

LIGHTWEIGHT DERIVATION AND TRANSLATION OF XML VOCABULARIES

<http://dyomedea.com/papers/2004-extreme/>

Extreme Markup Languages 2004

By:

Eric van der Vlist, Dyomedea (vdv@dyomedea.com)

G rard Vidal, ENS Lyon (Gerard.Vidal@ens-lyon.fr)

S bastien Chaumat, ENS Lyon (Sebastien.Chaumat@ens-lyon.fr)

S bastien Pilloz, ENS Lyon (Sebastien.Pilloz@ens-lyon.fr)

CONTEXT: ENS (ECOLE NORMALE SUP RIEURE) DE LYON

The strategy

Open Standards for the Learning Content Management System (LCMS)

It is the strategy of the PRATIC department of Ecole Normale Sup rieure de Lyon (ENS Lyon) to apply openness, normalization and standardization to the development of its Learning Content Management System (LCMS).

Models for data and metadata, files formats, architectures and software infrastructures for management and storage, human-machines interfaces and even software development style are affected. Thus the durability and the evolutivity of the whole set of the learning resources and connected systems are assured.

One of the major steps to an effective LCMS is the standardization of data. This means not only standard file format but more deeply a unification of the structure of the contents. Inquiries showed that all of our production (courses, tutorial classes, experimental work ...) could more or less be structured within a kind of "scientific article".

This structure contains not only classical hierarchy of sections and subsections but also important substructures like commented illustrations, metadata, highlighted text. Having a single structure suitable for all kinds of content helps implementing automatic processing on the data. It also reduces the amount of knowledge to transfer to authors.

After designing this theoretical structure, we looked for the most efficient implementation. XML came as a natural partial answer. Among the arguments in favor of XML, its use is made mandatory by the "Interoperability Appendix" of the French official guidelines for Numerical Working Spaces (French ministry of Education) [1].

The tactic

A customized vocabulary

Choosing the schema was more difficult. One half of our authors are comfortable with document processing (like LaTeX). But the other half only knows WYSIWYG words processors without any notion of data structuration.

As processing unstructured data is simply impossible we decided to initiate a move to document processing using something as simple as possible. DocBook sounded interesting to our technical team because it offers a lot of existing format processing tools. But it requires a long training. DocBookSimple was not really better. And LaTeX is not XML...

Furthermore, most of the teachers and assistants who will edit these lectures are not IT professionals and some of them do not know English.

We then decided to design our own XML schema, as close to Latex and DocBook as possible, but with limited vocabulary and the advantage of localized tags.

It was quite evident that we were reinventing the wheel and that existing vocabularies such as DocBook, TEI or maybe even XHTML could cover our functional needs and give access to a wide range of free resources, but using such vocabularies wasn't a solution for us either.

Small subset

Even their "lite" versions are very complex and cover a scope that is much wider than what they wanted to cover. This complexity isn't hidden by the XMLMind XML Editor (XXE) that they use and when the user wants to add an element or attribute, he needs to pick one amongst an gigantic and confusing list. Ideally, a customized subset of one of these vocabularies should be used.

Translation of element and attribute names in French

Even if a subset of one of these vocabularies was being used, the names of the elements and attributes would still not be very readable: for a user who speaks French and not English, it's not that easy to guess that "superscript" means "exposant"!

The translation part of this problem is a low frequency permathread on XMLfr's mailing lists: people often ask the question of how elements and attribute names could be translated in French and during the discussions that follow, lightweight solutions based on simple translations tables read by XSLT transformations or SAX filters have often been proposed. This sounded like an opportunity to put these recommendations into practice!

As a single and simple step

What was different here, was the requirement of defining a subset of the vocabulary, and thus came the idea to perform a schema based simplification and translation of DocBook in a single step.

THE OPENOFFICE "SCHEMA"

An OpenOffice "schema" (yes, that's a schema even though its format may seem unusual) has been designed as an OpenOffice spreadsheet with three different sheets: the first to define elements, the second to define attributes and the third to provide common definitions.

Element definitions

The definition of elements is made through six main columns:

Name

The first column holds the element name and defines the name of the element in our new vocabulary (lets call it "livredoc").

DocBook name

The second column holds the name of the corresponding DocBook element: to facilitate the translation of the documents to and from DocBook but also the translation of associated resources such as CSS stylesheets, we have followed the principle of a strict one to one matching between our "livredoc" and DocBook.

Prototype name

The third column holds the name of the corresponding element in the prototype developed by ENS-Lyon. This matching has been added for documentary reasons (it made it easier for users knowing this existing vocabulary to understand the meaning of our elements) and it is not a one to one matching: some livredoc elements have no equivalent in the prototype. This column has also helped to check that no features had been forgotten in livredoc.

Element content

The fourth column is holding the element content.

Attributes

The fifth column is holding the attributes

Miscellaneous

The sixth column is holding styling information

Other columns have been added for metadata (description, comments, questions, ...). They aren't used to generate anything at the moment but could be used to generate a detailed

Compact RELAX NG

How do you describe complex content models in a cell?

The big question while creating this spreadsheet has been: "how do you describe complex content models in a cell of a spreadsheet?"

The simple example given in the book "RELAX NG" shows how to design a simple data oriented application where it was enough to give coma separated lists of sub-elements but that was far from being good enough to describe even a simple subset of DocBook. In this case, we needed a compact, simple yet expressive syntax that would fit in a spreadsheet cell and define a content model.

That's a job for RELAX NG's compact syntax...

Compact, simple yet expressive? That looked like a job for the compact of RELAX NG and we decided to include snippets of compact RELAX NG in the fourth column: for instance, the definition of the content of "article" is:

```
titre, métadonnéesDArticle?, (contenuBloc|partie)+
```

And since we rapidly found that this was working very well, we have used the same principle to define the attributes and the definition of the attributes of "colonne" is:

```
attributsCommuns, alignement?, alignementVertical?
```

Design decision: a named pattern per element and attribute

Those of you familiar with RELAX NG's compact syntax will have noticed that in my definitions I am using references to named patterns ("titre", "métadonnéesDArticle", "contenuBloc", "attributsCommuns", "alignement", ...). Your next question is probably: "how are these named pattern defined?"

we have taken the very simplistic principle that each element and attribute definition would generate a named pattern with the same name. This principle have worked very nicely up to now and if it became and issue (for instance if we had to define an element and an attribute with the same name), it would always be possible and easy to add a column to define the name of the named pattern to generate when it should be different from the name of the element or attribute.

Attribute definitions

Same principle

Attribute definitions follow the same principle than element definitions, except that there is a single column to describe the content where for element definitions, we have a column for the content and a column for the attributes.

The content of attributes can be done directly as datatypes (for instance, the content of

"hauteur" is "xsd:positiveInteger") or using references to global definitions (for instance, the content of "alignement" is defined as " valeurDroite | valeurGauche | valeurCentre").

Global definitions

Same principle

Again, global definitions follow the same principle than attribute definitions with the difference that the "docbook" column is not always filled.

In fact, most of the time, the definitions are just definitions used as content elsewhere and the "docbook" column is meaningless. For instance, "contenuCommun" is just a named pattern for "table | liste | listeOrdonnée | figure" and this has no equivalent in DocBook.

But global definitions can also be literals, such as "valeurDroite" which is "'àDroite'" and in that case, it is useful to know that this is the translation of the value "right" and that's expressed using the DocBook column.

W3C XML SCHEMA GENERATION

The generation of the W3C XML Schema out of the "OpenOffice schema" is done in two steps: a XSLT transformation generates a RELAX NG compact schema and James Clark's converter `trang` generates a W3C XML Schema out of the RELAX NG schema.

OpenOffice to RELAX NG (XSLT)

Straightforward

Since the definitions are written as RELAX NG compact syntax snippets in the OpenOffice spreadsheet, the schema that is generated is also using that compact syntax and the role of the XSLT transformation is just to combine those snippets according to the semantics of the "OpenOffice schema".

That's a pretty trivial task and the XSLT transformation is only 120 lines (including empty lines to separate the templates) long!

Beware of "@table:number-columns-repeated"

One of the little tricks to know if you need to manipulate OpenOffice spreadsheets, is that adjacent cells that have the same content are serialized as a single "table:table-cell" element with a "table:number-columns-repeated" attribute and that deserves a specific template to handle that.

The other difficulty, but that's a XSLT FAQ relevant to any transformation that uses the text output method, is to be very strict with the whitespaces handling if you want to get an output that's nice to read.

Otherwise, we have basically three templates (one for each spreadsheet) that generate element, attribute and named pattern definition coying the RELAX NG snippets found in the

columns describing the content.

RELAX NG to W3C XML Schema (trang)

Works out of the box

Using trang is very straightforward and went without any trouble.

Substitution groups can be switched off

The only "surprise" we've had is that by default, trang generates many choices between elements as abstract elements and substitution groups (I think that substitution groups are probably one of the only features that James Clark finds elegant in W3C XML Schema!) and that has proven confusing for schema processors which are not fully conform to the spec.

There are two ways to disable this feature in trang: either on the command line or through schema annotation. I have preferred schema annotation and have added a:

```
[ tx:enableAbstractElements = "false" ]
```

annotation at the beginning of the schema that does the trick.

After this step, we have a W3C XML Schema describing our vocabulary that can be used with schema guided editors that do not support RELAX NG.

What kind of design pattern is that?

Protocol adaptation

In term of design patterns, what we have done here is usually known as "protocol adaptation".

Including a facade

The first step in this protocol adaptation is a "facade" design pattern that presents a subset of the DocBook vocabulary.

In Object Oriented programming, facades are amongst the most difficult design patterns to automate: they do not fit programming languages sub-classing methods that basically add features to classes but can't easily remove features.

Implemented as a "derivation by redefinition"

In our case, we have seen that the schema is generated from the OpenOffice spreadsheet without any reference to DocBook's schema or DTD.

That means that, strictly speaking, we've defined a subset by redefinition rather than by derivation and that while writing the definitions of our elements, attributes and global definitions, we need to take care that we define sub sets of the corresponding content models in DocBook.

Could deserve a formal proof

The main issue with defining a subset by redefinition is that there is (far as I know) no tool

available that can give a formal proof that our redefinition is a subset of the original definition.

To give that proof, we would need a tool that would compute the difference between two schemas. I have discussed this issue in private emails with Murata Makoto and he believes that this is possible using "Hosoya-san's neat algorithm, which was first presented at Plan-X and then CIAA" (see <http://www.kurims.kyoto-u.ac.jp/~hahosoya/papers/onatt.ps>).

The implementation of this algorithm was, however, far above the scope of our project.

Other approaches: derivation by restriction, by annotation, ...

Another approach would have been to start from a DocBook schema (RELAX NG or W3C XML Schema) and annotate this schema to define what needs to be removed and the names after translation.

This annotation could have been done internally to the DocBook schema using Bob DuCharme's schema stages proposal (<http://www.snee.com/xml/schemaStages.html>) but with the downside that we'd have to edit the normative DocBook schemas and that these annotations would have to be carried on to every new release of DocBook.

Also, Bob DuCharme's proposal require that you change the content model of a schema to perform restrictions such as converting an optional element to become mandatory.

The downside of "derivation by annotation" is that to be able to do the translation one would need to understand the semantics of the DocBook schema which are very complex and I have considered that the approach I have taken was easier to implement.

For that very reason, I am developing a new approach to describe XPath based derivation transformations to apply on a schema externally to the schema itself. This is still work in progress, but stay tuned, I should publish more on this subject in the future.

CSS STYLESHEET TRANSLATION AND AUGMENTATION

The idea here, is to be able to translate CSS stylesheets written for DocBook into CSS stylesheets for our vocabulary. This is most useful because most of the customization of the XXE editor is done through CSS stylesheets and if we are able to translate them we can immediately "borrow" its native support of DocBook.

Context aware translation

We have hesitated on the tool I would be using to do this translation. A simple regular expressions based tool such as awk could probably have been enough given the fact that the CSS stylesheets provided with XXE to support DocBook are not exploiting all the syntactical sugar of CSS and are very regular.

On the other hand, we wanted to be able to translate other stylesheets and also to be able if

needed to do more advanced processing such as removing the declarations that are relevant to DocBook elements or attributes removed from our vocabulary.

PyParsing

For that reason, we have preferred to use a grammar based tool and since I like Python, I have looked for a Python library that could do this.

A grammar based parsing library for Python

The library we have selected is "PyParsing", a library that lets you write your grammar in a "pythonic" way. For instance, the definition of a CSS selector is written as: "selector = Group (simple_selector + ZeroOrMore(combinator + simple_selector))".

Due to lack of time, the CSS grammar written with PyParsing is very far from having been exhaustively tested, but it does its job with the DocBook CSS stylesheets provided with XXE.

Here is a snippet of this grammar:

```
ruleset = Group(delimitedList(selector)) + \  
           Literal('{') + \  
           Group (ZeroOrMore(declaration)) + \  
           Literal('}')
```

Hacked to perform punctual replacements

The main difficulty we've had with PyParsing is that I am using it in an untypical fashion: PyParsing is good at parsing a document and providing to the application structures of tokens with all the syntactical sugar removed.

The problem we had is that we wanted to parse the document according to the grammar only to know where to perform punctual translations and we wanted to keep all the syntactical sugar.

Fortunately, PyParsing also sends pointers within the source document to show where it has found the tokens and we have ended up relying on these pointers to copy everything from the source string into a result string with only specific tokens (mainly element and attribute names) translated.

Spreadsheet driven translation

Based on my "XML data driven classes" library

The translation is directed from the translation tables found in the spreadsheet and to read the spreadsheet, we are using a data driven library that dynamically creates and binds objects to XML elements and attributes. With that library, no DOM requests are needed and the table holding the definitions of elements can be referred as "vocabulaire.document_content.body.tables[u'Eléments']".

Additional information on that library may be found at <http://dyomedea.com/papers/2004-OSCON/>.

With all that in hand, the translation is mainly the process of replacing element and attribute names in the selectors when they are found in the translation table. The whole story (including the definition of the grammar is less than 300 lines long.

CSS stylesheet augmentation

Being able to translate existing stylesheets has proved not to be enough and our users have strongly pushed to get new features added to these stylesheets.

Because CSS just adds all the declarations found in all the selectors matching a node, there has been no need to modify the stylesheets during translation and we've just had to generate an additional stylesheet with style information added in the OpenOffice "schema".

Here again, a simple XSLT transformation similar to the one that's been made to generate the RELAX NG schema did the trick.

XSLT GENERATION

LivreDoc to DocBook

The other resource that can be generated from our OpenOffice "schema" is an XSLT transformation converting our vocabulary into DocBook.

Gives access to the DocBook resources

Our goal being to provide access to the huge libraries of XSLT stylesheets for DocBook, a transformation from LivreDoc to DocBook was good enough and that transformation is also pretty straightforward to write (95 lines).

With that transformation in hand, we are able to take any LivreDoc document and convert it into a DocBook document that can be further processed using any DocBook tool.

Useful to test our derivation by redefinition

This resulting transformation is also handy to check that we've not made errors in our restriction by redefinition: by applying it to various LivreDoc documents covering different use case and by validating the result of the transformations against the DocBook schemas, we are able to test the quality of our redefinition.

The execution of the resulting transformation has been integrated to XXE in a LivreDoc menu that gives to the end user the possibility to export his LivreDoc documents as DocBook.

DocBook to LivreDoc: more challenging

The reverse transformation would be much more challenging to write since our vocabulary is a subset of DocBook and a reverse transformation would have to take decisions about how to cope with structures allowed in DocBook and disallowed in our subset.

CONCLUSIONS

The project isn't operational yet as we are writing these lines (June 25th) and that's too early to draw definitive conclusions. There are still some lessons that have already been learned and that I can present here.

A spreadsheet is a good XML design tool

The first one is that the creating a repository of information about a vocabulary grouping content models, correspondence with other vocabularies and styling information in a user friendly tool such as a spreadsheet has proved to be very convenient.

The idea isn't new and has been used by other projects such as UBL. What's different from the usage done in UBL that was more focused on defining schemas is the combination of different type of information in a single location.

Restriction and translation can be done in one step

The other lesson is that the combination of a restriction and a translation is something practical that can be done in a single operation.

Users still need to accept the design decisions of the base vocabulary

On the not so bright side, the difficulties we've had (and are still having) are linked with the decision of relying on a powerful and complex vocabulary such as DocBook instead of defining a brand new language.

We do think that it's worth for such a project that has not enough resources to create its own set of resources to publish documents to rely on an existing vocabulary, but the price to pay is high in term of frustration for the users who do not always perceive the need to cope with the complexity and rigidness of DocBook and that's creating some tensions during the definition and test of this subset.