 version="1.0"
 extension-element-prefixes="str"
 str:doc="http://www.exslt.org/str">
  <import href="str.tokenize.function.xsl"/>
  <import href="str.tokenize.template.xsl"/>
  <func:script language="exslt:javascript"
   implements-prefix="str" src="str.tokenize.js"/>
  <func:script language="exslt:msxsl"
   implements-prefix="str" src="str.tokenize.msxsl.xsl"/>
</stylesheet>
```

Using Michael Kay's version 6.5.3 SAXON processor we should be able to use the XSLT 1.0 implementation, though if we import the above stylesheet (`str.tokenize.xsl` instead of `str.tokenize.template.xsl`) we would get an error because SAXON has no idea of how to handle the script element! What a mess this was for developers just trying to use the various implementations, it became a standard FAQ to this day to just import the "specific" implementation. Clearly we had not tested or thought through the implications of importing and invocation enough and I believe this has made using EXSLT unduly complicated for new users.

There was little we could do with making users choose between `xsl:call-template` or function signature, though by not having the ability to generate specific distributions it became impossible for users to import the whole EXSLT library. Though this conflicted with the idea of portability... another issue all together.

I have a few other concerns with EXSLT, though these are directly related to the ever present constraints in terms of time and energy. I have simply listed them below:

1. Dependencies: Some XSLT 1.0 implementations of functions depen-

ded upon the existence of other functions. This was yet another headache for developers to contend with and we could have done more informing developers and managing dependencies.

2. Function Submission: By choosing a heavyweight XML format for defining functions it was hard for the wider community to contribute especially with the addition of RDF with relatively little benefit. I argue later on in the paper that this reluctance to add all manner of functionality actually turned out to be a "good thing".

3. Missed Opportunities: Over the years there have been other extremely high quality efforts; XSLT Standard library by Steve Ball and Dimitre Novatchev FXSL. I believe that integrating and aligning EXSLT efforts with these and others would have served the XSLT community better. FXSL demonstrated that many functions could indeed be created using pure XSLT 1.0.

4. Licensing: Currently EXSLT doesn't have a clear licensing model which needs to be addressed.

5. Namespaces: Absolutely required though having one namespace would have been just as appropriate.

Additionally, the EXSLT website is out of date and requires a good "scrubbing". This is common with Open Source projects which live and die based upon Darwinian principles of necessity and fitness, as well as how much time contributors have to donate. I hope that we will have addressed this by the time of the conference.

Where things went right

EXSLT must of have got some things right, as attested to the number of implementations with most every environment having access to some of the more common EXSLT functions.

Some XSLT processor implementators have chosen to provide a "cheap" route to EXSLT by dint of access to Javascript processing and using the javascript language implementations of EXSLT function.

Something for everyone

By providing immediate functionality to developers, definitions and test cases to implementators, as well as feedback to the W3C XSL Working Group EXSLT had something for everyone in XSLT community. Developers could at the least learn from XSLT 1.0 implementations, whilst implementators could refer to a specific part of the EXSLT website for them. The W3C XSL Working group benefited from gaining insight and feedback on the success or failure of any specific EXSLT module or function.

Multiple implementations meant no bias for any particular computing environment, though we were able to avoid Java and Python almost immediately due to implementator support in these environments.

The real catalyst for adoption was the willingness of the XSLT Implementators (Uche Ogbuji and Michael Kay) to quickly deploy versions of their software which supported the core functions.

Enabled XSLT 2.0 to take its time

If EXSLT hadn't been in existence would we have waited 5 or 6 years for XSLT 2.0? At the least we probably would have much greater interoperability issues, if not just for the existence of `exsl:node-set()`. Irregardless of the valuable feedback EXSLT provided to XSLT 2.0, it can be argued that it allowed the W3C XSLT Working Group to drop XSLT 1.1 and do some major refactoring resulting in a comprehensive suite of specifications.

Linking of W3C XML Standards meant that it was going to take a much longer time to negotiate issues between the various specifications. I think this approach in the end has somewhat surprisingly resulted in an XSLT 2.0 that many people are happy with.

It obviously didn't hurt to have a few developers of XSLT processors closely involved with EXSLT, as well as a few W3C experts. And in light of the considerable problems (and espousing an well worn aspect of the Agile methodology) I consider the people of EXSLT, instead of the processes underpinning EXSLT, most important to influencing XSLT 2.0 development.

Future: Open Source Specification?

We have standards bodies that promote Open Standards, but the process by which these standards are generated are far from "open" in process. The future of specifying standards could take a page from software development itself. Over the past years Open Source development by small tribes of developers have proven extremely useful. Why not apply the same methods in generating standards?

It is not such a stretch to see a SourceForge type approach to generating standards; though what is needed is the right kind of tools that allow a disparate group to collaborate on drafting such standards. Such an activity would require issue tracking, versioning, and perhaps an agreed upon format for the standards themselves.

By the date of this talk I will have implemented a prototype of such an approach based upon combining a Wiki with a predefined standards schema that allows users to comment and track issues. This can be found at

`http://www.workingdocument.com` .

My work on the `http://www.workingdocument.com` prototype has led me to believe that there will always be problems with such approaches especially if we avoid formal systems of describing specifications and the languages they spawn e.g. using context free grammers. Technology can help here, but can't replace or augment experience of the "right" way to describe something.

Ultimately, I suspect that these approaches, if purely theoretical in nature and devoid of implementation would suffer. Perhaps what EXSLT has taught us is to leave the heavy lifting to such organisations as OASIS and W3C, with micro specifications focusing on generating use cases, requirements gathering, tests, prototyping, and enabling tactical communication across early implementations.

Conclusion

Why was EXSLT moderately successful, especially in an environment that is strife with endless debate on permatopics?

Firstly all success needs a bit of luck, in EXSLT case it was the quality of such individuals as Dr. Jeni Tennison, Uche Ogbuji being involved, lets also not forget Dr. Michael Kay efforts here as well. Even more effective was that some of these individuals acted as a "bridge" between the developer community and the recognized standard bodies, funnelling requirements and use cases where appropriate.

Secondly, I think there was a bit of synchronicity in the fact that David Pawson, XSLT FAQ maintainer, arranged XSLT UK 2001 just at the right time to get the EXSLT message out. We are all used to virtual relationships these days and tend to forget the importance of face to face contact.

EXSLT had identified early on that implementators were their first "customer". By providing a range of test cases, well defined function signatures, and documentation we made it easier for implementators to "plug-in" support to their XSLT processor development.

Alternately, by providing developers with immediately usable XSLT 1.0 solutions for some key functionality EXSLT was able to get some visibility with developers. My own personal opinion is that EXSLT was successful because it had a tight constrained scope, solving some simple problems for both user and implementators alike.

Adoption was driven by the offering of multiple implementations; A user could choose language specific (Javascript, MSXML) or pure XSLT (where possible) solutions, or alternately could just choose a processor that supported the function or element. At a minimum a developer could learn how to do something through introspection, at a maximum stylesheets that used EXSLT were much more portable across XSLT processors.

Uniquely, and perhaps unknowingly, EXSLT had also identified the groups writing specifications as a possible consumer, benefiting from developer and implementator feedback to EXSLT. The incredible work done with the latest XSLT 2.0 draft specifications reflects this; it seems that the W3C XSLT working group was able to negotiate the vagaries presented by introducing linkage to other W3C XML specifications (XML Schema comes to mind) balanced with constraining XSLT to its original purpose. So far XSLT 2.0 seems like a measured and well thought through effort balancing some very tricky and contentious issues; even some XSLT old hands are calling it a "powertool".

It remains to be seen if EXSLT will have any relevance in the future, especially with XSLT 2.0. Here are a few ideas where XSLT might have some future resonance:

1. Repository for XSLT 2.0 defined functions
2. XSLT simple type system
3. New Modules: URI, HTTP, SOAP, SQL ?

We are hopeful that the new distribution will serve as a foundation for any future work, though it is quite possible for EXSLT to die a natural and peaceful death having fulfilled its original purpose.

There are many existing standards out in the XML wilderness today, that could benefit from being gently nudged...

1. XLink
2. XML Schema
3. Binary XML
4. What to do with XML Namespaces
5. SOAP/WSDL/UDDI

EXSLT existence should illustrate that small groups can make a difference, though I would argue that it was an effort borne by the entire XSLT community as a whole. Perhaps it is only through loose cooperation that these type of “white elephant” problems can be solved; which gives something for developers to use immediately, implementators to follow as guidelines, and feedback for those groups writing specifications.

Bibliography

Web References

Petition to withdraw `xsl:script` *from XSLT 1.1*
`http://www.oasis-open.org/cover/withdraw-xslScript.html`

References

EXSLT `http://www.exslt.org/`

XML `http://www.w3.org/XML/`

XSLT `http://www.w3.org/TR/xslt`

XPath `http://www.w3.org/TR/path`

XSLT 2.0 Requirements `http://www.w3.org/TR/xslt20req`

XSLT Standard Library `http://xsltsl.sourceforge.net/`

FXSL `http://fxsl.sourceforge.net/`

Michael Kay’s *SAXON XSLT Processor* `http://www.saxonica.com/`

Uche Ogbuji *Python 4Suite* `http://4suite.org/index.xhtml`

XQuery 1.0 and *XPath 2.0 Functions and Operators*
`http://www.w3.org/TR/xquery-operators/`

Binary XML and its Characterization

Robin Berjon (Expway)

Introduction to Binary XML

What is Binary XML?

The first thing to note about "Binary XML" is that, despite its name, it quite simply is not XML. An XML document is a document that conforms to the syntax defined in either the XML 1.0 or 1.1 specifications, usually modified to match the requirements expressed in the Namespaces in XML specification. Any document which does not match the productions of that syntax is not an XML document. It may contain information that corresponds to one of XML's many data models, it may be straightforwardly convertible into an XML document, but it still is not XML.

However, there is a number of specific properties to "Binary XML" formats that separates them from arbitrary bags of random bits. A general definition of what exactly is a Binary XML format is a complex task as there are many different approaches to this problem, and as we will see later the basic requirements that the XML Binary Characterization Working Group produced do exclude a fair number of existing solutions. However, such formats all have in common the fact that their processors expect either an XML document or an instance of one of XML's data models (Infoset, SAX, DOM, PSVI, XPath 1.0 Data Model, XPath 2.0 Data Model, etc.) as input to convert into the binary XML format, and conversely will produce a document or data model instance back from the format. They may lose some information in the original XML (eg. comments, processing instructions) but tend to provide some reasonable (for the use cases they were designed for at least) round-tripping of information between an XML document and the binary XML format. In short, binary XML formats are not XML, but they hold a strong relationship to XML.

The question of whether the name “Binary XML” is therefore justified or not has occupied the minds of many in the community, often leading to heated debate from both sides. A number of alternatives have been proposed to attempt to soothe the atmosphere (binary infosets, efficient encoding of XML, etc.) but all have issues of their own, and no matter how smart and perfect a new name may be, people will still call it “Binary XML”. So for the purpose of this discussion, we shall stay with that denomination and leave the name game up for idle moments.

What is the debate about?

Whoever has been paying even distant attention to the debates that have animated the XML community over the past five years will have noticed that binary XML is one of the hottest and most regular subjects of contention. How is it that what at first seems to be a simple optimization that some people may need leads to such heated discussion? The primary reason is that both sides tend to use completely wrong reasons to justify their positions, and that those drown the actually reasonable ideas that could lead to saner and more peaceful discourse. In a nutshell, here are some of the bad and good reasons that are used. We won’t go into too much details, hopefully this paper will help you make up your own mind.

Pro (bad). XML is inefficient no matter what you do with it; text is always slow and a waste of resources; any data format that isn’t immediately optimal is useless; data formats should never be human readable; XML is too complex and over-engineered.

Pro (good). There are strong use cases for it (see below); it increases the universality of XML; it simplifies the processing stack when a binary format is needed by making XML’s flexbility available there.

Con (bad). Just use gzip; all binary formats are evil; every binary format is proprietary; it is totally impossible to obtain the gains claimed by the proponents of binary XML; Moore’s Law will save mankind from all of its problems and lead to world peace.

Con (good). There are serious interoperability issues to consider; the feasibility of general binary XML still needs to be proven; a multiplication of formats would be problematic; it decreases the universality of XML.

In which universe is XML universal?

As you can note from the arguments depicted above, there seems to be two diverging opinions on the universality of XML. This stems from the fact that the universality of XML can be perceived in two subtly different manners: either you think that XML is everywhere that it *can* be, and therefore it is universal and ubiquitous; or you may think that XML is not everywhere that is *could* be, and therefore it is neither universal not ubiquitous.

This may seem like a metaphysician's distinction, but it is at the heart of the reason why binary XML exists in the first place, and why it is rejected by some. If your world is made of servers and desktops, all you see everywhere is XML and pretty much everything just works fine. If however your world is one of mobile or constrained devices and you're trying to bring the services that people want there, you cannot fail but notice that there isn't enough of XML available on such platforms, and that it needs to be brought into those spaces to avoid a highly undesirable split between mobile and fixed devices. We look at some of those use cases in the following section.

Why Binary XML – Some Use Cases

Mobile Web and Mobile Services

Not long ago the number of mobile devices overtook that of "fixed" ones. And naturally, there is strong demand for more advanced services and for mobile Web usage that would make use of the increased power available in such devices. However, to this date the various attempts at providing that have only been partially successful.

The mobile industry learnt the hard way that mobile-specific solutions would simply not take off. The whole WAP fiasco with its mobile-only markup language (WML) and its mobile-orientated binary format (WBXML) tought them that viable mobile solutions need to also work on fixed devices, and that separating the two worlds only leads to failure.

However the approach of making what is available to desktops on mobile devices as is has not worked really well either. For instance, SVG mappping applications (one of the most strongly desired services) don't work as well as they should. The typical map containing enough interesting information to be useful weighs a minimum of 100K and takes about 10 seconds to parse and render on a middle-tier device, not to mention issues that may arise with bandwidth consumption. Likewise, mobile applications that require frequent updates or requests (using for instance Web Services, or getting their information from a broadcasted feed) tend to be very slow on even high-end devices.

It must noted at this point that waiting for Moore's Law to apply to the mobile industry simply will not work. One could put faster processors into existing devices, but they would drain the batteries much faster, and it is unlikely that users would be happy with a mobile phone that they would have to recharge several times a day. Furthermore, even if battery technology took a massive leap forward, simple rules of physics would produce a final barrier. Even the most energy efficient processors that we can conceive give off large amounts of heat, and there is a strong limit on how much heat can be generated from a mobile device. If you have used some of the recent 3G devices, you will have noted that they occasionally become too hot to be stored in one's pocket after usage – double the CPU usage and they will be simply too unsafe to sell.

This is where binary XML steps in. By drastically decreasing the number of CPU cycles required to obtain a given information item, services that were previously impossible become much easier to implement and deploy. In some cases, costly 3G networking can even become unnecessary.

Documents

If you have worked on publishing systems that involved an XML workflow and brought PDF into the mix, you will likely have found that it is more complex than one would expect. Producing PDF as a final step is manageable, as is gathering data from a PDF form, but anything beyond that requires either large amounts of complex programming, or large amounts of money spent on one of Adobe's enterprise systems.

It is therefore not surprising that there is a strong push to convert PDF to use XML instead of its current format. However, doing it naively doesn't work. A 10,000 page manual (and sometimes twenty times that) as are used in some industries does not scale well when XML is used. In order to skip to a page the entire document has to be parsed. Adding a page in the middle of it requires lenghty processing that even on a powerful terminal will take too long for an end user to find viable. Embedding large binary blobs will not work nicely with XML either.

There is always the solution of providing a two-way mapping between PDF and an XML format, but that would amount to no less than an ad hoc binary XML solution, and it wouldn't be as extensible as the needs of modern electronic documents require it to be. The advantage of using a binary XML format here is that random access and random update can be largely simplified, and that binary data can easily be embedded in it, all the while retaining the features that make XML a really good fit for electronic documentation. Note that while I use PDF here as an example that everyone knows, the same issues apply to all other document formats.

Messaging and Web Services

For a large set of the situations in which Web Services are used today, there is no need for binary XML. Either because the volume of messages exchanged is too low, or because the time spent parsing XML is really small compared to other parts of the processsing stack, the meager gains brought by more efficient XML interchange are not worth the additional complexity of dealing with two interchange formats – especially as Web Services already have other interoperability issues to deal with.

But at the more real-time and lightweight side of the spectrum, use cases exist that strongly benefit from binary XML. For instance, a number of systems are being built on top of Jabber/XMPP to perform real-time processing of simple and short commands, or in some others cases to handle thousands of requests per second due to the sheer number of clients connected to a common server, and experiments with binary XML in those areas have proven conclusive.

A Few Notes

This presents only a small subset of the use cases that have been identified, trying to pick ones that are different in nature to show not only that the needs span multiple domains, but also that finding one single binary XML solution that would address these varied problems is not as obvious a task as one may think when scratching the surface.

It is also important to note that one shouldn't be fooled by the apparent verticality of these example use cases. While it would seem at first that one could devise a solution for each of them independently of the others and not bother trying to produce a generic format, a little further thought makes it clear that they cannot be separated. Mobile devices will be using Web Services, messaging, and documents in communication with desktops and servers. Documents are sent as the XML payloads in Web Services in enterprise publishing workflows. Messaging is used from any form of terminal. The tendency of Web and XML technologies is towards higher integration, not the reverse, and while mixes between these use cases and others already exist, they will only increase over time.

How to Define the Problem – Characterizing Binary XML

With those use cases already known at least in part, and discussed during a workshop on efficient XML transmission that the W3C held in August 2003, and taking into account many concerns that had been expressed around the idea of binary XML, the XML Binary Characterization

Working Group (or XBC WG for short) was chartered for a year starting in March 2004 to provide means to better define the binary XML problem, and decide on whether a single solution spanning multiple industries was possible. There is insufficient space here to go into a detailed description of the four documents it produced, but the following short introductions should be enough to get one started into reading them:

- **XBC Use Cases**
  (`http://www.w3.org/TR/xbc-use-cases/`)
  This document covers some of the ground that we have visited in the previous section, but goes into much more detail to describe the precise situations in which binary XML is believed to bring some advantages, and what exactly these different situations require from a binary XML format. It must be noted that not all of these use cases are actual use cases for a generic binary XML format – some may be niches that don't need a standard in that domain, others may be using XML happily most of the time and don't really need the optimizations of efficient XML interchange.
- **XBC Properties**
  (`http://www.w3.org/TR/xbc-properties/`)
  In order to speak coherently and compare correctly multiple formats between themselves and against XML, a common, well-defined terminology is needed that defines which properties can be found in a given format. There are many different properties that were found to be useful, some of them simple (Human Readable and Editable, Platform Neutrality, Transport Independence...) others more complex (Processing Efficiency, Explicit Typing, Generality...), and they are explained in detail in this document. Note that this document does not list requirements for a binary XML format – it would amount to too many of them, some of which are contradictory – but rather defines the properties and leaves the task of narrowing them down to a set of requirements to the last document in the series.
- **XBC Measurement Methodologies**
  (`http://www.w3.org/TR/xbc-measurement/`)
  It is a good start to have common terminology in the form of properties, but in order to agree on whether a given format supports a given property one needs a way to measure it. This document defines for each property a way in which it is to be measured. Some of the measurements are very straightforwards but others are highly technical. If you are not about to try to measure an actual format to see how it

fares, I recommend that you don't spend too much time reading this document and only use it for reference when you need to see how a property is implemented concretely in a format.

- **XML Binary Characterization**
(`http://www.w3.org/TR/xbc-characterization/`)
Last but not least, this document provides a synthesis of the groundwork performed by the previous ones. If your interest in the topic is casual, you may wish to read only this one and refer to the others in cases where you feel something requires more details.
The first part of the characterization is isolating requirements that a binary XML format would need to have to be successful and cover as many use cases as possible. This starts off by listing which requirements come from the fact that a format derives from XML and is produced by the W3C (Royalty Free, Human Language Neutral...). Then it looks at how many of the use cases require each property, which produces a list of requirements too long to be useful or realistically implemented. However upon closer inspection it appears that not all of those features need to be supported natively by a binary XML format, but rather that they should be defined to be generic and apply equally to XML documents, and that the binary XML format only needs to be created in such a way that these properties can be implemented efficiently on top of it. The way that these are filtered out is by passing each required property through a simple decision tree (in section 5) that sorts them easily on either side.
The result of this requirements process is exposed in section 6, where one can see that the final list is reasonably short. Also, when looking closely at it it becomes clear that most of these properties are already properties of XML, with extra compression and speed − this was not done on purpose, and simply reflects the fact that the "XML" part of binary XML is strongly needed and not there simply to benefit from the XML hype. The list of requirements is then compared to the features that are available in some of the existing binary XML formats (which have been anonymized for IP and secrecy issues that will be lifted if a Working Group starts work on a W3C binary XML format). The fact that one existing format supports all the properties and that several come close to it shows that a format addressing these requirements is feasible.
Finally, the document concludes that the work done has shown that binary XML is both needed and feasible, and therefore that the W3C needs to produce work in this area.

Is Binary in the Future of XML?

After several years working in the field of binary XML, and one year spent chairing the XBC WG, it is our opinion that binary XML is, sadly, unavoidable. We say sadly because no one in their right mind would want to have to deal with the incurred extra complexity if there were ways to avoid it. However the use cases are very compelling, and will not work without a binary XML solution.

The W3C has not yet made its final decision to charter a working group to defined a binary XML format, but that decision will come soon. A lot of how binary XML will happen in the close future depends on that decision, keeping in mind that people will use binary XML even if the W3C says it shouldn't be done. At the two extremes I can see the following scenarios:

- **Worst case scenario**
  There is no W3C standard. People produce a large amount of vertical ad hoc binary XML formats because off-the-shelf solutions are too costly or inappropriate for some given needs. There are three to five major competing implementors of either proprietary solutions or standards produced by organizations not recognized by the XML industry as a whole competing on the market and not interoperable between one another. Users pay (in complexity or in money) and the lack of interoperability makes a large set of services stagnate for a long period of time.

- **Best case scenario**
  There is a W3C standard, done right (and based on the work produced by the XBC WG, we have hope that it will be done correctly and not turn into a huge monstrous specification). A number of niche formats for areas that the standard does not apply to and that don't require interoperability with other domains remain in existence, but that is perfectly fine – in fact if the W3C standard tries to make absolutely everyone happy the format will end up being too complex to be useful. Users and services can use binary XML transparently thanks to a strong integration into the XML family of specifications.

It is of course highly likely that the final outcome will end up being somewhere between those two extremes, but there is hope that if we work with the latter as a goal instead of trying to stop the advent of binary XML when in fact it is already deployed and will only be deployed more, then the damage will be kept to a minimum.

XML Experiments in Science and Publishing

Miloslav Nič (ICT Prague)

Introduction

Science without exchange of information is an oxymoron. Its development is very tightly coupled with availability of distribution channels and appearance of important new channels dramatically changes scientific landscape. In the lecture our contributions to this information exchange will be summarized to demonstrate that in the age of Internet limited resources are not unsurmountable barriers to sizable contributions.

Zvon.org [`http://zvon.org`]

The Zvon site is a multilingual educational site based on XML technologies. It contains tens of thousands of heavily crosslinked pages, and is available in several display modes. It offers several types of searches, and yet can be run in a standalone mode on an off-line computer.

Zvon was founded by a group of people who felt that free information exchange can open new horizons and can be very beneficial for people of similar ideas and needs. Even though Zvon has grown in various ways its original idea is not altered.

In the center of its attention stands XML. The basic idea behind XML is really simple. Its supporters including us believe that this human readable, universal and easily comprehensible format provides scaffolding around which the information architecture of 21st century can be built.

Zvon has two relevant meanings in Czech. Firstly, it means a bell, one of the first information devices. And there is the second meaning, little humorous but even more fitting. Zvon is used as a name for a plunger, a device used to clear clogged pipes. XML and other techniques demonstrated at Zvon are tools that help clean information channels. Information pipes which connect people are filled with ballast of incompatible formats and programs which can be repaired only by an experienced plumber. We believe that the site helps to make this cleaning more efficient and affordable. With 10-20,000 visitors the pipes are hopefully getting cleaner.

ICT Press [`http://vydavatelstvi.vscht.cz`]

ICT Press publishes a number of scientific books and other publications in Czech and English. Publishing of mathematical and chemical texts is very demanding and ICT Press is at this moment the only publisher in the Czech Republic which regularly publishes in chemistry domain. Its in-house expertise was leveraged by introduction of XML technologies.

ICT Press published over the years many chemistry textbooks. They sources are available in many legacy formats. Last year we have created a new environment enabling full text searching and browsing of these materials:

`http://vydavatelstvi.vscht.cz/katalogy-e/katalog_e-books_cze.html` .

New materials (encyclopedias, interactive learning tutorials, ...) are created using simple XML formats which are tailored to easiness of authoring. In our experience it is more efficient to use home-made flexible and simple formats for authoring and only if needed transform them with XSLT to a format commonly used.

IUPAC GoldBook [`http://gold.zvon.org`]

IUPAC Compendium of Chemical Terminology,popularly referred to as the “Gold Book”, is one of the series of IUPAC “Color Books” on chemical nomenclature, terminology, symbols and units.

The XML version was created as part of the IUPAC project: Standard XML data dictionaries for chemistry. It is based on the online PDF version of IUPAC Gold Book that is available on the Royal Society of Chemistry’s chemsoc website.

XML format enables generation of several different presentational outputs tailored to particular needs, it improved indexation including navigational maps of dependence between terms. Both chemistry and mathematics is now captured in computer readable form and so extensibility to further additions and/or incorporation to other materials is assured.

Law-Ref.org

It is not enough to have information freely available, it is also necessary to have some possibility to find it and combine information from several sources. This is the reason while one of main selling points of scientific and technical publications is quality of indexing.

Based on XML technologies and Zvon experience we have recently opened a new site: Law-Ref.org where many important documents of international law were transformed to XML and thoroughly indexed.

As an example can be used "Treaty Establishing a Constitution for Europe", which is a very large document of a great importance. Thorough indexing makes the text much more accessible. While we cannot make sure that people read relevant documents before taking decisions in matters of great importance we at least hope that we can help to make them more informed.

Incorruptible XML editor

Very often we are cooperating with people who can be tought to write basic XML documents but they cannot be expected to detect well-formedness errors. It would be of a great help to have a widget which can be incorporated to any program where XML input is expected but which would not be able to introduce well-formedness errors. We have written such a widget as an extension of Python Tkinter Text class. The widget is now at alpha stage and I am using it regularly in preparation of my lectures (including this one) and other projects.

Informatics and chemistry [`www.vscht.cz/informatika-chemie`]

If every important information would be available in some structured way it would literary change the world. Unfortunately, we cannot expect from experts in science or any other fields that they will learn how to properly structure information and present it in a computer readable form. It may not be expected in foreseeable future that computers will understand scientific jargon and so we decided to manufacture and program "human hardware and software" for this enormous task. We are now teaching many BSc. students (and in a few year MSc. as well) different aspects of both science and publishing (including several XML related courses). These people will be at home in both worlds – natural sciences and XML – and we hope that in few years they will enable us to achieve tasks we are not strong enough to tackle at this moment.

XSH - XML Editing Shell (An Introduction)

Petr Pajas (Charles University, Prague)

Abstract

This paper gives a brief introduction to the XML Editing Shell (XSH), its major features, typical use cases, and some aspects of its design. XSH was designed and implemented by the author of this paper with the aim to create a scripting language and a productive shell-like environment that would allow easy XPath-driven analyzing and manipulating of XML documents. XSH is implemented in Perl and highly benefits from its programming power and rich spectrum of available modules.

Historical background

During the last few years XML (the Extensible Markup Language) became the primary data format for most applications equally on server and desktop, effectively forcing out not only various proprietary formats but also plain text. This of course applies even to UNIX systems, so well known for their excellent text processing capabilities and invaluable tool-chains. In time, XML has found its use in almost all areas of IT industry, not just those related to the Internet. Along with adoption of XML, new technologies, tools, APIs, and languages for manipulating XML encoded data have emerged. The core of these new tools is built around a wide family of XML-related recommendations by W3C (World Wide Web consortium). We may say without exaggerating that XML technologies are the key ingredients of today's IT industry.

XSH originated in 2001. At that time, there were only few command-line tools that could be easily used to perform simple query or find/replace tasks over XML documents. Of course, in some situations one could fall back to using UNIX text-oriented tools like `grep` and `sed`, disregarding all the benefits of a fully structured document that XML provides. For complex XML documents such approach is not only inconvenient, but also technically impossible. One of the solitary representatives of XML-aware command-line tools available at that time was the LTXML toolkit by the Edinburgh Language Technology Group. Nevertheless, its list of features was limited and in some scenarios the tools gave unreliable results. In most cases there was no other option than writing a single-purposed program in order to accomplish even a trivial task.

Author's idea of a tool that would fill this gap has crystallized over the year 2001, resulting in a first public release of XSH in January 2002. Since then XSH has been developed in irregular intervals of alternating development activity and in the same manner is being developed till now. In the year 2004 the XSH language was redesigned to allow even tighter integration with Perl, adding XSLT-like variables to XPath and cutting off some rough edges that emerged during the previous periods of incremental and somewhat ad-hoc additions to the language. The new version of the language and interpreter (backward incompatible with earlier releases) was first released in December 2004 as XSH2. In this paper we shell talk about XSH2 in its latest development version that approaches its release as XSH2-2.0.3.

A typical use case

XML is mostly used to store hierarchical data that can be later picked up by an application, processed, sometimes merged with other sources and finally transformed to some output format (e.g. XHTML). A developer of such an XML-oriented application is regularly confronted with at least some of the following tasks:

- getting familiar with a new XML vocabulary.
- examining various XML instances (sometimes even deriving a "schema" from an instance).
- defining a new XML vocabulary and transforming existing data (hopefully from XML) to the new format.
- upgrading XML instances when the XML format changes (which happens rather often during the early development phases).
- assembling, testing, and debugging XPath queries.
- writing test suits (that includes generating malicious XML instances).

Some of these tasks would typically require writing single-purposed XSLT stylesheets or programs in Java, C#, Python, or Perl, other could be accomplished with the help of specialized GUI applications (such as XMLSpy) that however typically don't allow for much automation. Both the XSLT and the Java (C#, ...) approach involve writing a considerable amount of extraneous lines of code just to turn a few relevant expressions or statements into something that actually can be applied on the data.

Similarly, an author preparing a website, book, manual or other kind of a document in an SGML or XML format (such as HTML, XHTML or DocBook) is often in a need to refactor portions of the document (e.g. if he or she decides to change sectioning of the document, restructure a complex table, reorder lists of items according to complex criteria, and so on), sometimes even to semi-automatically add technical markup (such as unique identifiers, media-object specifiers, citation and index term tags). All these tasks are often too specific to be given direct support by the authoring environment used (i.e. a XML-aware text processor), however sophisticated it may be.

In both of the outlined scenarios XSH can effectively reduce the amount of effort and time needed to achieve the desirable result, while allowing the user to take the full advantage of dealing with structured data. XSH language and the built-in shell allow the user to invoke XPath queries and numerous manipulation commands directly on the XML data without all the unnecessary overhead, while still offering full programming power.

Overview of XSH features

XSH language naturally glues its own shell-like syntax with the XPath 1.0 language, Perl, and the UNIX system shell. It offers over 100 commands covering the most common XML-oriented tasks, such as

1. XML and HTML parsing, XML validation (DTD, Relax NG, XML Schemas).
2. XML and HTML serialization, XML canonicalization (C14N), character encoding conversions.
3. manipulating the parsed DOM tree (deleting, adding, and renaming nodes, moving and copying nodes within the same document or between documents, surrounding portions of a document in an element, etc.).
4. inspecting the DOM tree by navigating it (as if it was a UNIX filesystem), making XML listings of portions of the document tree.

5. high-level syntactic constructions (sub-routines, blocks, conditional statements, foreach and while loops, try/throw/catch exception handling).
6. interfaces to other languages: in-line Perl code, system shell access, custom XPath extension functions, XSLT and XUpdate processing.

The interactive shell, among other, features built-in documentation for every command, keyword, and XPath extension function, and supports a fully context-sensitive XPath completion. Typing a partial location path and pressing TAB automatically completes element and attribute names as well as XPath functions and axes according to the current document and context.

In a typical XSH session, one parses one or more XML and/or HTML documents or streams to DOM (using `open` command), inspects the DOM tree(s) (e.g. with commands `cd` and `ls`), modifies them (commands like `cp`, `mv`, `set`, `rm`, `insert`, `wrap`, and `xslt`) and saves the result (command `save`). Most of these commands operate on nodes of DOM trees. A natural way to specify a node of a DOM tree is using XPath node-set expressions. Wherever XPath isn't sufficient, a Perl expression (enclosed in braces) can be used instead. XSH transparently translates between XPath objects (node sets, strings, integers, boolean values) and the corresponding Perl and DOM objects, allowing XPath and Perl use the same variable space. It is thus possible to e.g. organize and look-up DOM nodes in Perl arrays and hashes. While node-sets resulting from XPath expressions are always processed in the document order (which is defined in the XPath recommendation), Perl lists can be used to enforce processing in any other custom order (the `sort` command provides an easy way to reorder lists of nodes).

Many XSH commands return values. For DOM manipulation commands the return value is typically a node-set consisting of the nodes created by the command. This is particularly useful when moving nodes, because XSH for various reasons never moves existing nodes directly but instead creates identical copies at the target locations and then removes the originals.

The taste of XSH

So far we only enumerated some XSH features and aspects without giving an opportunity to feel the taste of the language. To show at least one not completely trivial real-world example, let's say we have a XML document with many `date` elements containing flat dates in the usual format `YYYY-MM-DD`, which we now want to to turn into XML structures with year, month, and day parts contained in separate sub-elements. It could be truly argued, that this can be easily achieved with text-oriented tools, so let's assume that we only want to convert some dates, say those of releases of software products, leaving all other `date` elements untouched. Because the same thing can be achieved in many ways, we have a good opportunity to reuse this example several times in order to demonstrate various syntactical constructions of XSH.

```
foreach //release[@product-type="software"]/date
  xinsert chunk
    { $_=literal('.'); # get the content
      s{^(\d{4})-(\d\d?)-(\d\d?)$}
        {<year>$1</year><month>$2<month><day>$3</day>};
      $_
    } replace text();
```

The `foreach` statement iterates over all nodes of a current document selected by the `//release[@product-type="software"]/date` XPath expression and setting each of these nodes as the context node executes the `xinsert` command. This command creates one or more nodes of a given type according to a given expression and inserts the resulting nodes to specified destinations. In our particular case the nodes are specified as a "chunk", meaning a XML representation of a document fragment; other possible types would be element, attribute, text, processing instruction, CDATA section, and comment. The desired XML chunk is constructed by transforming the flat input string with a Perl's regexp substitution. In XSH, destinations of copy-like operations are specified as node-lists preceded by a location parameter that determines where the source nodes go in relation to the destination nodes (one of `into`, `after`, `before`, `replace`, `prepend`, `append`). The initial `x` in `xinsert` stands for "cross", meaning that (copies of) the source nodes go to all destinations (rather than to the respective destinations as with `insert`, where "respective" in this sense means that n'th node in the source node-list would go to the location relating to the n'th node in the destination node-list).

```
foreach my $date in //release[@product-type="software"]/date {
  my $y $m $d; # lexically-scoped variables
  if { ($y,$m,$d) = ($date=~/(\d{4})-(\d\d?)-(\d\d?)/) } {
    remove $date/node();
    xinsert chunk <<"EOF" into $date;
<year>${y}</year>
<month>${m}</month>
<day>${d}</day>
EOF
  }
}
```

In the above code snippet we use a similar approach, but this time we first separate the individual date fields, store them in variables and use a here-document to form a XML fragment populated with the previously obtained values.

By employing XPath extension functions of XSH such as `xsh:split` or `xsh:match`, and exploiting the already mentioned "respective" source-to-destination mapping feature of various node-manipulating XSH commands, we may achieve the same result with a much more concise:

```
for //release[@product-type="software"]/date
  copy xsh:split("-",.)/text()
    into &{ xinsert chunk "<year/><month/><day/>" replace text() }
```

The `&{ ... }` construct on the last line evaluates the enclosed XSH command(s) and returns the result. In our case, it creates three empty elements that replace the original text context of the `date` element and returns a node-list consisting of the three newly created element nodes. Prior to that, XSH evaluates the XPath expression `xsh:split("-",.)/text()` obtaining a node-list consisting of three text nodes each of which represents one part of the string value of the current `date` element (which is specified as the second parameter `.` to `xsh:split`) after being chopped up on the delimiting dashes. Obtaining three source nodes and three destination nodes, the copy command places each of the source nodes into the respective destination element.

To complete, we present yet another solution, demonstrating how Perl data structures can easily be translated to the corresponding XPath/DOM objects. The helper Perl function `nodelist` converts a list of its arguments to a node-list by translating each of them to a DOM object. In our case, the arguments are the three components of the flat date string as matched by the same regular expression as above. The `nodelist` function translates them to a node-list consisting of three text nodes, which are subsequently used to create the desired sub-elements, in this example using the `set` command of which we will talk later.

```
foreach //release[@product-type="software"]/date {
  my $date = string(.);
  my $parts = { nodelist( $date=~/^(\d{4})-(\d\d?)-(\d\d?)$/ ) };
  remove text();
  set year $parts[1];
  set month $parts[2];
  set day $parts[3];
}
```

It has been noted that XSH extends XPath with many useful functions (similar in functionality and syntax to numerous EXSLT functions as well as most Perl core string-related functions such as `grep`, `split`, `join`, `sprintf`, etc.). Moreover, user-defined functions can be added to XPath as Perl sub-routines. For instance, if for a particular application it would be useful to have the possibility to select nodes based on the SHA digest of their text content, then one could instantly turn an existing Perl SHA implementation to a new XPath function, say `sha()`, with the following commands:

```
# import the Perl function sha256_hex() from Digest::SHA module
perl { use Digest::SHA qw(sha256_hex) };

# register this Perl function for XPath as sha()
register-function sha sha256_hex;
```

The newly defined function `sha()` can now be used as any other XPath function:

```
# compute SHA hash for a particular paragraph:
$sha = sha(//para[23]);

echo $sha;

# prints e.g.
# f75af3a78b9c7f6f812c6fbecacfb5f8c0a8846318d5ed677ce62fac02a9e252

# get cannonical XPath for a node with a given SHA hash
locate //node()[sha(.)=$sha];

# prints /book/chapter[2]/section[1]/para[2]
```

Recently a new powerful command, named `set`, has been added to XSH providing a concise syntax for building DOM subtrees from scratch as well as for adding data to existing ones. This command takes one or two arguments the first of which is an XPath expression (a location path) and the second is arbitrary expression resulting either in a node-set or in a string value. The `set` command processes the location path one location-step at a time and if necessary, auto-creates nodes matching the missing steps. Finally it copies the second argument as the content of the node corresponding to the last location step. Hence, for example, to build the entire DOM subtree representing the following XML fragment

```
<customer id="jm">
  <name>
    <firstname>John</firstname>
    <surname>Moore</surname>
  </name>
  <detail/>
  <detail/>
  <detail type="email">john@moore.com</detail>
</customer>
```

one can use the following five commands:

```
set customer/@id "jm";
set customer/name/firstname "John";
set customer/name/surname "Moore";
set customer/detail[3]/@type "email";
set customer/detail[3] "john@moore.com";
```

Although only location-steps that follow the child-node axis are allowed by the current implementation, it is planned to extend the `set` command to support all non-transitive axes.

Implementation

XSH interpreter is written in Perl. The implementation builds on top of several other open source projects which the author feels obliged to briefly mention.

Daniel Veillard's libxml2 and libxslt C libraries give XSH the fundamental XML components, namely the XML parser, validators, in memory tree representation, and complete implementations of XPath 1.0 and XSLT. To that the Perl bindings add a significant portion of the standard DOM Layer 3 API.

Implementation of the parser of XSH syntax is based on an excellent Perl module Parse::RecDescent authored by Damian Conway, one of the key designers of the upcoming Perl 6. RecDescent grammars defined by this module incorporate both the lexer and parser functionality which significantly accelerates the development process. RecDescent grammars are compiled to Perl modules providing the actual parsers.

It is not without interest that while RecDescent grammars are usually written in a plain-text syntax similar to that of tools like `yacc`, XSH has its grammar described using an XML file from which not only the plain-text RecDescent grammar, but also documentation, on-line help, and completion lists of various kinds are automatically generated. Indeed, this approach proved very useful: during the XSH development, XSH was several times successfully used to refactor its own grammar definition as well as the embedded Perl code snippets (often to employ some low-level optimizations).

Places for improvement in future revisions

There is of course a lot of space for improvement both on the language side and that of the implementation. XSH's loose shell-like syntax (with individual command arguments separated by white-space, rather than, say, commas) has some undeniable drawbacks. The most notable one is that one can't use white-space in an XPath expression unless the whole expression or its relevant sub-expression is enclosed in brackets. Another syntax collision of this kind is in XSH's pipe-line redirection syntax borrowed from shell languages. In XSH, any command can be ended by the `|` character and followed by one or more system commands to which its output is redirected. Unfortunately, `|` is also used in XPath as an operator for union of node-sets. Hence, in XSH, this XPath operator may only occur within a bracketed part of an XPath expression.

XSH is an interpreter written itself in an interpreted language. Hence, XSH is not as fast as other scripting languages, like Perl, Python, or Ruby. Also, the RecDescent grammar for XSH compiles to a relatively large parser module (about 1MB of Perl code) which by itself on today's desktop hardware takes Perl about 1 second to parse. That means it takes at least 1 second for XSH to start up. These implementation drawbacks could be partially, if not completely, eliminated by rewriting portions of XSH in C and possibly replacing current XSH's RecDescent grammar with suitable `lex` and `yacc` equivalents. On the other hand, such changes would discard some of the benefits of the development process gained by the current approach.

On the brighter side, the XML parser used by XSH is one of the fastest available and the memory footprint of a parsed XML document is also acceptable, especially in comparison with other DOM implementations (about 320MB RAM for a 50MB lexicon-like medium-structured XML document). There is still some room for optimization in the XPath evaluation code, which is one of the areas where the author would like to continue contributing with incremental improvements.

Conclusion

XSH implements a rich set of features, borrowing many from the underlying libxml2, libxslt, their Perl bindings and several other Perl modules. Over the years of part-time development the extensive (and sometimes rather aggressive) use of these modules in XSH allowed the author to discover many hidden bugs in these components and a couple of times even in Perl itself. The open source nature of these projects allowed the author to partially contribute to them, if not more, then at least by providing detailed bug reports, test cases, and often patches addressing these issues.

While current implementation of XSH cannot compete in processing speed with well-optimized XSLT and XQuery processors, it is still sufficient for manipulating most medium-sized documents (up to, say, tens of megabytes). Moreover, its shell and batch processing capabilities, the ability to simultaneously access several XML documents, and most of all its expressive power offer a productive environment that can make life of an XML developer much easier by allowing him or her to solve problems directly, without having to write hundreds of lines of extraneous syntactic bloat. It is suitable for various pre-processing and normalization procedures on XML data, use in makefiles, etc. It is also an ideal testbed for prototyping application components, XPath expressions, XML schemas and protocols and might serve even as a teaching tool for XML-related technologies.

Bibliography

Other sources of information about XSH

XSH project home page
`http://xsh.sourceforge.net`

Kip Hampton, *XSH, An XML Editing Shell*, XML.org, July, 10, 2002
`http://www.xml.com/pub/a/2002/07/10/kip.html?page=1`

Randal L. Schwartz, *Using XSH to Scrape Web*, Linux Magazine, January 15, 2004
`http://www.linux-mag.com/content/view/1559/2082/`

Wellie Chao, *XSH: Interactively Manipulate and Analyze XML Data*, DevX.com, March 17, 2005
`http://www.devx.com/xml/Article/27567`

References

XML `http://www.w3.org/XML/`

DOM `http://www.w3.org/DOM/`

XSLT `http://www.w3.org/TR/xslt`

XPath `http://www.w3.org/TR/path`

DocBook `http://www.docbook.org`

libxml `http://xmlsoft.org`

Parse::RecDescent
`http://search.cpan.org/dist/Parse-RecDescent/lib/Parse/RecDescent.pod`