

OPENOFFICE, THE NEW WXS IDE

Eric van der Vlist (vdv@dyomedea.com)

OpenOffice, the new WXS IDE

XTech 2005

May 2005

Copies : <http://dyomedea.com/papers/xtech2005>

XML SCHEMAS ARE... NOT SCHEMATIC!

XML schemas are about validation

The main (and only common) feature of the many XML Schema languages is validation. XML Schema languages are about validation and when they provide features that lets you describe XML structures, these features describe validation rules more than anything else.

They are poor modelling tools

Because of this target, XML schema languages are low level, difficult to learn and to read for a non specialist and they are poor modelling tools compared to first class modelling tools such as UML.

Higher level modelling tools are needed

Higher level modelling tools are needed, especially if the design is done by people who are not XML schema specialists.

Relying on the user friendliness of an XML schema IDE isn't an option: the most user friendly tool will leave to the user technical decisions that should taken by specialists.

HIGHER LEVEL MODELLING TOOLS

UML

UML is the first option that comes to mind when we talk of modelling tools. Class diagrams are quite handy to describe XML documents. Their big benefit is that they are universally understood.

However, they require stereotypes and/or convention to specify XML specificities such as the distinction between elements and attributes, defining when order matters, ...

Also, the graphical nature of class diagrams makes them difficult to read when you have a high number of elements to represent.

Examples

Examples of documents are often easier to read than XML schemas. For that reason, XML snippets are often added to the documentations to illustrate the schemas.

Proposals have been done to use annotated XML documents as higher level views of XML schemas (see for instance <http://examplotron.org>), but this approach is very marginally used.

Spreadsheets

Spreadsheets have become one of the most popular tools in the toolbox of many computer users and that's no surprise that many people have had the idea to store model information in their favourite tools.

Spreadsheets are less graphical than UML diagrams and a simple model with few elements will be less readable in a spreadsheet than in an UML diagram. On the other hand, when the number of elements is increased, the ability of spreadsheets to handle large tables becomes predominant and the spreadsheet easier to read than the UML diagram.

Another argument in favour of the spreadsheets is that UML tools are expensive and have limited interoperability while spreadsheets are either open source or likely to be already installed on most of the workstation. Furthermore, the different spreadsheet processors are quite interoperable.

WHICH SPREADSHEET?

My requirements

My requirements for choosing a spreadsheet processor have been:

Works on multiple platforms including Linux

Because I am a Linux user, I needed a tool that's working on Linux. Because there is a risk that other people may not all be Linux users, I wanted a tool that runs also on Mac OS-X and Windows...

+ XML friendly

Because I wanted to use XSLT to transform my spreadsheets into W3C XML Schemas, I needed a tool that, at the very minimum, was able to produce XML documents.

= OpenOffice

The first reason alone was enough to impose OpenOffice. The XML features of OpenOffice have been the cherries on the cake... We'll see that later on.

1 vocabulary, 3 approaches

A prospect called me to ask what was the best approach to model a XML vocabulary. After discussion, it appeared that this vocabulary was an exchange format involving three different teams working from three different locations. Each team has its own approach to describe the format and they needed external expertise to choose the best approach.

UML

One of the team was using an UML modelling tool and generating W3C XML Schema schemas from the UML class diagrams. Since UML was not a strategic choice for the customer, this team was the only one to have licenses for this UML tool and this option wasn't considered generalizable.

W3C XML Schema

Another team was using an XML IDE to edit WXS schemas. The different teams had not followed WXS trainings and this option was considered too complex.

Pseudo XML snippets

The last team was producing plain text documentations including pseudo XML snippets (XML pieces with non well formed annotations to specify types and cardinalities. This option was considered anecdotal.

My proposals

Given the context, I proposed three (other) different approaches and documented their advantages and drawbacks.

Examplotronish

The first option was an adaptation of their XML snippets, keeping as much as possible of the conventions they had invented but making them well formed XML. I provided sample XSLT transformations showing how they could be transformed into WXS schemas.

OpenOffice spreadsheets

The second option was to use OpenOffice spreadsheets as a modelling tool.

RELAX NG

The third option was to use the RELAX NG compact syntax which is much easier to learn and to read than W3C XML Schema and to convert that syntax into W3C XML Schema schemas using trang.

Their choice

OpenOffice spreadsheets

After looking at these options, they decided to use OpenOffice spreadsheets which was the option that they were the most confident with.

HOW DOES THAT WORK (DEMO)

1 model = 1 spreadsheet with 5 columns

A model (schema) is defined in a spreadsheet with five columns.

Element or attribute name

The first column is the element or attribute name.

Content

The second column is used for the definition of the content of the elements when they have a complex type.

Cardinality

The third column is the cardinality using a min-max convention.

Type

The fourth column is the datatype.

Comment

The fifth column is used for documentation.

Simple type element or attribute = 1 line (DEMO)

A global simple type element or attribute requires just one line :

Element or attribute	Content	Card	Type	Documentation
@id name			xs:ID xs:token	id attribute name (definition)

Complex type complex content element = 1 line + 1 line per sub-element and attribute

Element or attribute	Content	Card	Type	Documentation
author	name surname born died	0-1	xs:token xs:date	name (reference) Surname (local) birth date (reference) death date (optional) (local)

Complex type simple content element = 1 line + 1 line per attribute

Element or attribute	Content	Card	Type	Documentation
title	@lang		xs:token xs:language	book's title language (no support for namespaces yet)

Notes

For simplification purposes, we have adopted the following conventions and restrictions in the current version (most if not all of them could be relatively easily fixed):

No support for namespaces

The current version does not support namespaces.

No support for global type definition

The current version has no support for defining global types (either simple or complex).

Complex type elements are always global

From the conventions we've used in the layout of the table, complex type elements are always defined globally.

Attributes and simple type elements can be either local or global

Global attributes and simple type elements use the first column for their names. Local ones are defined as content of an complex type element and their names appear in the second column.

FULL SCHEMA (DEMO)

Element or attribute # Description of a library (example)	Content	Card	Type	Documentation
library	book			Library element (root) book element (reference)
book	@id isbn title author character	0-n 0-n	char(10)	book element (description) id (reference) ISBN number title author(s) character(s)
title			xs:token	book's title
	@lang		xs:language	language (no support for namespaces yet)
author	name surname born died	0-1	xs:date	name (reference) Surname (local) birth date (reference) death date (optional) (local)
character	name born qualification	0-1	xs:token	name (reference) birth date qualification
# Common attributes				

@id	xs:ID	id attribute
-----	-------	--------------

Common elements

name	xs:token	name (definition)
born	xs:date	birth date (definition)

WXS EXPORT FILTER (DEMO)

XSLT transformation

The simplest OpenOffice export filters are basically XSLT transformations designed to transform OpenOffice's native XML format into any other XML format.

In our case, I have written an XSLT transformation "schema.xsl" that transforms OpenOffice into W3C XML Schema.

Library.xsd

The schema generated from this example by schema.xsl" is (note the difference of verbosity):

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation> Description of a library (example)</xs:documentation>
  </xs:annotation>
  <xs:element name="library">
    <xs:annotation>
      <xs:documentation>Library element (root)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="book">
          <xs:annotation>
            <xs:documentation>book element (reference)
          </xs:documentation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="book">
    <xs:annotation>
      <xs:documentation>book element (description)</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

```

</xs:annotation>
<xs:complexType>
  <xs:sequence>
    <xs:element name="isbn">
      <xs:annotation>
        <xs:documentation>ISBN number</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:maxLength value="10"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element ref="title">
      <xs:annotation>
        <xs:documentation>title</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element ref="author" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>author (s)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element ref="character" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>character (s)</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute use="required" ref="id">
    <xs:annotation>
      <xs:documentation>id (reference)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
</xs:element>
<xs:simpleType name="title">
  <xs:annotation>
    <xs:documentation>book's title</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token"/>
</xs:simpleType>
<xs:element name="title">
  <xs:annotation>
    <xs:documentation>book's title</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="title">

```

```

        <xs:attribute use="required" name="lang" type="xs:language">
            <xs:annotation>
                <xs:documentation>language (no support for namespaces
                yet)</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="name">
                <xs:annotation>
                    <xs:documentation>name (reference)</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="surname" type="xs:token">
                <xs:annotation>
                    <xs:documentation>surname</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element ref="born">
                <xs:annotation>
                    <xs:documentation>birth date</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="died" minOccurs="0" maxOccurs="1" type="xs:date">
                <xs:annotation>
                    <xs:documentation>death date (optional)</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="character">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="name">
                <xs:annotation>
                    <xs:documentation>name (reference)</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element ref="born" minOccurs="0" maxOccurs="1">
                <xs:annotation>
                    <xs:documentation>birth date</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

```

        <xs:element name="qualification" type="xs:token">
            <xs:annotation>
                <xs:documentation>qualification</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:annotation>
    <xs:documentation> Common attributes</xs:documentation>
</xs:annotation>
<xs:attribute name="id" type="xs:ID">
    <xs:annotation>
        <xs:documentation>id attribute</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:annotation>
    <xs:documentation> Common elements</xs:documentation>
</xs:annotation>
<xs:element name="name" type="xs:token">
    <xs:annotation>
        <xs:documentation>name (definition)</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="born" type="xs:date">
    <xs:annotation>
        <xs:documentation>birth date (definition)</xs:documentation>
    </xs:annotation>
</xs:element>
</xs:schema>

```

Xmlschema.jar

In order to facilitate their installation, export filters can be packed into jar files. Their installation becomes then a matter of a couple of mouse clicks.

Export as W3C XML Schema

As soon as the export filter has been installed, a model can be exported as W3C XML Schema straight away from the OpenOffice application.

OPENOFFICE, A NEW XML SCHEMA IDE?

This is cool, but we could do much more, including (to name few):

Importing WXS schemas

That would be tough in the general case, but pretty trivial for schemas that are following the style used in our exports.

Importing/exporting RNG schemas

That should be rather easier than importing and exporting WXS schemas given the fact that RELAX NG is easier to read and write than WXS.

Generating textual documentation

We could generate the textual documentation from a schema as an OpenOffice Writer document. This document could then be exported in the many formats supported by OpenOffice.

Generating graphical documentation

This one would probably be the killer app!