xmlprague

XML Prague 2015

Conference Proceedings

University of Economics, Prague
Prague, Czech Republic

February 13–15, 2015

XML Prague 2015 – Conference Proceedings
Copyright © 2015 Jiří Kosek

ISBN 978-80-260-7667-4

<oXygen/>® XML Editor

The Complete XML Development and Authoring Solution

XML Authoring

<oXygen/> XML Author is the most efficient solution for implementing single source publishing and content reuse.

XML Development

<oXygen/> XML Developer is specially tuned for XML development, providing the best coverage of today's XML technologies.

- **Visual XML Authoring**
- **Single Source Publishing**
- **Multiplatform Desktop and Web**
- **Highly Customizable**
- **Supported by Major CMSs**

- **Intelligent XML Editing**
- **Visual Schema Modeling**
- **XSLT and XQuery Debugging**
- **XML Databases Support**
- **Web Services Analyzer**

www.oxygenxml.com

Availability

<oXygen/> is available in three editions: **<oXygen/> XML Author** for content authors, starting from 349 USD, **<oXygen/> XML Developer** for XML developers, starting from 349 USD and **<oXygen/> XML Editor** for XML developers and content authors, starting from 488 USD.
For Academic/Non-Commercial use **<oXygen/> XML Editor** is available at a discounted price of 99 USD.
All editions can run as a standalone application or as an Eclipse IDE plugin, on Windows 8, Windows 7, Vista, XP, 2000, Mac OS X, Linux and Solaris.

Table of Contents

General Information

Date

Friday, February 13th, 2015 (preconference day)
Saturday, February 14th, 2015
Sunday, February 15th, 2015

Location

University of Economics, Prague (UEP) – Vencovského aula
nám. W. Churchilla 4, 130 67 Prague 3, Czech Republic

Organizing Committee

Petr Cimprich, *Xyleme*
Vít Janota, *Xyleme*
Jirka Kosek, *xmlguru.cz & University of Economics, Prague*
Pavel Kroh, *pavel-kroh.cz & Macness.com*
Mohamed Zergaoui, *ShareXML.com & Innovimax*

Program Committee

Robin Berjon, *W3C*
Petr Cimprich, *Xyleme*
Jim Fuller, *MarkLogic*
Michael Kay, *Saxonica*
Jirka Kosek (chair), *University of Economics, Prague*
Uche Ogbuji, *Zepheira LLC*
Adam Retter, *freelance consultant*
Felix Sasaki, *DFKI / W3C Fellow*
John Snelson, *MarkLogic*
Eric van der Vlist, *Dyomedea*
Priscilla Walmsley, *Datypic*
Norman Walsh, *MarkLogic*
Mohamed Zergaoui, *Innovimax*

Produced By

XMLPrague.cz (http://xmlprague.cz)
Faculty of Informatics and Statistics, UEP (http://fis.vse.cz)
Ubiqway, s.r.o. (http://www.ubiqway.com)

Sponsors

Gold Sponsor

Mark Logic Corporation (http://www.marklogic.com)

Sponsors

oXygen (http://www.oxygenxml.com)
Mercator IT Solutions Ltd (http://www.mercatorit.com)
le-tex publishing services (http://www.le-tex.de/en/)
appsoft Technologies GmbH (http://www.xeditor.com)
Saxonica Ltd (http://www.saxonica.com)
OverStory Consulting Ltd (http://www.overstory.co.uk/)

Preface

This publication contains papers presented during the XML Prague 2015 conference.

In its tenth year, XML Prague is a conference on XML for developers, markup geeks, information managers, and students. XML Prague focuses on markup and semantic on the Web, publishing and digital books, XML technologies for Big Data and recent advances in XML technologies. The conference provides an overview of successful technologies, with a focus on real world application versus theoretical exposition.

The conference takes place 13–15 February 2015 at the campus of University of Economics in Prague. XML Prague 2015 is jointly organized by the XML Prague Organizing Committee and by the Faculty of Informatics and Statistics, University of Economics in Prague.

The full program of the conference is broadcasted over the Internet (see http://xmlprague.cz)—allowing XML fans, from around the world, to participate on-line.

The conference starts with a pre-conference day which provides space for various XML community meetings in three parallel tracks. During the weekend classical single-track format is used and papers from it are published in the proceeedings. Additionally, we coordinate, support and provide space for W3C XSLT and XQuery working group meetings collocated with XML Prague.

Last but not least—this year the conference celebaretes its 10th anniversary. We have not even imagined that XML Prague one day becomes one of the largest and the most respected XML events when we have been preparing the first conference ten years ago.

We hope that you enjoy XML Prague 2015.

— Petr Cimprich & Jirka Kosek & Mohamed Zergaoui
XML Prague Organizing Committee

Parallel Processing in the Saxon XSLT Processor

Michael Kay
Saxonica
`<mike@saxonica.com>`

Abstract

One of the supposed benefits of using declarative languages (like XSLT) is the potential for parallel execution, taking advantage of the multi-core processors that are now available in commodity hardware.

This paper describes recent developments in one popular XSLT processor, Saxon, which start to exploit this potential. It outlines the challenges in implementing parallel execution, and reports on the benefits that have been observed.

1. Introduction

In recent years, increased hardware speeds have been achieved largely by packing more processors onto a chip. To take full advantage of the processor capacity, therefore, it is necessary to take advantage of parallelism. Languages that can exploit parallel processing, ideally "under the hood" without involvement of the programmer, therefore have great appeal.

The potential for parallel execution has always been one of the justifications for making XSLT a purely declarative language. Declarative languages, without mutable variables, give the compiler much more freedom to rearrange the order of execution, including the ability to perform tasks in parallel. To take a simple example, XSLT's `<xsl:for-each>` "loop" is not actually a loop in the sense of traditional procedural programming languages; rather it is a mapping operator: it applies a function (the body of the `<xsl:for-each>` instruction) to an input sequence (the result of evaluating the expression in the *select* attribute). As demonstrated by the popularity of the map-reduce paradigm, a functional mapping operation is an ideal candidate for parallelisation. While most XSLT processors today will process the selected items one by one, in order, there has always been the freedom to process them in a different order, or in parallel. The difficulty, of course, is to decide when this is an appropriate strategy.

The connection between declarative languages and parallel processing is not new. See for example [11], or [1], or [7]. For example, Chakravarty writes:

> *Declarative programming languages have long been seen as good candidates for programming parallel computers. Their clean semantics makes them suitable for optimizing program transformations, on the source level and during compilation. However, researchers have found it difficult to present efficient parallel execution models for declarative languages. Generally speaking, it appears to be impossible to automatically identify the exact parallelism that leads to a reduction in execution time, and the irregular access to dynamic data structures can result in considerable overhead on distributed memory machines.*

The idea of exploiting parallelism has been mooted since the earliest days of XSLT: here for example is Eric Ray writing on the xsl-list forum on 1 Oct 2000:

> *The subtree processing model of XSLT seems to make it a good application for parallel processing (i.e. using multiple CPUs to process different subtrees simultaneously). Since many people have remarked on the inherent slowness of XSLT processors, I wonder if anyone has succeeded in creating an XSLT processor that successfully divides the work among different processors, resulting in a gain of processing speed. Has anyone tried this? Some of the implementations are written in Java, such as XT. Do they use multi-threading and can they take advantage of multiple CPUs?*

Research attempts at parallel execution of XSLT transformations have been reported. See for example [10], or [8] et al. A characteristic of these systems is that the entire architecture of the processor is designed from the ground up to support parallelisation. While this can yield useful research results, there is a danger that from an engineering perspective, other objectives get sacrificed.

In the commercial domain, there are high-end XSLT processors from IBM and Intel, marketed as hardware-assisted XSLT accelerators, which may well make use of parallel processing internally, but if so, no details are available in the public domain. Altova's marketing literature for RaptorXML intriguingly claims "the engine takes advantage of today's ubiquitous multi-CPU computers to deliver lightning fast processing of XML and XBRL data"; but it is hard to find any technical details on how it does so.

This paper describes the incremental approach to parallelism adopted in the Saxon product (see [9]). Saxon is a widely used XSLT and XQuery processor available in both open-source and commercial editions. Unlike some of the research vehicles described in the literature, Saxon is not designed from the ground up for parallel processing, and there has been no attempt to make radical changes to its architecture to take advantage of multi-core computers. But where opportunities for parallel processing have presented themselves, they have been grasped, and for some workloads they deliver substantial benefits. This paper describes how parallel processing is used in Saxon today, and explains some of the benefits that can be achieved, and the challenges that need to be overcome.

2. Running Multiple Transformations in Parallel

The most trivial form of parallelism, which has probably been offered by all XSLT processors since day one, is the ability to apply the same stylesheet independently to several source documents at the same time. Although this capability is quite easy to achieve, and is probably taken for granted by most users, it is worth saying a few words about it, firstly because it delivers substantial benefits, and also because it creates a few challenges.

Clearly some workloads will benefit immensely from this approach. XSLT is often used server-side on high-throughput web publishing platforms to render XML documents on demand into HTML for viewing on the browser. Typically such platforms have a small number of XML document types, with a large number of instances of those types, and the same XSLT stylesheets are used with high frequency. In such an environment, there is considerable benefit in compiling the stylesheet to efficient code (actual machine code or something higher level), and in executing that code in parallel threads to meet the throughput and response time targets of the online user community.

Saxon has always been optimized for this kind of workload. Perhaps excessively so: figures published by [5] show that while Saxon's run-time performance ranks with the best, this is sometimes at the expense of relatively poor compile time performance, which can be attributed to the amount of time spent optimizing. This is a good strategy for this server-based workload, but a poor one for single-shot adhoc processing where the cost of compiling a large family of stylesheets, such as those used for the DITA or DocBook vocabularies, may dwarf the cost of a typical transformation.

Although parallel execution of independent transformations may appear trivial, it is not without its complications. Two problems in particular have been recurrent over the years: contention, and reliability.

2.1. Contention

A very intensive operation in performing an XSLT transformation is the matching of element and attribute names appearing in the source document with names used in the stylesheet. Comparison of strings is slow, especially when they include lengthy namespace URIs. To eliminate this cost, Saxon has for many years implemented a *NamePool* which maps strings to integer codes, so during execution, the string comparisons are replaced by much faster integer comparisons. Of course, the same mapping must be used when a stylesheet is compiled and when a source XML document is parsed into a tree representation. But because a single compiled stylesheet can be used to transform multiple XML documents, this means that all the XML documents must use the same integer codes, and this means that the

NamePool used to allocate these codes is a shared resource, and as such it can suffer contention. This has been known, in some cases, to cause a significant bottleneck.

A number of techniques have been used to reduce this problem. In earlier releases, all QNames were represented by integer codes, including for example the names of variables and functions, and names in the result document. Today these codes are used only in source documents to which XPath processing is applied. In addition, the module that builds source documents uses caching techniques to minimise contention: synchronized access to the *NamePool* has therefore been greatly reduced in the common case where the vocabulary of names reaches steady state quickly. Nevertheless, it remains a potential bottleneck. There is scope to reduce contention further by partitioning, for example by creating one *NamePool* per namespace, but this can only be a partial solution. A more radical approach has been considered, in which there is a mapping table (one per transformation) from integer codes used in the stylesheet to (different) integer codes used in the source document. This indirection could noticeably reduce transformation speed, but if it increases the scope for parallelism, it could be worthwhile. This lesson illustrates the need to find engineering compromises between different objectives and a variety of workloads; the danger with a research project that focuses on parallelisation to the exclusion of all else is that it fails to achieve a balance.

Contention of course becomes even more of a problem once parallel processing is used within a single transformation. In fact, it becomes the limiting factor on what can be achieved.

Another point worth mentioning here is that the need to avoid contention tends to impose a design where stylesheet compilation and optimisation is completed before execution starts. This way, the data structures representing the compiled stylesheet, whatever form they take, are read-only and therefore contention-free at execution time. However, as we have seen, there are workloads where compiling everything before execution starts is far from optimal. In the massive stylesheets that come with DocBook or DITA, most of the template rules define processing for elements that rarely or never occur in a typical source document; effort spent compiling and optimising these template rules is wasted if they are never used. A just-in-time compilation approach in such cases has many attractions; but it also runs the risk of increasing contention when used in a shared workload.

2.2. Reliability

Even with the very limited form of multi-threading described in this section, there has been a steady trickle of bugs over the years. These bugs are rare, but potentially devastating. They often take years to discover (because the occurrence is probabilistic), and when they do occur they are hard to diagnose. It is often impossible to reproduce the problem “in the lab”, that is, anywhere other than the site running a production workload. One such bug a year is too many. We have been very fortunate that the users who discovered these bugs have had the technical competence and commercial patience to take the lead in collecting the data needed to solve them.

Preventing such bugs arising is not easy (see for example [12]) Quote: "Creating software that can be run by multiple threads concurrently is a daunting task—dwarfed only by the act of testing that code". Saxon is implemented in Java, and switching to a different language is not a realistic option, given the existence of around 250K lines of code. Even if it were an option, it's not clear that a different language would really help. Java offers the basic primitives needed to coordinate multiple threads; the problem is that it offers very little in the way of tooling to ensure that a complex program is thread-safe. The basic discipline to ensure that multiple executions of the same stylesheet can run concurrently is very simply stated: code that runs at execution time must not modify the expression tree. One can envisage tools (assisted by annotations in the code) that check such an assertion statically, but we are not aware of any. Saxon includes about 400 classes that interact directly with the expression tree, and if we rely on programmer discipline alone, mistakes will occasionally happen.

(Having said this, we could do better with soak testing. We should probably have a test where we run each of the 10,000 stylesheets in the W3C test framework concurrently in a dozen threads for 24 hours or so, and check that each thread produces correct output. As it is, our concurrency testing is a woefully small part of our total test programme.)

Again: if reliability is imperfect with the relatively trivial parallelism described in this section of the paper, then we need to be extremely cautious about introducing more ambitious parallelism, because reduced reliability is not a price we are prepared to pay for any performance benefits.

3. Multi-threading and Streaming

While memory sizes appear astronomic compared with a few years ago, the size of data files that people want to transform grows at a similar rate, and there will always be a handful of users who need to transform files that are too big to fit in physical memory. Streamed XSLT processing, which avoids the need to build a tree representation of the source document in memory, has therefore been an increasing area of focus in recent years. It is the main focus of XSLT 3.0 ([13]), and is a major area for implementation work in Saxon ([4]).

Multi-threading and streaming are not orthogonal. Indeed, many of the opportunities for multi-threading become more difficult when processing has to be streamed: it is then no longer possible, in the terms used by Eric Ray cited above, to process different subtrees in parallel, because this relies on buffering data in memory. (But for a counter-argument to this assertion, see the paper by Jakub Malý at this conference: [6]). However, all is not lost.

The first use of multi-threading in Saxon was in fact to implement a form of streaming. This provided a mode of processing in which the source document was split into a sequence of subtrees, and each subtree was transformed independently (this is still a simple and useful processing model that is often good enough to solve the streaming requirement). The reason for using multi-threading was primarily to solve a push-pull conflict in the processing pipeline: see [3] and [4]. We refer to a software component as operating in pull mode when it performs a sequence of read operations to obtain its input, and as operating in push mode when it is invoked repeatedly by a supplier of data to process data as it becomes available. A conflict arises when two components in a pipeline both want to be in control: in this case, an XML parser which wants to push data to the XSLT/XPath processor, and an XPath processor which wants to pull data from the XML parser. One solution is to run the first component (the XML parser) to completion, putting all the data in memory, before starting execution of the second component (the XSLT processor). This is the traditional architecture of today's XSLT processors. An alternative solution, adopted in Saxon, is to use two threads for the two processes, passing data from one to the other via a synchronized queue.

But although this approach breaks the document into subtrees that are processed independently, in the current implementation they are processed sequentially rather than in parallel. There are only two threads, one parsing and building the subtrees, the other processing them one at a time as they become available. It would be difficult to split the parsing thread into multiple threads, because it reads the input data sequentially. Splitting the processing thread would be easier, though it would still need coordination to ensure that results are written to the final result tree in the right order.

This approach has fallen into disuse in more recent releases, though it is still used in some cases, for example in the streamed implementation of the new XSLT 3.0 `<xsl:merge>` instruction. The reason is that it is no longer required: the push-pull conflict has been eliminated by rewriting the XPath engine to operate in push mode, accepting input in the form of events triggered by the XML parser. Any performance benefits obtained by running two threads rather than one were an incidental part of the design (the main objective being to reduce memory requirements). It would of course be possible to continue running the parser and XSLT processor in separate threads even when there is no push-pull conflict forcing this, but we would need to make careful measurements to ensure that this actually delivered benefits.

4. Multi-threading in Saxon Today

In the current Saxon release (9.6) there are four main ways multi-threading is used, and they will be described in this section. In all cases, multi-threading is a feature offered only in the Enterprise Edition of the product.

4.1. The collection() function

The *collection()* function reads a set of input files. The W3C specification is deliberately vague about what constitutes a collection, because it needs to accommodate a variety of different database architectures. Although the facility was designed to allow searching a collection of documents held in an XML database, it is also very useful for transforming a collection of raw documents held in filestore (for example, I have a stylesheet that transforms the thousands of documents making up the W3C XQuery test suite into a set of tests suitable for testing XSLT).

By default, Saxon-EE implements the *collection()* function in multiple threads. A pool of threads is allocated (we choose a number based on the number of CPUs available, for want of any better indicator), and the parsing of the source documents making up the collection is distributed among these threads. The XPath expression that invoked the *collection()* function receives the parsed documents in the order in which parsing is completed.

This is a very straighforward use of multi-threading for a task that is easily distributed. There is very little scope for contention (the only shared resources being the *NamePool*, discussed above, and the queue on which each parsing thread places the parsed document on completion). Because XML parsing cost can often dominate transformation cost, the benefit is high, and the risks in terms of contention and reliability are low.

There are decisions to be made about the order of results. Although W3C does not mandate that collection results are delivered in any particular order, users may have an expectation about the order. Another complication is that an expression like `collection()/doc` is mandated to deliver results in document order, which is somewhat arbitrary, but it cannot be assumed that this is simply the order in which the results become available. (Smart users will write `collection()!doc` to avoid any risk of triggering a sort; but not all users are this smart, and some will deliberately prefer a construct that works in XPath 2.0 as well as 3.0.)

In Saxon 9.6, the multi-threading of the *collection()* function was implemented in the default *CollectionURIResolver* class, which is tasked with taking a URI as input and delivering a sequence of documents as output. There are two drawbacks to this design. Firstly, multi-threading doesn't work if the user substitutes their own *CollectionURIResolver*, which is a perfectly reasonable thing to do. Secondly, the approach is incompatible with streaming. If we want each of the documents in the collection to be processed using streaming, then having a *CollectionURIResolver* that pre-builds each document in memory scuppers this. The design has therefore been changed for Saxon 9.7.[1] XSLT 3.0 introduced a new function *uri-collection()* to handle this case. In the new design, the *CollectionURIResolver* returns (synchronously) a sequence of URIs, and the stylesheet can then process the collection either by constructing

[1]Anything this paper says about future releases is subject to change without notice.

in-memory documents (using the *collection()* function) or, for example, by streaming: the code might be written:

`<xsl:for-each select="uri-collection('my-dir')"> <xsl:stream href="."> <xsl:apply-templates mode="streaming"/> </xsl:stream> </xsl:for-each>`

This change puts the responsibility for multi-threading onto the *collection()* function or the `<xsl:for-each>` instruction respectively.

The performance benefits of multi-threading the *collection()* function can be illustrated by a simple experiment. The query

`count(collection('shakespeare')//LINE)`

took 160ms to count all the `LINE` elements across the corpus of Shakespeare's plays without multi-threading, reducing to 80ms with multi-threading enabled. In this test, 8 threads were used.

4.2. Multiple result documents

When Saxon-EE encounters an `<xsl:result-document>` instruction, it starts a new thread to process it. The original thread continues processing with the next instruction after the `<xsl:result-document>`. When a transformation produces multiple result documents, they are therefore produced in parallel.

This use of multi-threading is considerably more complex, because we now have different instructions in the stylesheet executing simultaneously. It is simplified, however, by the fact that the output of each thread is written (typically serialized to disk) independently of the other threads, so there is no need to combine the outputs of different threads on completion. Nevertheless, there can be interactions between threads. These mainly arise because of the use of lazy evaluation. The different threads can access the same local and global variables, which would be fine if variables really were immutable, but internally, Saxon evaluates variables lazily (and progressively), so access to variables needs to be synchronized. This applies only to variables declared outside the scope of the `<xsl:result-document>` instruction; for variables inside its scope, each thread has its own copy. Any apparent cost that might arise from repeated evaluation of the same variable is eliminated by Saxon's compile-time optimization rewrites, which use loop-lifting to extract expressions from loops if their value is not dependent on the looping variables.

Another complication which might not be immediately obvious is the use of the XSLT 3.0 *try/catch* mechanism to recover from dynamic errors that occur during the execution of the `<xsl:result-document>` instruction. This is the only way that the spawned thread can affect anything that happens in the original thread. Before an `<xsl:try>` instruction completes, it must check that all threads spawned within its scope have completed successfully, and if necessary, it must wait for them to complete. In fact dynamic errors also need to be considered even in the absence of *try/catch* instructions, because the top-level invocation of the transformation via an API call such as *transform()* needs to throw an exception if any dynamic error has occurred in the transformation. Although we could make the concurrency visible at the application level, we choose not to: the *transform()* method does not return until all threads have completed, and if any thread raises a dynamic error, the call to *transform()* throws an exception.

Executing multiple instructions simultaneously has various other implications which are perhaps mundane, but worth mentioning because getting the detail right can be a lot of effort. One such detail, for example, is the need to ensure that messages produced by different threads (using `<xsl:message>`) are not intermingled in a log file, at least to the extent that the output from each invocation of `<xsl:message>` retains its integrity.

Another detail is the evaluation of the *last()* function: if one result document is produced for each element in some input sequence, then it is quite possible that the *last()* function will be called within the scope of the `<xsl:result-document>` instruction. When *last()* is evaluated against a particular sequence, Saxon has a number of strategies; if the sequence is the result of a path expression, then the path expression will be evaluated twice, once to compute the value of *last()* (which is then retained for future use), and once to retrieve the actual elements. So the various threads handling different items in the input sequence need to co-ordinate with each other to ensure that the cached value of *last()* is shared between them.

Attempting to measure the effect of this optimization, it appears that the effect depends on how much computation is actually done within the `<xsl:result-document>` instruction. In transformations that are merely splitting the input into multiple outputs (where the body of `<xsl:result-document>` is nothing more than an `<xsl:copy-of>` instruction) it appears to make very little difference to the total elapsed time. This appears to be because the transformation time is limited by the I/O activity of reading the input, and creating and writing the serialized output files. In other cases, where more intensive transformation work is involved, we will often see a doubling of overall execution speed.

4.3. Multi-threaded <xsl:merge>

The new `<xsl:merge>` instruction in XSLT 3.0 allows pre-sorted input files to be merged, using streaming to avoid building a tree representation of the files in memory. An example application would be merging the transaction logs from multiple sales outlets into a single transaction log, ordered by time-stamp.

Saxon's implementation of `<xsl:merge>` uses one thread for each input file. This allows Saxon to use SAX (push-based) parsing, as well as spreading the load over multiple processors. There's no intrinsic reason why several StAX (pull-based) parsers couldn't be instantiated in a single thread, one per input file, in which case Saxon could pull data from each one as required without the use of multiple threads; but using multiple push parsers is convenient both because of the performance benefits of spreading the workload, and also because of the engineering benefits of using the same approach to parsing source documents that is used in other parts of the product.

The design of `<xsl:merge>` is such that each input source delivers a sequence of *snapshots* — subtrees of the source document. Each snapshot is built by the parser (as a small in-memory tree) and is then placed on a shared queue. The `<xsl:merge>` process examines these queues (one per source document) and selects the next one for processing based on the values of the merge keys.

At this stage we have not made any performance measurements for `<xsl:merge>`, either with or without streaming or parallel processing. It's probably a feature of minority interest: the capability is important if you need it, but not everyone does. So it hasn't been at the top of our list for optimization.

4.4. Multi-threaded <xsl:for-each> and <xsl:apply-templates>

The main XSLT instructions used to process a sequence of nodes from the input tree are `<xsl:for-each>` and `<xsl:apply-templates>`. In both cases Saxon allows the user to request multi-threading by means of a vendor extension attribute, *saxon:threads="N"*. Unlike the other facilities described in this section, there is no multi-threading "out of the box" in this case; it is available only on request.

This facility essentially allows map-reduce applications to be written in XSLT, with parallelism under the control of the user rather than the compiler. This is not necessarily a disadvantage; users may be able to achieve better results than a system optimizer.

This design is a cautious one. We know that a feature like this will be ignored by 95% of users. To some extent this is our aim, because we know the feature is dangerous. There's a danger of bugs, but more particularly, there's a danger of misuse. We have no idea how many threads to allocate to such an instruction, so we leave it to the user to decide; but we know that most users have no idea either. The more adventurous will hopefully find a good design by trial and error, knowing how to measure the effect on their particular workload. There will probably be a few who see the feature, guess a number, and never test their assumptions, but such users deserve what they get. Hopefully we will slowly get experience and feedback of what works well and what doesn't, and perhaps rules of thumb will emerge that are sufficiently sound for us to automate the process. Perhaps use of a fixed value is the wrong approach anyway; perhaps `<xsl:for-each>` should allocate N-M threads where N is the some maximum for the transformation as a whole, and M is the number of threads currently active. Only experiment, with a variety of realistic benchmarks, will provide the answer.

Unlike the use of multiple threads for the *collection()* function (multiple input files) and the `<xsl:result-document>` instruction (multiple output files), its use on `<xsl:for-each>` and `<xsl:apply-templates>` instructions creates a serious risk that performance is degraded by the cost of interaction between the threads. These instructions are defined by the language semantics to deliver their results in a particular order, and this means that the results of each thread must be saved in memory and reassembled in the correct order before the instruction completes. This cost can be significant, bearing in mind that XSLT instructions will normally stream their results directly to the serializer, without building temporary trees in memory. Nevertheless, one of our users has reported a reduction in the elapsed time of a heavy transformation by a factor of three by using 8 threads in an `<xsl:for-each>` instruction.

5. Futures

I have described the ways in which Saxon uses multi-threading today. What of the future?

We need to consider developments in two categories: internal use of multi-threading to support operations such as the *collection()* function or the `<xsl:merge>` instruction, and language features that allow users to take advantage of multi-threading, along the lines of the existing multi-threaded `<xsl:for-each>`.

In the first category, a natural candidate is a multi-threaded sort: both for explicit `<xsl:sort>` elements, and for the implicit sorting that occurs when a sequence of nodes needs to be delivered in document order. Saxon uses an implementation of QuickSort, and this lends itself well to parallel implementation. Another candidate might me a multi-threaded implementation of `<xsl:for-each-group>`.

In the second category, we will be guided by user experience with the facilities we already provide. It may well be that the need now is not for more multi-threading features, but for instrumentation to help users establish whether their multi-threading strategies are proving effective. Until we get more feedback on how the features work in practice, I don't see us introducing more automatic multi-threading; I can't see Saxon deciding to use multi-threading for `<xsl:for-each>` or `<xsl:apply-templates>` without an explicit user request.

Another interesting instruction with multi-threading possibilities is the new `<xsl:fork>` instruction in XSLT 3.0. This was developed for use with streaming, and allows several actions to operate on a single pass of a streamed input document. The current Saxon implementation is not multi-threaded (input events are passed to each of the actions in turn, and the actions are performed sequentially. But a multi-threaded implementation would be very natural.

6. Conclusions

In this paper I have described the facilities in the current Saxon release (specifically, Saxon-EE 9.6) to allow multi-threaded stylesheet execution. A few users are already getting substantial benefits from the use of these features, but they are not widely known about or understood. Hopefully this paper will help to increase awareness.

What is needed now is for users to report their experiences, to experiment and report their results, and for the product to improve in response to this feedback.

References

[1] Manuel M. T. Chakravarty. On the Massively Parallel Execution of Declarative Programs Ph. D. Dissertation, Technische Universität Berlin, 1997. http://www.cse.unsw.edu.au/~chak/papers/diss.ps.gz

[2] Kay, Michael. Anatomy of an XSLT Processor. Published online by IBM DeveloperWorks https://www.ibm.com/developerworks/library/x-xslt2/

[3] Kay, Michael. You Pull, I'll Push: On the Polarity of Pipelines. Presented at Balisage: The Markup Conference 2009, Montréal, Canada, August 11 - 14, 2009. In Proceedings of Balisage: The Markup Conference 2009. Balisage Series on Markup Technologies, vol. 3 (2009). doi:10.4242/BalisageVol3.Kay01. http://www.balisage.net/Proceedings/vol3/html/Kay01/BalisageVol3-Kay01.html

[4] Kay, Michael. Streamability in Saxon. XML Prague 2014. http://archive.xmlprague.cz/2014/files/xmlprague-2014-proceedings.pdf

[5] Kay, Michael and Lockett, Debbie. Benchmarking XSLT Performance. XML London 2014. http://www.saxonica.com/papers/xmllondon-2014mhk.pdf

[6] Malý, Jakub. Parallel XSLT Processing of Large Documents. XML Prague 2015. http://archive.xmlprague.cz/2015/files/xmlprague-2015-proceedings.pdf

[7] Rishiyur S. Nikhil (Bluespec) and Arvind (MIT). Making the transition from sequential to implicit parallel programming: Part 5 Online newsletter, UBM Electronics, Sept 2007. http://www.embedded.com/design/mcus-processors-and-socs/4007173/Making-the-transition-from-sequential-to-implicit-parallel-programming-Part-5

[8] A Scalable XSLT Processing Framework based on MapReduce Journal of Computers, Vol 8, No 9 (2013), 2175-2181, Sep 2013 10.4304/jcp.8.9.2175-2181 http://www.ojs.academypublisher.com/index.php/jcp/article/view/jcp080921752181

[9] Saxonica: XSLT and XQuery Processing http://www.saxonica.com/

[10] Tianyou Li ; Qi Zhang ; Jia Yang ; Yuanhao Sun Parallel XML Transformations on Multi-Core Processors IEEE International Conference on e-Business Engineering, 2007. ICEBE 2007.

[11] Philip W. Trinder, Kevin Hammond, James S. Mattson Jr., Andrew Partridge, and Simon L. Peyton Jones. GUM: a Portable Parallel Implementation of Haskell Proceedings of Programming Languages Design and Implementation,

Philadelphia, USA, 1996. https://www.macs.hw.ac.uk/~dsg/gph/papers/ps/gum-ifl95.ps

[12] Watts, Nick Getting Started: Testing Concurrent Java Code Online Blog, July 2011. http://thewonggei.com/2011/07/18/getting-started-testing-concurrent-java-code/

[13] XSL Transformations (XSLT) Version 3.0. W3C Working Draft, 2 October 2014. Ed. Michael Kay. http://www.w3.org/TR/xslt-30

Parallel XSLT Processing of Large Documents

Jakub Malý
Barclays
<jakub@maly.cz>

Abstract

After the introduction of streaming in XSLT 3.0, new possibilities and applications for XSLT opened up. Streaming stylesheets can process documents with bounded memory consumption, even large documents that would not fit into memory when a non-streaming processor is used. With bounded memory consumption and disc space de-facto unlimited (and SSD drives providing fast access to stored data), CPU speed can become a bottleneck in many scenarios. However, contemporary commodity machines have 2, 4 or more CPU cores, of which only one is usually used by present day XSLT processors (and the CPU is therefore underutilized). In this paper, we discuss scenarios in which optimal CPU utilization can be achieved by processing the input file in a parallel manner. As the experiments show, such an approach can significantly increase the performance (=shorten run time) of a transformation by up to ~35% in our experimental setting.

Keywords: XSLT, parallel processing, streaming, optimization

1. Introduction

Streaming is an essential part of XSLT 3.0 [3], which in Last Call Working Draft 2 state at the time of writing this paper. The introduction of streaming has broaden the use cases for XSLT as a technology. The transformation scenarios are no longer limited by the size of available memory (which can be a multiple of the size of the input document, depending on implementation). The stylesheet author can choose to process the input in streaming manner, which will ensure a bounded memory consumption not proportional to the size of the input. Thus, XSLT processors can work with larger documents than before, even on common hardware. Of course, the option of streaming comes with a price.

One thing to consider is the actual speed of processing, e.g. in the case of Saxon [5], it is reported that streaming mode is approx. 1.2 – 1.3 times slower than non-streaming [1]. But the real catch with streaming is that it implies limitations on the expressions and constructions allowed in the template itself. The majority of limitations comes from the fact that the input document is read only once and only limited portions of it can be buffered by the processor (like the current node, its attributes and ancestors). E.g. XPath preceding axis cannot be used in expressions, because it would require buffering or repeated reads of the whole document by the processor.

It is up to the user to mark that streaming mode is required and to author the templates in such a way that they do not violate streamability rules [2]. The processor checks the rules at compile time and will not start processing if they are not satisfied. The authors of the recommendation chose the approach of giving the user the possibility to use streaming, but he has to give up using some techniques and practices used routinely in nonstreaming stylesheets for the benefit of bounded memory consumption.

Processing a large document can take a significant amount of time. E.g. to run a simple stylesheet on a 700MB input file takes about 30s on a PC with Intel i7 2.4 GHz CPU with 4 logical cores, 8 GB of RAM and SSD. The key information for this paper is that even though the CPU speed is the bottleneck, the CPU is not fully utilized, because the used XSLT processor (Saxon EE 9.6, [5]) operates in single thread in streaming mode.

In this paper, we will experiment with running the transformation in multiple threads. We measure the gains in performance and discuss the limitations of this approach. As with streaming, parallel processing puts additional restrictions on the constructs allowed. Again, the user must sacrifice some of the flexibility and power of the language in order to apply this approach. We describe the class of scenarios, where parallel processing is applicable.

2. Motivation

For our examples, we will use the largest file in XML Data Repository [4] - Protein Sequence Database, a 683 MB XML file. The basic structure of the file is depicted in Figure 1. Basically, after some initial metadata, the document contains a long list of `ProteinEntry` notes, each containing details about one protein.

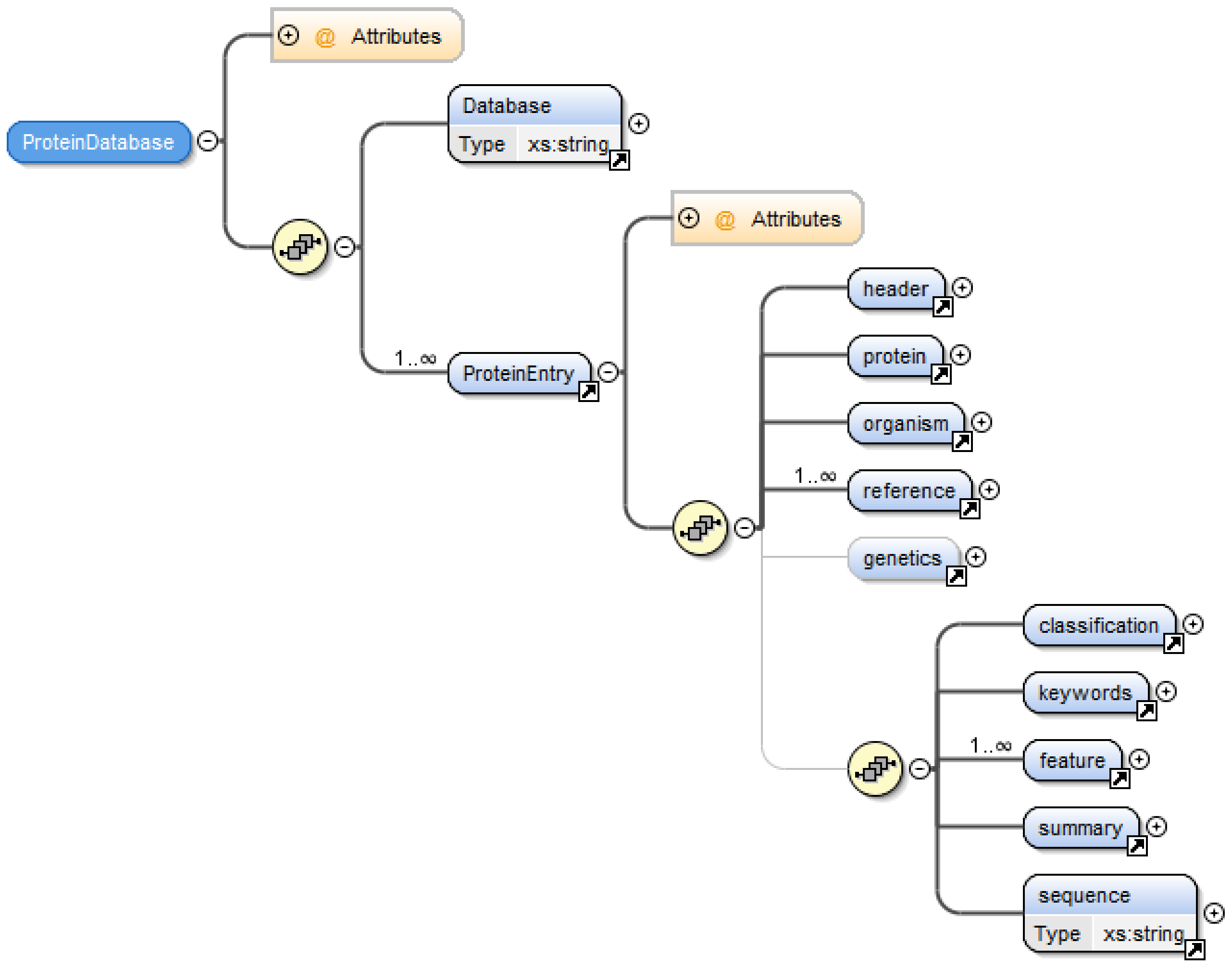


Figure 1. Protein Sequence Database Structure

From our knowledge of the data, we can infer several key properties of the document:

1. The document has a well-defined structure (schema)
2. A major part of the content is in a sequence of nodes of certain types (we will call these core types, in our example, there is exactly one – `ProteinEntry`).
3. Core types and their ancestors are not recursive.
4. Contents of core types are reasonably independent. This property is defined vaguely, what we mean is roughly that data from one core type instance does not reference other instances.

Now let us introduce a stylesheet to process this document. Stylesheet in Figure 2 performs a simple transformation – it preserves protein names and sequences and discards all the rest. Note that the stylesheet is streamable.

```
<xsl:stylesheet version="3.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:pxsl="http://jakubmaly.cz/pxslt">
    <xsl:output indent="yes" method="xml"/>
    <xsl:mode name="stream" streamable="yes"/>

    <xsl:template match="/ProteinDatabase" mode="#all">
        <ProteinDatabase>
            <xsl:apply-templates select="ProteinEntry"
                                 mode="#current"/>
        </ProteinDatabase>
    </xsl:template>

    <xsl:template match="/ProteinDatabase/ProteinEntry" mode="#all"
        pxsl:core="yes">
        <ProteinEntry id="{@id}">
            <xsl:apply-templates mode="#current"/>
        </ProteinEntry>
    </xsl:template>

    <xsl:template match="ProteinEntry/protein/name" mode="#all">
        <name>
            <xsl:value-of
                    select="replace(., '^\s*(.+?)\s*$', '$1')" />
        </name>
    </xsl:template>
    <xsl:template match="ProteinEntry/sequence" mode="#all">
        <sequence>
            <xsl:value-of select="normalize-space(.)" />
        </sequence>
    </xsl:template>

    <xsl:template match="text()" mode="#all"/>
</xsl:stylesheet>
```

Figure 2. Stylesheet for Protein Sequence Database

It is apparent that in our case the document could be processed in parallel – once we get to `ProteinEntry` nodes, we can split the long sequence of `ProteinEntry` nodes into several smaller sequences of roughly the same size and process each of these *core nodes subsequences* in a separate thread as a standalone subdocument. Finally, we concatenate the results. Attribute *pxsl:core* in the template for `ProteinEntry` denotes a template for a *core node*.

3. Experiment

Figure 3 shows how parallel processing affects performance. The tests were run on a PC with Intel Core i7-4500U CPU with 2 cores, 4 logical processors with maximum speed 2.4GHz, 8GB of RAM and an SSD. All times are measured when invoking Saxon using Java API and do not include the time required to load and compile the stylesheet. The first measurement is for Saxon without any modification. The second measurement shows that our approach brings some overhead when only 1 thread is used. Measurement with 2 threads shows the most prominent speedup (because of 2 physical cores being available on the test machine). Running with 4 threads shows an additional (smaller) increase in speed. Adding more threads does not bring an improvement on our test machine (no more than 4 threads can run at the same time), but we have every reason to assume that we would see continuously improving results on a machine with more CPU cores when adding more threads.

Our experiment clearly shows that at least in some cases, parallel processing can significantly speed up the transformation scenario. In the following sections, we describe how the parallel processor works and also the constraints of our approach.

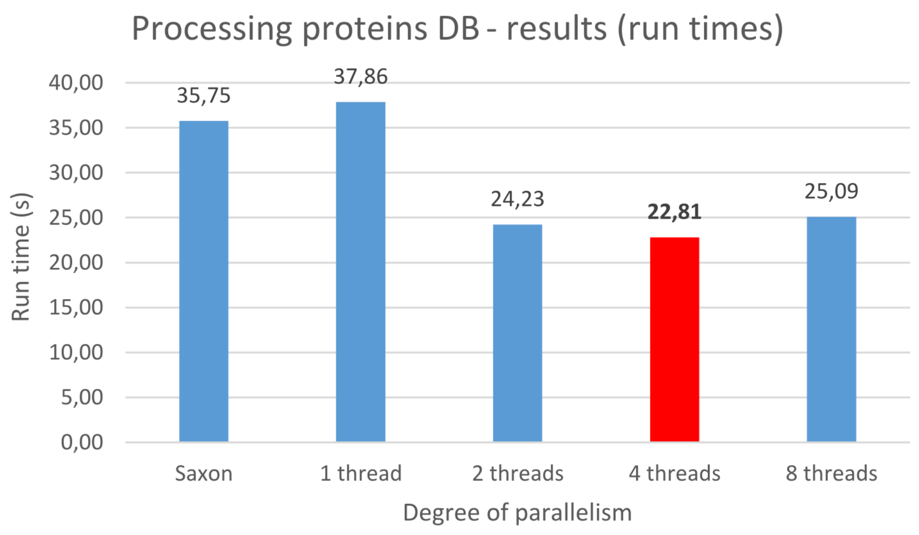


Figure 3. Parallel Processing Performance Gain

4. Parallel Processing XSLT Extension

To allow parallel processing, the user must provide additional information about the document. This comes in the form of marking the templates of core nodes (using `pxsl:core=yes`"). The processor must the right points to split the document, in our example these are between `<ProteinEntry>` and `</ProteinEntry>` tags. These split points (see Figure 4) have to be found by reading the document as text (without

XML parser). To apply the XSLT template to a split sequence (in our case sequence of `ProteinEntry` elements with their subtrees), the sequence must be inserted into a proper context, in our example the nodes have to be child nodes of `ProteinDatabase` element (otherwise match patterns like "`/ProteinDatabase/ProteinEntry`" would not work. This is also the reason why we require core nodes to be non-recursive. If they weren't, the processor would not be able to infer the context (should it be `/ProteinDatabase/ProteinEntry` or `/ProteinDatabase/ProteinEntry/ProteinEntry`?).

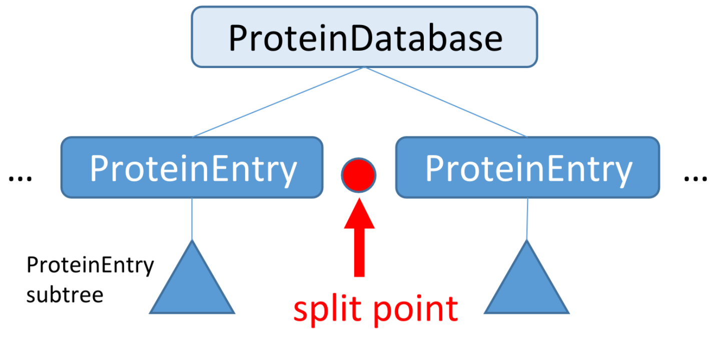


Figure 4.

How can the processor tell what context should be supplied? Either by using schema information or from the match pattern of the core node template (`match="/ProteinDatabase/ProteinEntry"` in our example). For this reason the core node match pattern must be absolute and use only child axis with element names.

```
1 <ProteinEntry>
2     ...
3 <!--
4 </ProteinEntry>
5 <ProteinEntry>
6     ...
7 -->
8 </ProteinEntry>
```

Figure 5. Comments problem

The simple process of finding split points described above is not reliable in every situation. Consider the piece of XML in Figure 5. If the parser starts searching for split points on line 4, it will find the end tag and place a split point between the end tag on line 4 and start tag on line 5. The problem of course is that those are not tags at all, but contents of a comment node. Similar problem arises with processing instructions and CDATA sections. Such situations cannot be easily solved without reading the whole document and since XML parsing can dominate transaction processing time, it is not suitable.

A trivial solution is just to report an error when a split point is accidentaly placed inside a comment/PI/CDATA (since it always will be discovered during the parsing later).

More involved solution would require some kind of light-weight parser and a phase of split-point verification, which will be optimized to handle exactly this problem. Such a parser could easily ignore some features of XML, such as schema validation, namespaces, attributes and text nodes. Whether this would reduce the cost of parsing enough to be actually feasible, we do not know. Another option would be to still treat the input as linear string and only be aware of the potentially problematic language elements, since those all also linear from the point of XML data model.

Since we are using streaming mode of the processor also for parallel processing, all limitations imposed by streaming apply for parallel processing too. There are also additional limitations. In streaming mode, ancestor nodes are available for XPath expressions. For parallel processing, we are creating some ancestor nodes artificially for the subdocuments (`ProteinDatabase`). These artificially created nodes cannot be used to provide data. For this reason, no expressions used in the core templates and templates invoked from them can traverse out of the core node subtree.

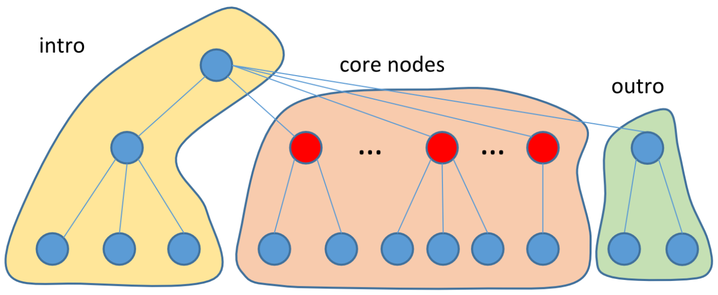


Figure 6. Document with Intro, Core Nodes and Outro

The limitation described above can be lifted in a special case, where the data in the document have a certain structure depicted in Figure 6. The documents has an intro part, which is relatively small compared to the size of the document, than a sequence of core nodes, which are all children of the same node from intro, and an outro part, again small relative to the size of the document. Parallel processing only deals with the core nodes section and can start after intro is processed. Because all core nodes have the same ancestry, the processor can first process the intro part and then start parallel processing of core nodes with the same initial context. In this case we are not creating the initial context artificially, but using the real one, so all ancestors of core nodes (from intro) can be accessed.

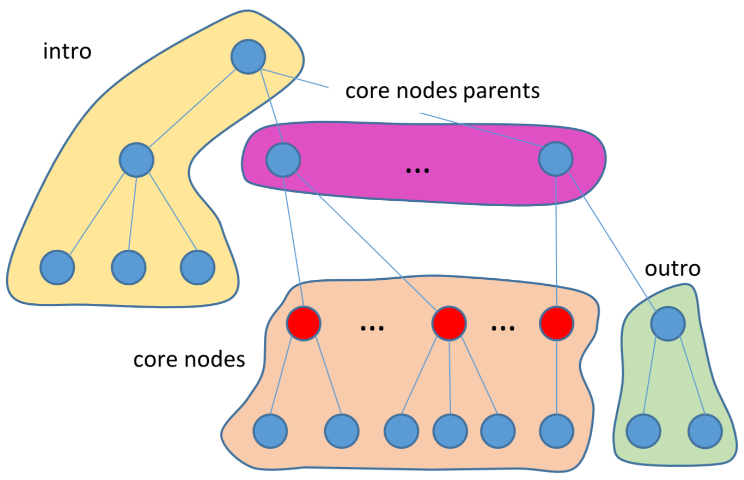


Figure 7. Core Nodes Without Common Parent

On the other hand, if the document structure resembles the one depicted in Figure 7, where core nodes are not children of the same node, the approach described above cannot be used (because it is no longer true that the data for the context of the core nodes comes before all the core nodes in document order). The solution for this case might be to choose the nodes in the pink area as core nodes.

Finally, our approach is applicable only when the input document can be read repeatedly (unlike streaming, which was designed with a requirement that the input document can be read only once). In our case, the document is accessed by several threads at once, each reading a different part of the document. For this reason, we also recommend reading the input from a medium with fast random access, such as an SSD.

5. Use Case

We will show another example of parallel processing large XML files. This use case comes from a problem we solved a couple of years ago without XSLT. The input is a file exported from a database managed by RUIAN – a Czech agency maintaining data about cities, streets, buildings, addresses etc. RUIAN uses an XML format and provides exports at the level of city. The export for the Czech capital Prague is a 614MB file. The file is in fact a long list of individual units (streets, buildings…) and the task was to extract these into separate files. In total it is about 700k files. Figure 8 shows the speedup achieved by running with parallel processing – from 42 minutes to 16 minutes. The stylesheet used for the transformation can be found in Appendix A. Note that the constraints imposed by the parallel processing requirements (notably allowing only absolute match patterns with named child steps for core nodes patterns) add to the templates complexity.

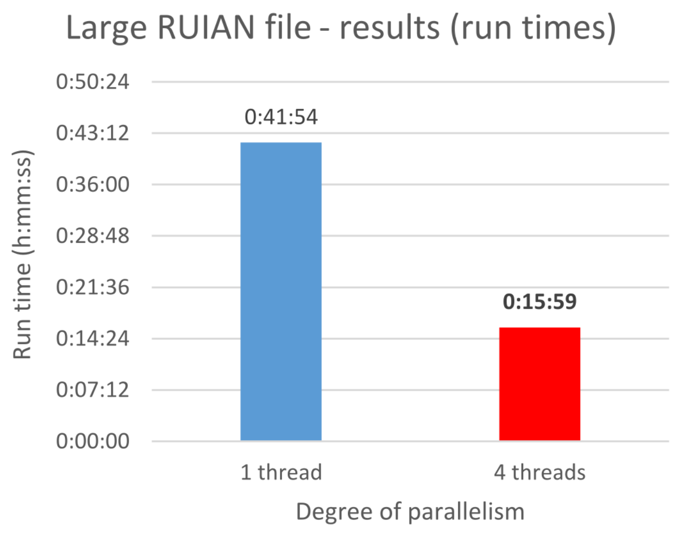


Figure 8. RUIAN processing results

6. Implementation

Currently, there is no open source streaming XSLT processor available, so instead of making the necessary changes in an XSLT processor, we created a thin wrapper application that uses Saxon EE for the actual processing. Our parallel processor works as follows:

1. Firstly, we identify possible points where the input document can be split. This is implemented trivially by dividing the input file into portions of roughly the same size and then shifting the dividing points to the beginning of the nearest core node.
2. Each portion of the document is prefixed and suffixed with a string required to make it into a well formed document with structure matching the structure of the original document. For our example, this is achieved by adding the prefix `<ProteinDatabase>` and suffix `</ProteinDatabase>` to each subsequence of `ProteinEntries`. This ensures that match patterns going out of the core node subtree, e.g. `/ProteinDatabase/ProteinEntry`, will continue to work.
3. Nodes created by adding prefixes and suffixes in step 2 should not affect the final output (e.g. otherwise multiple root `<ProteinDatabase>` element would appear in the final output). The experimental implementation does not deal with this issue.
4. An instance of Saxon is used to transform each sub-document
5. All the results are concatenated to get the final result.

This is a very crude and naïve implementation used only for demonstration of the method described in the paper. To implement the approach properly, changes would better be made in the processor itself (most notably to avoid steps 2 and 3).

There are also limitations. Some of them we discussed earlier (no access to the ancestors of core nodes in XPath). Another potential problem is the use of `xsl:message` instruction -- the output from these needs to be treated carefully to avoid interleaving messages from different threads.

Finally, the use of `xsl:accumulator` might deal unexpected (incorrect) results due to the fact that the accumulator might be accessed from one thread before other nodes that trigger the accumulator and appear before the current node have been visited. One of the motivations for accumulators was to support some data scenarios in streaming modes. Maybe the future will similarly bring new constructs specifically targeting parallel scenarios.

7. Multithrading capabilities already available in Saxon EE

Saxon transformations by default usually operate in single thread. However, there are some existing multithreading capabilities already supported by Saxon and available to the user. We will address them here.

In some cases, Saxon performs multithreading optimizations without any explicit declaration from the user to do so. This is the case of `collection()` function used to read a set of files from the file system. Saxon will process the collection using a threadpool (if running on a multi-CPU system). Output to multiple files (via `<xsl:result-document>`) is similarly parallelized.

In other cases, Saxon offeres multithreaded capabilities to the user, but only when the user explicitly requests it. The developer can use vendor-prefixed `saxon:threads="N"` attribute on `xsl:apply-templates` and `xsl:for-each` instructions.

It should be noted that neither of these functionalities is applicable in the use cases described in this paper, because Saxon does not allow multithreaded processing in streaming mode and we were targeting specifically the large documents that require streaming. If the transformed document has the properties we listed in Section 2, but it fits into the memory, using `saxon:threads` on the core nodes template will most likely be a better option than parallel processing described in this paper.

For the detailed discussion of multithreading in Saxon EE, please refer to paper [6] in this proceedings.

8. Conclusion

In this paper we examined the possibilities of parallel processing of XML documents. We have demonstrated a significant improvement in run times of XSLT transformations (up to 35% on a common PC, expected to be even better on a multi CPU machine) and discussed in which scenarios such an approach can be used. Similarly as in the case of streaming, our approach adds additional restrictions on what constructs and expressions are allowed.

Bibliography

[1] Kay, Michael: Streaming in the Saxon XSLT Processor. 2014. XML Prague 2014 proceedings, p. 81. http://archive.xmlprague.cz/2014/files/xmlprague-2014-proceedings.pdf

[2] Braaksma, Abel: XSLT 3.0 Streaming for the masses. 2014. XML Prague 2014 proceedings, p. 29. http://archive.xmlprague.cz/2014/files/xmlprague-2014-proceedings.pdf

[3] W3C: XSL Transformations (XSLT) Version 3.0. http://www.w3.org/TR/xslt-30/

[4] XML Data Repository.http://www.cs.washington.edu/research/xmldatasets/

[5] Saxonica: Saxon EE 9.6.http://saxonica.com

[6] Kay, Michael: Parallel Processing in the Saxon XSLT Processor. 2015. XML Prague 2015. http://archive.xmlprague.cz/2015/files/xmlprague-2015-proceedings.pdf

A. RUIAN Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:pxsl="blinded"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:vf="urn:cz:isvs:ruian:schemas:VymennyFormatTypy:v1"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    exclude-result-prefixes="pxsl">
    <xsl:param name="output-folder" as="xs:string"
            required="yes" />
    <xsl:output indent="yes" />
    <xsl:mode name="stream" streamable="no" />

    <xsl:template match="/vf:VymennyFormat" mode="stream">
        <DocumentListing>
            <xsl:apply-templates select="vf:Data/*"
                    mode="stream" />
        </DocumentListing>
    </xsl:template>
    <xsl:template mode="stream" pxsl:core="yes"
        match="/vf:VymennyFormat/vf:Data/vf:Obce/vf:Obec">
        <xsl:call-template name="doProcess" />
    </xsl:template>
```

```
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:CastiObci/vf:CastObce"
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/
    vf:KatastralniUzemi/vf:KatastralniUzemi">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:Zsj/vf:Zsj">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:Ulice/vf:Ulice">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:Parcely/vf:Parcela">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/
    vf:SpravniObvody/vf:SpravniObvod">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/
    vf:StavebniObjekty/vf:StavebniObjekt">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/
    vf:AdresniMista/vf:AdresniMisto">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:Momc/vf:Momc">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template mode="stream" pxsl:core="yes"
    match="/vf:VymennyFormat/vf:Data/vf:Mop/vf:Obec">
    <xsl:call-template name="doProcess" />
</xsl:template>
<xsl:template name="doProcess">
    <xsl:variable name="data" select="copy-of(.)" />
```

```
        <xsl:variable name="identifier" select="@gml:id" />
        <xsl:variable name="file" as="xs:string"
            select="$output-folder || '/' || local-name(.)
                   || '/' || $identifier || '.xml'"/>
        <file collection="{local-name(.)}"
            ruian-type="{ local-name(.) }" kod="{ $identifier }"
            href="{ $file }" />
        <xsl:result-document href="{ $file }">
            <xsl:sequence select="."/>
        </xsl:result-document>
    </xsl:template>
</xsl:stylesheet>
```

Semantic Hybridization: Mixing RDFa and JSON-LD

R. Alexander Miłowski
School of Information, University of California, Berkeley
`<alex@milowski.com>`

1. Overview

The popularity of JSON-LD [1] and "Web schemas" such as `schema.org` are in part due to the promise of ability of "search giants" to use your information once you've published it onto the Web. While that promise plays into the hope of better utilization of information for dissemination and financial gains by the publisher, there are alternate motivations. One such motivation is the use of information by local consumers (e.g. a Web application or widget) [3].

As a semantic Web format, RDFa has gained some traction but JSON-LD, riding on the "coat tails" of JSON, seems to be gaining more. The recent use of JSON-LD as a format attached to e-mail messages within Google's gmail is a good example [4]. In such a context, JSON-LD is a much more compact mechanism for transporting triples of information.

Further, JSON-LD does not require that every triple be encoded using some aspect of HTML with RDFa [2] attributes. That is, you aren't forced to turn every triple of information into some markup embedded within a document. As such, data that needs to be attached to a document doesn't need to be turned into superfluous markup and this simplifies the encoding for various producers.

That said, this doesn't mean that JSON-LD completely replaces other formats. For example, RDFa is very good at reducing redundancy between triples and information represented within a document as markup and text. RDFa has the ability to encode information in one place by judicious use of annotations via attributes.

As such, the use of JSON-LD for non-replication nicely compliments the use of RDFa information represented within a document. This raises the question as to how these two standards and representations can be used together and what positive and negative affects this may have on the representation of information. This paper describes an approach and attempts to enumerate the possibly synergistic outcomes of using JSON-LD and RDFa together.

2. Hybridization

Let us presume we have a contact person we wish to encode for use within an HTML document. For ease of use, we'll use the author's contact information:

```
Alex Miłowski
alex@milowski.com
http://www.milowski.com/

School of Information
University of California, Berkeley
http://www.ischool.berkeley.edu/
```

We wish to produce the following set of triples regardless of representation:

```
@prefix s: <http://schema.org/>
<http://www.milowski.com/#alex> a s:Person ;
  s:name         "Alex Miłowski";
  s:email        "alex@milowski.com";
  s:url          "http://www.milowski.com/";
  s:organization <_:1> .
<_:1> a        s:Organization ;
  s:name       "University of California, Berkeley" ;
  s:department <_:2> .
<_:2> a s:Organization ;
  s:name "School of Information" ;
  s:url  "http://www.ischool.berkeley.edu/" .
```

2.1. RDFa Only Issues

The information in our use case can be encoded within an HTML document as such:

```
<div>
<p><a href="http://www.milowski.com/">Alex Miłowski</a>;
   <a href="mailto:alex@milowski.com">(personal e-mail)</a></p>
<p><a href="http://www.ischool.berkeley.edu/">School of Information</a>;
   University of California, Berkeley</p>
</div>
```

If we try to encode this information using RDFa without radically changing the markup, we might start as follows:

```
<div vocab="http://schema.org/"
     resource="http://www.milowski.com/#alex" typeof="Person">
<p><a property="url" href="http://www.milowski.com/">
      <span resource="http://www.milowski.com/#alex"
            property="name">Alex Miłowski</span>
   </a>;
   <a property="email" href="mailto:alex@milowski.com"
      content="alex@milowski.com">(personal e-mail)</a>;
</p>
<p property="organization" typeof="Organization">
   <span property="department" typeof="Organization">
      <span property="name">School of Information</span>
      <a property="url" href="http://www.ischool.berkeley.edu/">(website)</a>
   </span>;
   <span property="name">University of California, Berkeley</span>
</p>
</div>
```

Yet, given the markup structure and processing rules of RDFa, some things are not possible or easy:

- The resource URI `http://www.milowski.com/#alex` for the person must be repeated to pickup the name of the person because the subject URI changes to the Website due to how RDFa processes links and changes the subject to the linked resource.
- Removing the `mailto:` from the e-mail address requires duplicating the e-mail address.
- The name of the organization cannot easily be added because the organization does not have an assigned resource URI (it is a blank node).

Fixing these issues requires restructuring and a lot more markup just to make the RDFa properties produce the right set of triples. Such changes in markup representation not only change the design of the presentation but also may cause the publisher confusion. The reasons for the structural changes are due to the RDFa algorithm and assumptions it makes. As such, developers and publishers alike may find such use of RDFa unacceptable.

2.2. JSON-LD Only Issues

The JSON-LD approach is to encode information directly in a JSON compatible syntax that is then embedded within a script element:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type" : "Person",
  "@id"   : "http://www.milowski.com/#alex",
  "name"  : "Alex Miłowski",
  "email" : "alex@milowski.com",
  "url"   : "http://www.milowski.com",
  "organization" : {
     "@type" : "Organization",
     "name"  : "University of California, Berkeley",
     "department" : {
        "@type" : "Organization",
        "name" : "School of Information",
        "url"  : "http://www.ischool.berkeley.edu/"
    }
  }
}
</script>
```

The representation is a straightforward transliteration of the triples into a JSON syntax object. The benefits to the developer and producer are direct as the information can easily be embedded within any document. The information also does not affect the design of other information represented within the document.

Unfortunately, there are two important negative aspects of JSON-LD:

1. The same information represented within the document must be duplicated.
2. The JSON-LD triples have no formal relationship to information elsewhere within the document.

As such, a using application has to make a lot of assumptions to process the JSON-LD information with the intent of affecting its use within the document and consuming Web application.

2.3. Semantic Hybridization

The qualities we wish to retain of both from both RDFa annotation and JSON-LD are:

1. Identifiable content sub-trees that encode resources.
2. Extensible annotations of content without restructuring or copying information.
3. Flexible augmentation of resource annotations using straightforward encodings.
4. Ease of use with minimal intrusion into the act of publishing information.

We do so by following a simple strategy:

1. Resources that are represented by content within the regular content of a document (e.g. displayable HTML content) are identified by an RDFa `resource` attribute which holds the subject URI and a `typeof` attribute that identifies its type.
2. Information in regular content (e.g. HTML elements) is identified by application of RDFa `property` attributes.
3. All other information not within regular content is encoded as JSON-LD using the matching subject URIs of the typed resources in the regular content to augment the triples available from processing the RDFa.

The previous example using these rules is as follows:

```
<script type="application/ld+json">
{
  "@context" : "http://schema.org/",
  "@id" : "http://www.milowski.com/#alex",
  "givenName" : "Raymond",
  "otherName" : "Alexander",
  "familyName" : "Miłowski",
  "email" : "alex@milowski.com",
  "url" : "http://www.milowski.com/",
  "organization" : { "@id": "http://www.berkeley.edu/" }
}
</script>

<p vocab="http://schema.org/">

<span resource="http://www.milowski.com/#alex" typeof="Person">
<span property="name">Alex Miłowski</span>
</span>

works at
<span resource="http://www.berkeley.edu/" typeof="Organization">
<span property="name">University of California, Berkeley</span>'s
<span resource="http://www.ischool.berkeley.edu/" typeof="Organization"
      property="department">
<span property="name">School of Information</span>
</span>
</span>
</p>
```

The triple graph is union of all the triples generated from processing the RDFa annotations and each `script` element with a type of `text/json-ld`. In the example, the connection between the person and the organization is made in the JSON-LD. The RDFa markup derived triples are augmented by the JSON-LD triples.

The result is an application can track the location of RDFa annotations to find locally annotated elements. Scripts can then use the triples to augment the user interface. For example, a receiving application might make the information about the person and organization available on a mouse or touch event.

3. Conclusion

The goal is to promote the durability of information while retaining identifiability, extensibility, and flexibility that is balanced by ease of use. Either RDFa or JSON-LD alone have their limitations. By using Semantic Hybridization, the these representations can be used together and build upon each other using each format for the kinds of information it encodes best. The resulting documents enable encoding information to support polyvalent information resources.

The merged graph retains the locations within the markup that are the origins of the typed subjects. The information provided within the graph by the JSON-LD triples can be used to augment these locations and provide alternate user interfaces or semantic inference. This allows scripting to provide an augmented user experience while the information has a minimal but complete representation.

In the end, we have a document format that utilizes the best representation for the information contained. While a processor has to process a variety of information formats, each syntax is optimal for the producer of the information and this will induce the behavior we want: information is encoded with triples. Such a behavior enhances the durability of the information over the many years it will be retained.

Bibliography

[1] *JSON-LD 1.0: A JSON-based Serialization for Linked Data*, Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Niklas Lindström; http://www.w3.org/TR/json-ld/

[2] *RDFa Core 1.1*, W3C, 2012-06-07, Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman http://www.w3.org/TR/rdfa-core/

[3] *Local Knowledge for In Situ Services*, R. Alexander Miłowski and Henry S. Thompson, *XML Prague 2013*, 2013-02-09 http://archive.xmlprague.cz/2013/files/xmlprague-2013-proceedings.epub

[4] *Email Markup* https://developers.google.com/gmail/markup/getting-started

Using DocBook to Produce a Polyvalent Academic Work

Murray Maloney
Muzmo
Robert J. Glushko
UC Berkeley
R. Alexander Milowski
UC Berkeley

Abstract

Creating many different versions or customizations by configuring a collection of components is a desirable goal in many domains. A single automobile production line can support the assembly of customized variations of a car model. Software product line engineering enables the creation of many similar software systems from a shared set of software assets. In this article we discuss how a collection of content elements can create a family of related texts whose different members are generated according to configurations of variables found in the content markup. This markup is created by the author, but anyone can create a particular edition of the text by defining a configuration file at book-building time, and a reader can do this interactively at reading-time by making selections from a configuration control widget. We call this configurable collection of content elements a "polyvalent" document: "Poly" means "more than one" or "many" - "valent" means "having combining power."

There are some common challenges in all of these domains. The first is to distinguish the components that are contained in every manifestation, typically called the core, base, or platform, from those that vary, typically called the features, options, or supplements. The second is to organize the variable components to indicate the different customizations, versions, or editions that can be built by selectively combining optional components with the required ones. The third challenge is to convey to the builders, users, or others who want to use the variable components any dependencies or constraints that might exist, since not every possible combination will be feasible or sensible.

In this paper, we examine the facility with which DocBook was coerced into supporting a polyvalent text, and the challenges encountered. We observe the parallels and disjunctions among the vocabularies used in book production, the suitability of XHTML and CSS as content delivery agents, the varying *capabilities of current ePub3 readers, and the suitability of relying upon CSS and JavaScript in an ePub context.*

1. Introduction

This article describes the challenges and insights that emerged in the design, development, and delivery of a book titled The Discipline of Organizing ([9]). TDO proposes a transdisciplinary synthesis of ideas from library and information science, computer science, informatics, cognitive science, business, and other disciplines that arrange collections of resources to enable interactions with them.[1]

Organizing is a fundamental issue in many professional fields. However, these fields have only limited agreement in how they approach problems of organizing and in what they seek as their solutions. Nevertheless, despite their obvious differences, the books in libraries, the animals in zoos, weather observations in a data repository, and digital songs on a music player are all "resources" – "things that have value that can support goal-oriented activity" – that have been intentionally selected and organized. Similarly, despite their obvious differences, libraries, zoos, data repositories, and music collections can all be described as "organizing systems" – each is "an intentionally arranged collection of resources and the interactions they support."

A discipline of organizing complements the conventional disciplinary focus on specific resource and collection types (libraries organize books, museums organize art, business systems organize product and customer information) with a framework that views organizing systems as existing in a multi-dimensional design space in which different types of resources can be considered simultaneously, better exposing the relationships and contrasts among them. There are five groups of design decisions, phrased in generic language to emphasize their broad applicability: What is being organized? Why? How much? When? By what means?

A book with the ambitious goal of defining a new discipline must be broad enough to include all the disciplines that contribute to the "transdiscipline" that emerges at their intersection. It must treat each contributing discipline with enough depth so that the new concepts of the emergent discipline can be re-applied meaningfully to discipline-specific concepts and examples.

To make TDO both broad and deep without making it bloated and hard to understand required some innovations in book design and implementation. Our key idea was to tag the book's content by discipline, effectively creating a family of re-

[1]MIT Press published The Discipline of Organizing in print and ebook formats in 2013. The published book names seventeen co-authors, led by the second author of this article, who also edited the book. The first author of this article, one of TDO's six principal co-authors, also served as the markup and production editor. The third author of this article implemented interactivity in the ePub editions. O'Reilly Media published two enhanced ebook editions in August 2014 to take greater advantage of the capabilities of the digital medium and to make it suitable as a textbook for a more diverse set of courses.

lated texts in which varying configurations or subsets of content tailor the book for different courses and perspectives. We are just beginning to explore the complex design space and tradeoffs between author, instructor, and reader contexts and the corresponding user interfaces that are best suited to any particular design.

1.1. Content Structure

Many books, especially technical and professional ones, are designed with a core body of content that is augmented by supplemental content of various types. The types of supplemental content, the structures that organize it in books, and its presentation and formatting are highly conventional (see [11], [14] for historical perspectives, [8], [20] for design guidance).

Tables, figures, illustrations, and sidebars are often supplemental content, and are usually constrained to appear as close as possible to the core text that mentions them. These types of supplemental content are usually created by the author or by people who are following the author's specifications.

Footnotes, endnotes, annotations, bibliographic citations, glossary entries, and indexes are types of supplemental content that are also closely anchored to particular parts of the core text. Footnotes and annotations are usually constrained to appear on the same page as their text anchor, but the other types of content are more typically arranged at the end of larger text units like chapters or at the end of the book. The author does not typically create some of these types of content, especially indexes.

Appendixes, commentaries, reviews, and case studies are types of supplemental content that are typically associated more coarsely with the book as a whole. People other than the book author also commonly create them.

The basic contrast between core and supplemental content is a very old one but the emergence of digital documents has enabled some new variations. Selectable links that "transport" the reader to the linked content or that "transclude" the content into the core text stream were foundational concepts of hypertext proposed by ([17]). The now familiar idea of web browser "plug-ins" or "extensions" for embedding new digital format types into documents was anticipated by ([18]) who developed "multivalent documents" and an extensible reading application in which new layers of content and their specialized interactions and behaviors could be overlaid on the "base" layer. Contemporary examples include the Hypothes.is[2] open annotation platform and the Lens viewer for scientific publications that allow readers to rearrange and focus on different parts of the article.

The goal of interdisciplinary comprehensiveness was undermining the coherence and comprehensibility of the TDO manuscript. At the same time in late 2011 many of the co-authors moved on to other jobs and projects, leaving a much smaller au-

[2] http://hypothes.is

thoring group led by the first two authors of this article to finish the book. This gave us an opportunity to rethink and revise the book from end to end and to attack rather than surrender to the breadth vs. depth challenges.

We decided to restructure the book to emphasize the transdisciplinary core of the new discipline of organizing while preserving the disciplinary identity of the concepts, methods, technology, and people that contributed to it. We did this by editing each chapter to more tightly focus on transdisciplinary content, extracting discipline-specific content into paragraph size chunks collected into a set of endnotes at the end of each chapter.

Some of this restructuring was straightforward because it was simply making explicit the organization of chapters and section. Many followed the "hourglass" or "inverted pyramid" organization typical of news stories and textbooks to begin with an introduction and easily understood or generic examples, followed by additional refining concepts and more specific examples. Often the last paragraph of a section contained the most discipline or industry-specific content, which we then moved into an endnote.

However, most of the restructuring required more thoughtful analysis to determine whether a paragraph should be considered core or supplemental. We considered using text processing tools to calculate term frequency statistics to locate paragraphs that contained concentrations of discipline-specific vocabulary, but these were unlikely to work well given the small size of the text units we sought to restructure. We instead used simpler heuristics that keyed on the occurrence of obscure words or proper nouns like "fonds" or "Sarbanes-Oxley" as indicators of disciplinary-specific paragraphs. Nevertheless, we discovered that we often referred to discipline-specific vocabulary when we proposed design patterns or introduced more abstract terms, so removing their first occurrences from the core text would be a mistake. We sometimes needed to replace pronouns and indirect referrals with more concrete referents to fix continuity problems caused by restructuring.

When TDO went to press in early 2013, we had factored about 24% of the chapter text into six disciplines, which within months were further refined into ten: Library and Information Science, Museums, Archives, Computing, Web, Cognitive Science, Linguistics, Philosophy, Law, and Business.

1.2. Single-source Publishing

Single-source publishing promises the ability to deliver multiple forms of a given work from a single set of source files. For our purposes, that initially meant print-ready and hypertext formats.

By late 2012, the chapters of the book, in simple Word documents, were nearing completion. The front and back matter existed only notionally in the mind of the markup editor. We wanted to publish in both print and ebook formats because we expected it would need frequent revision to stay current. We were dissatisfied with

Word as our source framework, partly because its glossary and indexing tools evaded us, and partly because of our own biases toward non-proprietary, standards-based encoding schemes. We had surveyed XML-based publishing tools and frameworks, and had narrowed our sights on either DocBook or DITA. As a matter of practicality, we chose DITA because Eliot Kimber had agreed to work with us to convert our sources and adapt the standard DITA framework to produce our required output formats. We were actively preparing to produce the work using the DITA framework in Austin, TX in early December 2012. A chance encounter at a publishing conference presented us with a different approach.

We were fortunate to become beta testers for O'Reilly's Atlas single-source publishing environment. Atlas enabled us to deliver print-ready copy, epub, and mobi versions of TDO from the same XML source files, which we marked up using the DocBook schema. DocBook contains chapters, sections, paragraphs, sidebars, lists, figures, tables, links, citations, quotes, glossary and index terms, and other elements needed for books and technical publications. Because it is straightforward to transform text with this rich markup into the other formats and assign corresponding style sheets to them, the first editions were essentially identical except for the layout and formatting flexibility, search, and hyperlinking that are intrinsic to the digital formats.

Atlas was an essential productivity boost, but we resisted the siren call of complete single-sourcing. Rather than take a break during the period that the print book was working its way through manufacturing and distribution, we decided to invest heavily in ebook-targeted content enhancements and semantic markup that were not used in the print version. Even before the print edition of TDO reached bookstores in May 2013 we were well along on an enhanced ebook design that included several dozen photos, embedded quizzes, annotation capability, and other features that took advantage of the digital reading platform.

Atlas continued to meet our requirements for producing a second printing, but DocBook, XHTML5, and ePub had all moved forward and Atlas was not keeping pace with our requirements for greater interactivity. We worked with Bob Stayton to upgrade our sources and customize the DocBook 5 framework to suit our needs. As it turns out, the people at O'Reilly Media had been thinking that DocBook was too complex for non-technical authors and were about to abandon it, so our decision turned out to be prescient in retrospect. Now that the dust is beginning to settle on their new strategy, relying upon a simpler HTML5-based source model, we find ourselves tempted again by the siren call of single-source publishing, we are tempted to consider transforming our DocBook XML to XHTML5 to rejoin the Atlas tool chain, but only if we can do it while preserving the rich semantics we have encoded in the former. (We will propose some ways we might do this in Section 3.4.)

1.3. Selective Inclusion by Discipline

Readers are familiar with the contrast between core and supplemental content and somehow decide how much of the latter to read when they encounter it in a book. However, the distinction is generally not based on disciplinary specificity, so we could not predict how it would affect the TDO reading experience.

TDO readers understood that the disciplinary labels on each endnote could help them decide whether to read it or not. We realized too late for the first print edition of TDO that we could append discipline labels to the superscripts marking the note in the core text, but we were able to do so in the ebook editions. This made it even easier for readers to be selective about supplemental content because it made it unnecessary to flip to the end of the chapter to determine the disciplinary focus of the endnote.

We informally surveyed students in courses that used TDO about how they read the book. Some students with the print edition read everything in one pass, including the endnotes by flipping back and forth to the end of the chapter. Other students using the print edition read the core text first, and then read all the notes in a second pass. Some students reported that they ignored notes that were in unfamiliar disciplines, while others said they paid more attention to disciplines that were unfamiliar.

Tracking reading behavior is technically straightforward in digital reading environments but the most common ebook readers from Apple and Amazon are proprietary and closed, and neither firm shares user experience data with authors or publishers. Software that can unzip ebooks and render them in ordinary web browsers can exploit web analytic mechanisms for tracking user events ([4]), but will students let their instructors monitor how or if they read an assigned textbook?

A more systematic survey was conducted at Berkeley in the fall semester 2014, where all students used TDO in ebook formats in a graduate course taught by the second author of this article. This survey revealed greater likelihood to read supplemental content in ebooks than with print versions, but also confirmed that both the propensity to read and disciplinary preferences for supplemental reading were highly variable. These findings gave us more motivation to find a reading-time customization approach, which we describe in Section 2.2.

1.4. Inclusion vs. Transclusion vs. Exclusion

We had initially hoped to employ transclusion as the mechanism for incorporating supplemental content; readers would be alerted to its presence with a disciplinary-specific symbol in the page margin, and selecting the symbol would seamlessly insert the content into the core text stream, perhaps subtly altering its text formatting or font to remind the reader of its supplemental role.

However, transclusion isn't supported in any existing book reader, so we were forced to rely on more traditional inclusion mechanisms of link following (and return) and pop-up notes. We preferred the latter because it better maintains the reading context, but its poor implementation forced us to rely on the former. In particular, Apple's popular iBooks reader supports pop-ups (see Figure 1), but doesn't allow link traversal from a pop-up note. Since most TDO endnotes contain citations, pop-up notes would become dead ends. We aspire to the happy compromise that presents itself in the Lucifox plug-in for Firefox (see Figure 2).

We now realize that there is a third class of design mechanisms that we need to explore. Framing our content architecture in terms of core and supplemental content assumes that readers are selectively incorporating additional content to a book. It is interesting to consider starting with the complete book and enabling readers to selectively exclude rather than include content. One possibility would be to invert the user experience we imagined for transclusion, leaving a margin symbol to indicate where the reader has chosen to exclude content. Or we might use the familiar presentation conventions of reducing the font size or graying out for content that is excluded by the filter applied by the reader.

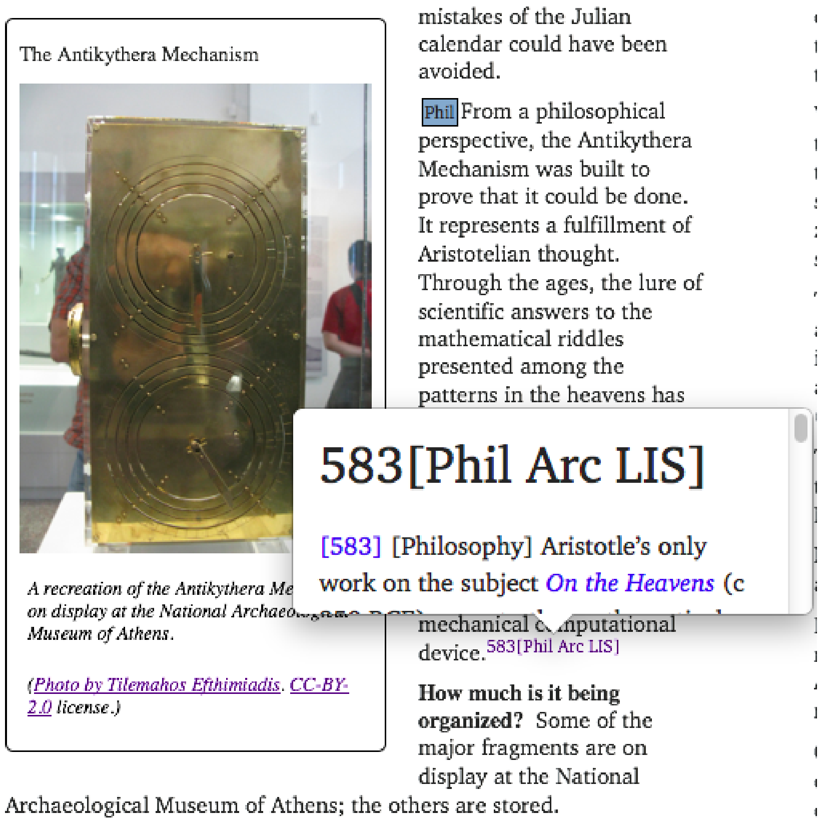
1.5. A Family of Related Books

In its first year, TDO was adopted in whole or in part by about 20 schools for a variety of courses in information organization, content management, collection development, and information architecture. However, many instructors were using only parts of TDO and asked for a simpler shorter version more suitable for undergraduate courses, which meant we needed to further refine our classification of the book's content.

We extended the idea of disciplinary labeling of content to identify an additional 15% of the chapter content as being focused on disciplinary-specific rather than transdisciplinary content. This was easier this time because it was easier to evaluate consistency and continuity when it wasn't necessary to flip back and forth between the chapter body and its endnotes to consider whether to remove a paragraph from the core.

During this time we also invested in a substantial amount of markup to enhance the amount of processable semantics represented in the source text. In addition to its structural elements, the DocBook schema contains semantic elements which we used to identify people, organizations, locations, products, applications, abbreviations, foreign phrases, and other potentially useful semantic "nuggets" that were mixed into the text. We invested in this semantic markup because we imagined being able to interconnect digital versions of TDO with other semantically described web resources by exposing it according to the conventions for "linked data" ([3], [1]) even if the current technology for ebooks was incapable of enabling it.

A more speculative form of semantic markup we explored was to label content according to its rhetorical purpose and intended audience. We classified some phrases as definitions, principles, statements, examples, parenthetical cross-refer-



The Antikythera Mechanism

A recreation of the Antikythera Me
on display at the National Archaeo
Museum of Athens.

(Photo by Tilemahos Efthimiadis. CC-BY-2.0 license.)

mistakes of the Julian calendar could have been avoided.

Phil From a philosophical perspective, the Antikythera Mechanism was built to prove that it could be done. It represents a fulfillment of Aristotelian thought. Through the ages, the lure of scientific answers to the mathematical riddles presented among the patterns in the heavens has

583[Phil Arc LIS]

[583] [Philosophy] Aristotle's only work on the subject *On the Heavens* (c

mechanical c mputational device.583[Phil Arc LIS]

How much is it being organized? Some of the major fragments are on display at the National Archaeological Museum of Athens; the others are stored.

The iBooks endnotes. A build-time option injects ePub3 semantics enabling an iBooks footnote interaction that presents the pop-up viewport. Sadly, it exhibits odd behavior with links. Exercising an external link presents the resource in the same cramped viewport. Exercising an internal link yields an empty viewport.

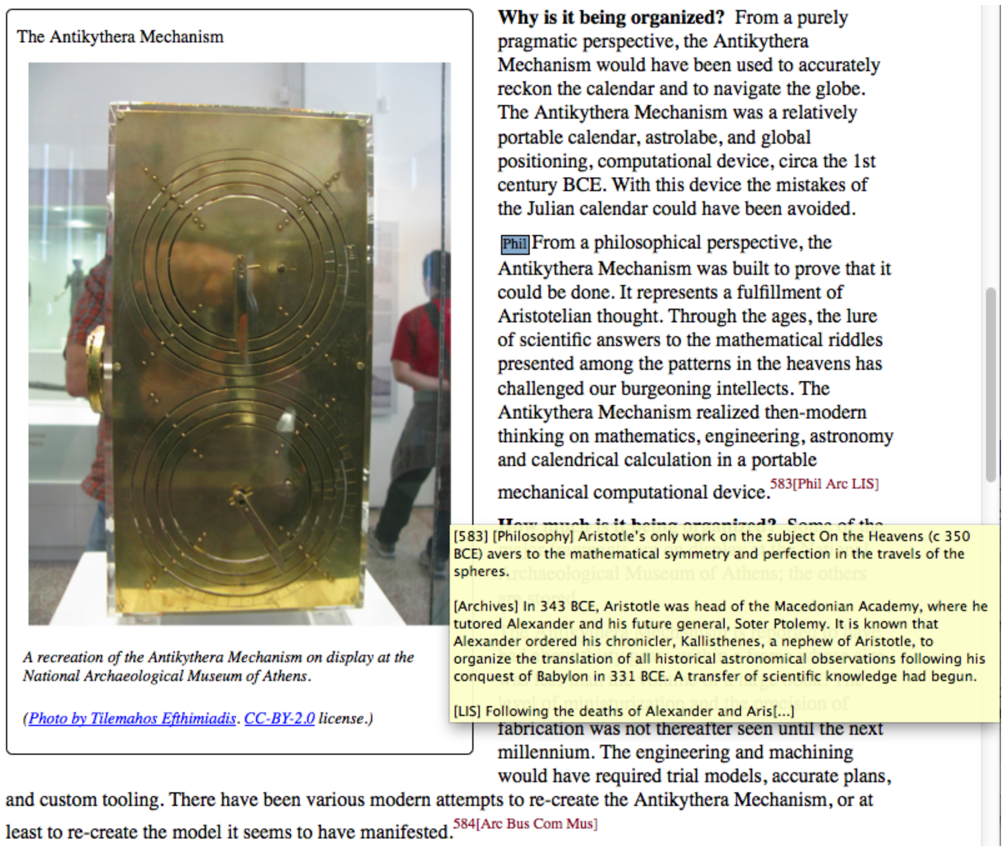
Figure 1. iBooks Popup

ences, and editorial asides; we classified some content as suitable for undergraduate, graduate, or professional audiences. We consulted some work in rhetoric, critical theory, and computational linguistics to create the former classification, but the categories and their boundaries we discovered in this diverse literature are not entirely consistent (see [2]).

We are currently exploring some different approaches for creating custom editions based on this markup. We note that there are considerable similarities between this goal of creating a "textbook family" and that of creating a "software product family" (see [12]; [13]). Furthermore, the techniques we propose have analogues with conditional compilation ([5]) and visualizations that contrast core and supplemental content ([6]; [7]).

2. A Polyvalent Academic Text

When a person or a thing is “polyvalent,” it presents many different functions, forms, or facets; it is adaptable; like the “Jack of all trades”. A polyvalent text is one



The Antikythera Mechanism

A recreation of the Antikythera Mechanism on display at the National Archaeological Museum of Athens.

(Photo by Tilemahos Efthimiadis. CC-BY-2.0 license.)

Why is it being organized? From a purely pragmatic perspective, the Antikythera Mechanism would have been used to accurately reckon the calendar and to navigate the globe. The Antikythera Mechanism was a relatively portable calendar, astrolabe, and global positioning, computational device, circa the 1st century BCE. With this device the mistakes of the Julian calendar could have been avoided.

Phil From a philosophical perspective, the Antikythera Mechanism was built to prove that it could be done. It represents a fulfillment of Aristotelian thought. Through the ages, the lure of scientific answers to the mathematical riddles presented among the patterns in the heavens has challenged our burgeoning intellects. The Antikythera Mechanism realized then-modern thinking on mathematics, engineering, astronomy and calendrical calculation in a portable mechanical computational device.583[Phil Arc LIS]

How much is it being organized? Some of the

[583] [Philosophy] Aristotle's only work on the subject On the Heavens (c 350 BCE) avers to the mathematical symmetry and perfection in the travels of the spheres.

[Archives] In 343 BCE, Aristotle was head of the Macedonian Academy, where he tutored Alexander and his future general, Soter Ptolemy. It is known that Alexander ordered his chronicler, Kallisthenes, a nephew of Aristotle, to organize the translation of all historical astronomical observations following his conquest of Babylon in 331 BCE. A transfer of scientific knowledge had begun.

[LIS] Following the deaths of Alexander and Aris[...]

fabrication was not thereafter seen until the next millennium. The engineering and machining would have required trial models, accurate plans, and custom tooling. There have been various modern attempts to re-create the Antikythera Mechanism, or at least to re-create the model it seems to have manifested.584[Arc Bus Com Mus]

The Lucifox add-on for Firefox offers hypertext cues. When hovering over any link that is local to ePub3, this feature presents a tooltip with an excerpt from the target of the link and invites the reader to follow the link. This feature works with footnotes, cross-references, glossary terms, citations, and bibliography, and even for the referents of index terms

Figure 2. Hypertext cues in Lucifox

that is transformative and can adapt to the information needs of the reader. Such text often exhibits a dynamic quality where the user may express their information needs via implicit or explicit methods and the flow and contents of the text changes to represent the subset of information contained that matches their needs. Considered as a whole, the polyvalent text is a superset of all the information needs of its various consumers.

The text is written with multiple disciplines, or audiences, in mind, including footnotes and paragraphs identified as LIS, Museums, Archives, Computer Science, Linguistics, Cognitive Science, Business, Law, and so on.

Example 1. Paragraphs and Footnotes by Discipline

```
<para audience="CORE IA"><phrase role="statement">Classification
makes systems more usable when it is manifested in the arrangement
of resource descriptions  or controls in user interface components
like list boxes, tabs, buttons, function menus, and structured
```

```
lists of search results.</phrase
 ><footnote xml:id="endnote-394" label="394" audience="IA">
    <para audience="IA">[IA] The application of classification and
      organizing principles more generally to the design of
      user interfaces to facilitate information access, navigation,
      and use is often called <quote>Information Architecture.</quote>
      See <citation xml:id="cite_Morville2006-7.1"
            linkend="Morville2006">(Morville and Rosenfeld 2006)</citation>.
    </para></footnote></para>
```

Example 2. Bibliography by Discipline

When a bibliography entry is cited only in one disciplinary endnote, and is not mentioned in the surface text, we may choose whether to assign that entry only to the discipline specified by its parent endnote, or rather to also mention other disciplines so that the entry can appear in other editions, even though the corresponding citation and endnotes may not also appear. Here, we have an entry that will only appear in editions in which information architecture, computing, or the web are selected:

```
<biblioentry xml:id="Morville2006" audience="IA Computing Web">
  <authorgroup>
    <author><personname><firstname>Peter</firstname>
                        <surname>Morville</surname></personname></author>
    <author><personname><firstname>Louis</firstname>
                        <surname>Rosenfeld</surname></personname></author>
  </authorgroup>
  <pubdate>2006</pubdate>
  <title>Information Architecture for the World Wide Web</citetitle>
  <address>Sebastopol, CA</address>
  <publisher<publishername>O'Reilly</publishername></publisher></biblioentry>
```

Some structural elements of the content are identified by user levels, including Professional, Instructor, Graduate, and so on. These elements are used for the edition-specific content, such as the Stop and Think exercises in the Core Concepts Edition.

Example 3. Userlevel selection

```
<sidebar userlevel="Professor Instructor Graduate Undergraduate"
         xml:id="StopAndThink-2.4.2.1-search-engines">
  <title>Stop and Think: Browsing for Books</title>
  <para>How does the experience of browsing for books in a library
    or bookstore compare with browsing using a search engine?
    What aspects are the same or analogous in all these contexts?
    What aspects are different?</para></sidebar>
```

Some structural elements are identified by vocation, including Archivist, Curator, Linguist, and so on. These are used for edition-specific content, such as the cover, subtitle, ISBN, and so on.

Example 4. Edition-specific

```
<subtitle userlevel="Professional"
          audience="LIS Archives Museums
                    Computing IA Web Law Business
                    CogSci Linguistics Philosophy"
         >Professional Edition</subtitle>

<subtitle userlevel="Undergraduate"
          audience="CORE"
         >Core Concepts Edition</subtitle>

<subtitle userlevel="Librarian Archivist Curator"
          audience="LIS Archives Museums"
          >Academic Edition: Memory Institutions</subtitle>

<subtitle userlevel="Philosopher Linguist"
          audience="CogSci Linguistics Philosophy"
         >Academic Edition: Sensemaking</subtitle>
```

2.1. Static Multivalent Editions

We modified the build process we used to produce the initial ebooks to produce an ebook that contains whatever set of disciplines is specified in a configuration file. An XSLT script filters content based on the values assigned to AUDIENCE and USERLEVEL attributes when we assemble books from our DocBook XML source files.

This approach turns the source files for a book into a family of books with a common core extended with discipline-specific content. With many disciplines the combinatorial possibilities make this an extremely large family (with eleven discipline types, there are 2048 distinct combinations of zero to eleven), and even if we apply strong reasonableness or familiarity constraints it is still easy to imagine many subset configurations of disciplines that would generate appealing custom textbooks:

- Academic Edition: Memory Institutions (LIS, Museums, Archives)
- Academic Edition: Sensemaking (Cognitive Science, Linguistics, Philosophy)
- Academic Edition: Informatics (Computing, Information Architecture, Web, Business, Law)
- Academic Edition: Information Architecture (Information Architecture, Linguistics, Web)

However, for business and marketing reasons we decided, with our publisher, to go to market with just two combinations that define the endpoints of possible disciplinary customization:

- Professional Edition (all disciplinary endnotes)
- Core Concepts Edition (no disciplinary endnotes, added Stop and Think exercises)

We expect that as more schools adopt the book for a wider range of courses and student populations we will gain experience that might cause us to market other combinations.

Static editions are customized for a particular set of audience and user-level parameters by filtering content, based on profiles of DocBook 5 attribute values. Any given content element may be a member of one or more user levels and audiences. Membership in the set is primarily dependent on any one of the @audience values being selected.

For example, two paragraphs might be tagged as @audience="Computing IA Linguistics" and @audience="Computing Business Law". Those paragraphs will be members of the set if any of those audience values are selected in the build configuration. So long as "Computing" is selected, both paragraphs will be included.

Similarly, selection of one or more user levels will restrict the set accordingly.

Example 5. Configuration Excerpt

```
<group xml:id="Professional"
       name="Professional Edition"
       ref=" professional
             core lis archives museums
             computing ia web law bus
             cogsci ling phil"/>

<group xml:id="Undergraduate"
       name="Core Concepts Edition"
       ref=" undergrad core"/>
```

Thus, we were able to produce a static "Professional Edition" alongside a static "Core Concepts Edition" for undergraduates. These editions represent the two ends of the audience spectrum: all footnotes and no footnotes. We are currently testing the production of more nuanced editions for "Memory Institutions," "Informatics," and "Sensemaking"

Example 6. Academic Editions

```
<group xml:id="Memory"
       name="Academic Edition: Memory Institutions"
       ref=" graduate librarian archivist curator
             core lis museums archives"/>
```

```
<group xml:id="Informatics"
       name="Academic Edition: Informatics"
       ref=" graduate programmer
             core computing ia web bus law "/>

<group xml:id="Sensemaking"
       name="Academic Edition: Sensemaking"
       ref=" graduate philosopher linguist psychologist
             core cogsci ling phil"/>
```

The "Instructor Edition" will include supplemental material, such as in-class exercises, essay assignments, and typical examination questions. Because userlevel and discipline are orthogonal axes of selection, we will be able to create static editions that correspond to the Academic Editions, as well as dynamic polyvalent editions.

Example 7. Instructor Edition

```
<group xml:id="Instructor"
       name="Instructor Edition"
       ref=" instructor
             core lis archives museums
             computing ia web law bus
             cogsci ling phil"/>
```

As a tool for development, two special editions were designed for editorial and production purposes. The *Editor's Edition* serves as a staging ground for new material that is not quite ready for inclusion in the main corpus, and includes notes left by the editors to remind each other about needed re-writes, citations, glossary entries, and so on. The *Markup Edition* of the work includes an extra chapter, *Production Notes*, written during development and used to test DocBook markup, transformed into XHTML, and manifested in print and ePub results, using CSS to control the presentational and behavioral characteristics.

Example 8. Editor and Markup Editions

```
<group xml:id="Editor"
       name="Editor's Edition"
       ref=" editor
             core lis archives museums
             markup production publishing
             computing ia web law bus
             cogsci ling phil"/>

<group xml:id="Markup"
       name="Markup Edition"
```

```
    ref=" marker producer publisher
          core lis archives museums
          markup production publishing
          computing ia web law bus
          cogsci ling phil"/>
```

2.2. Dynamic Polyvalent Editions

A second approach to customization is motivated by the reality that not all students in a particular course have the same disciplinary backgrounds and interests, and not all parts of a book require or permit the same pattern of disciplinary coverage. In TDO, for example, Chapter 3 on "Resources" discusses philosophical topics about ontology and identity like "carving nature at its joints" and "the ship of Theseus", in contrast with Chapter 8 on "The Forms of Resource Description" that has much more need to focus on technical architecture and implementation concerns. So it would be desirable for disciplinary customization to be determined by the reader in response to his preferences given a particular type of content.

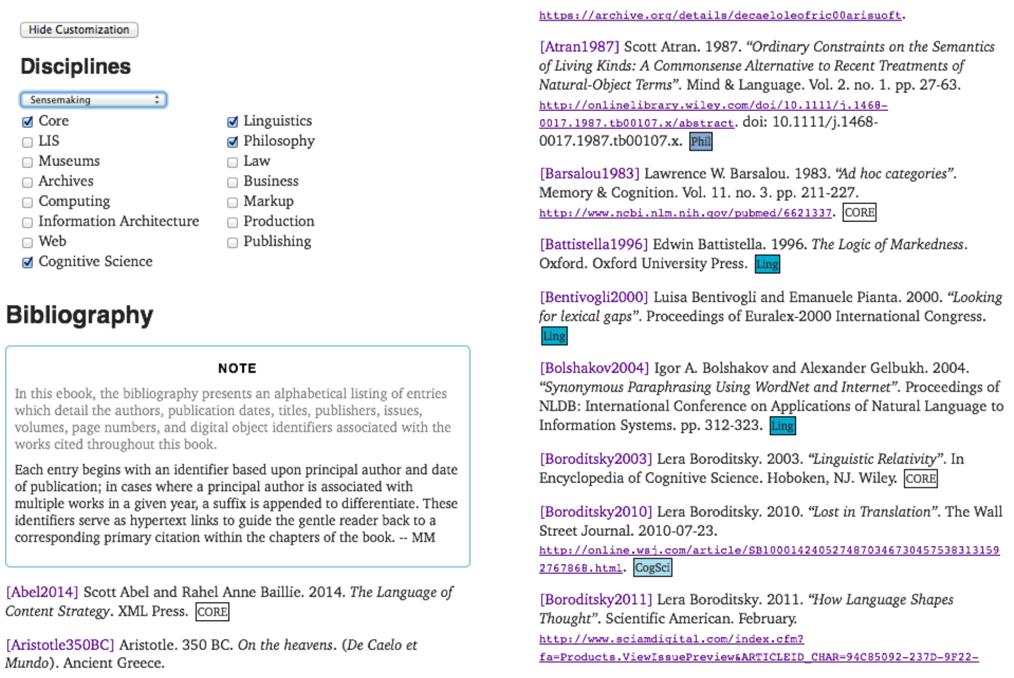
To enable "reading time" or dynamic customization, we modified the build process to convert the discipline AUDIENCE attribute into a CLASS attribute in the generated XHTML. The third author of this paper implemented a prototype interface (see Figure 3 and Figure 4) that uses CSS and JavaScript to insert a list of check boxes before each section of the book; the reader can select which disciplines to include and exclude; the ebook dynamically re-formats accordingly by modifying the CSS display property of the affected elements. For convenience, groupings of discipline types that correspond to a named edition are readily chosen from a pull-down menu; this approach allows a single product to dynamically morph itself into any of the family members. However, its reliance on JavaScript limits its deployment to a select few ebook readers.

Although this prototype script is sufficient for trivial demonstration, we have had to limit its capability and utility for lack of local storage. Ideally, we would offer a facility to set and store user preferences to adjust the presentation of semantically-significant elements, under different reading scenarios. A student reading a book for the first time may want it all, while a student preparing for an examination may prefer to see only content that relates to their field.

Example 9. Excerpt from tdo-toggle.js

```
TDOToggle.prototype.update = function() {
   var css = "";
   for (var key in this.checkboxes) {
      if (!this.checkboxes[key].checked) {
         css += "."+this.checkboxes[key].value+" { display: none; }\n";
      }
   }
```

```
        this.styleElement.innerHTML = css;
    }
```



Hide Customization

Disciplines

Sensemaking

- ✓ Core
- ☐ LIS
- ☐ Museums
- ☐ Archives
- ☐ Computing
- ☐ Information Architecture
- ☐ Web
- ✓ Cognitive Science
- ✓ Linguistics
- ✓ Philosophy
- ☐ Law
- ☐ Business
- ☐ Markup
- ☐ Production
- ☐ Publishing

Bibliography

NOTE

In this ebook, the bibliography presents an alphabetical listing of entries which detail the authors, publication dates, titles, publishers, issues, volumes, page numbers, and digital object identifiers associated with the works cited throughout this book.

Each entry begins with an identifier based upon principal author and date of publication; in cases where a principal author is associated with multiple works in a given year, a suffix is appended to differentiate. These identifiers serve as hypertext links to guide the gentle reader back to a corresponding primary citation within the chapters of the book. -- MM

[Abel2014] Scott Abel and Rahel Anne Baillie. 2014. *The Language of Content Strategy*. XML Press. CORE

[Aristotle350BC] Aristotle. 350 BC. *On the heavens*. (*De Caelo et Mundo*). Ancient Greece.

https://archive.org/details/decaeloleofric00arisuoft.

[Atran1987] Scott Atran. 1987. *"Ordinary Constraints on the Semantics of Living Kinds: A Commonsense Alternative to Recent Treatments of Natural-Object Terms"*. Mind & Language. Vol. 2. no. 1. pp. 27-63. http://onlinelibrary.wiley.com/doi/10.1111/j.1468-0017.1987.tb00107.x/abstract. doi: 10.1111/j.1468-0017.1987.tb00107.x. Phil

[Barsalou1983] Lawrence W. Barsalou. 1983. *"Ad hoc categories"*. Memory & Cognition. Vol. 11. no. 3. pp. 211-227. http://www.ncbi.nlm.nih.gov/pubmed/6621337. CORE

[Battistella1996] Edwin Battistella. 1996. *The Logic of Markedness*. Oxford. Oxford University Press. Ling

[Bentivogli2000] Luisa Bentivogli and Emanuele Pianta. 2000. *"Looking for lexical gaps"*. Proceedings of Euralex-2000 International Congress. Ling

[Bolshakov2004] Igor A. Bolshakov and Alexander Gelbukh. 2004. *"Synonymous Paraphrasing Using WordNet and Internet"*. Proceedings of NLDB: International Conference on Applications of Natural Language to Information Systems. pp. 312-323. Ling

[Boroditsky2003] Lera Boroditsky. 2003. *"Linguistic Relativity"*. In Encyclopedia of Cognitive Science. Hoboken, NJ. Wiley. CORE

[Boroditsky2010] Lera Boroditsky. 2010. *"Lost in Translation"*. The Wall Street Journal. 2010-07-23. http://online.wsj.com/article/SB10001424052748703467304575383131592767868.html. CogSci

[Boroditsky2011] Lera Boroditsky. 2011. *"How Language Shapes Thought"*. Scientific American. February. http://www.sciamdigital.com/index.cfm?fa=Products.ViewIssuePreview&ARTICLEID_CHAR=94C85092-237D-9F22-

Screenshot of Bibliography with Customize script activated and Sensemaking Edition selected.

Figure 3. Sensemaking Edition

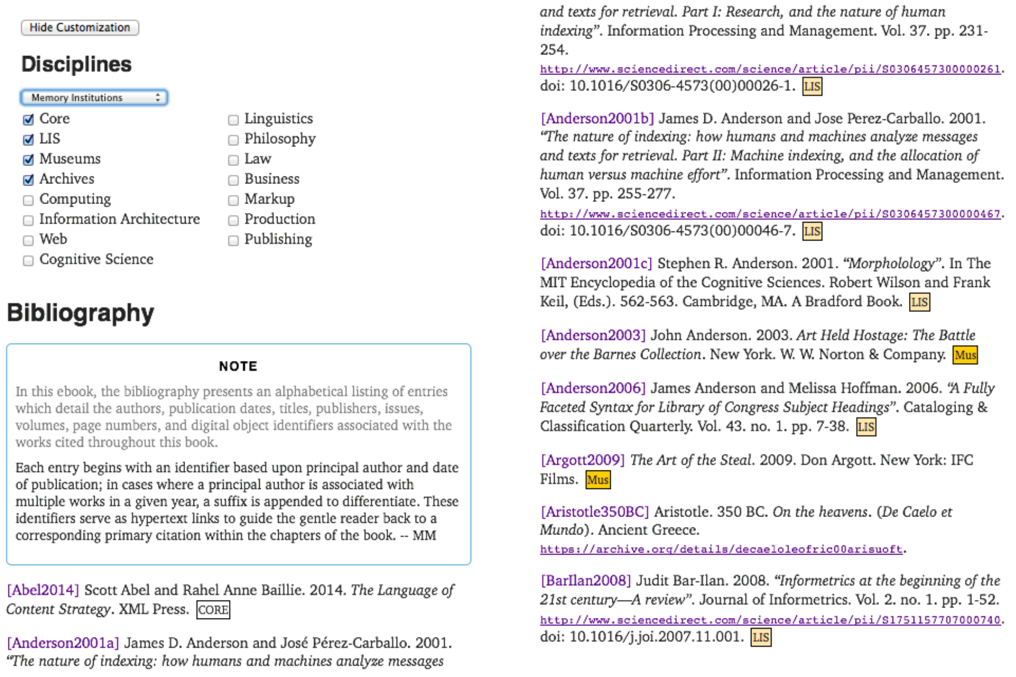
In the second of the Bibliography screenshots, we can see that a different set of disciplines are selected and the bibliography entries correspond to the new selection. The only entries in common are [Abel2014] and [Aristotle350BC]; the first being a CORE entry and the second being unattributed to a specific discipline. The decision to not attribute [Aristotle350BC] to a specific discipline was an editorial one, purposely including the entry in every bibliography, in spite of the fact that the citation is contained within a discipline-specific endnote paragraph.

3. Reflections and Directions

Wherein we reflect upon our decisions and explore our future directions.

3.1. Core vs. Supplemental

We restructured TDO into core and supplemental content relatively late in the authoring process as a response to multidisciplinary bloat, and it was often necessary to rearrange and revise paragraphs to preserve syntactic and conceptual continuity.



Disciplines

Memory Institutions

- [✓] Core
- [✓] LIS
- [✓] Museums
- [✓] Archives
- [ ] Computing
- [ ] Information Architecture
- [ ] Web
- [ ] Cognitive Science
- [ ] Linguistics
- [ ] Philosophy
- [ ] Law
- [ ] Business
- [ ] Markup
- [ ] Production
- [ ] Publishing

Bibliography

NOTE

In this ebook, the bibliography presents an alphabetical listing of entries which detail the authors, publication dates, titles, publishers, issues, volumes, page numbers, and digital object identifiers associated with the works cited throughout this book.

Each entry begins with an identifier based upon principal author and date of publication; in cases where a principal author is associated with multiple works in a given year, a suffix is appended to differentiate. These identifiers serve as hypertext links to guide the gentle reader back to a corresponding primary citation within the chapters of the book. -- MM

[Abel2014] Scott Abel and Rahel Anne Baillie. 2014. *The Language of Content Strategy*. XML Press. CORE

[Anderson2001a] James D. Anderson and José Pérez-Carballo. 2001. *"The nature of indexing: how humans and machines analyze messages and texts for retrieval. Part I: Research, and the nature of human indexing"*. Information Processing and Management. Vol. 37. pp. 231-254. http://www.sciencedirect.com/science/article/pii/S0306457300000261. doi: 10.1016/S0306-4573(00)00026-1. LIS

[Anderson2001b] James D. Anderson and Jose Perez-Carballo. 2001. *"The nature of indexing: how humans and machines analyze messages and texts for retrieval. Part II: Machine indexing, and the allocation of human versus machine effort"*. Information Processing and Management. Vol. 37. pp. 255-277. http://www.sciencedirect.com/science/article/pii/S0306457300000467. doi: 10.1016/S0306-4573(00)00046-7. LIS

[Anderson2001c] Stephen R. Anderson. 2001. *"Morphololology"*. In The MIT Encyclopedia of the Cognitive Sciences. Robert Wilson and Frank Keil, (Eds.). 562-563. Cambridge, MA. A Bradford Book. LIS

[Anderson2003] John Anderson. 2003. *Art Held Hostage: The Battle over the Barnes Collection*. New York. W. W. Norton & Company. Mus

[Anderson2006] James Anderson and Melissa Hoffman. 2006. *"A Fully Faceted Syntax for Library of Congress Subject Headings"*. Cataloging & Classification Quarterly. Vol. 43. no. 1. pp. 7-38. LIS

[Argott2009] *The Art of the Steal*. 2009. Don Argott. New York: IFC Films. Mus

[Aristotle350BC] Aristotle. 350 BC. *On the heavens*. (*De Caelo et Mundo*). Ancient Greece. https://archive.org/details/decaeloleofric00arisuoft.

[BarIlan2008] Judit Bar-Ilan. 2008. *"Informetrics at the beginning of the 21st century—A review"*. Journal of Informetrics. Vol. 2. no. 1. pp. 1-52. http://www.sciencedirect.com/science/article/pii/S1751157707000740. doi: 10.1016/j.joi.2007.11.001. LIS

Screenshot of Bibliography with Customize script activated and Memory Institutions Edition selected.

Figure 4. Memory Institutions Edition

For example, endnotes must be anchored at the end of sentences, most often at the end of paragraphs, and generally at the end of a section.

We concluded that it would have been much easier to write a book with this core + supplement architecture if we had started with this architecture in mind. This of course is conventional wisdom in software engineering; re-factoring is harder than building in modularity from the outset on a more generic platform designed to be extended with plug-in components. Nevertheless, several TDO co-authors and instructors have proposed to add additional categories of discipline-specific content to make the book a better fit to their courses and expertise, and we were able to add the eleventh disciplinary category for "Information Architecture" relatively late in the process of creating the 2014 editions. Ideally, we can identify "discipline editors" who are responsible for their evolution, adding new endnotes, sidebars, or other supplemental content as required.

Example 10. Supplemental Content

```
<sidebar audience="IA">
  <title>Information Architecture</title>
  <para>...<footnote audience="IA">
     <para>...</para> </footnote> </para>
</sidebar>
```

3.2. User Interaction

The ePub3 platform offers many features to enable the reader to interact with an ebook and often offers the reader mechanisms by which the interface can be modified to suit the individual reader's preferences, although much depends upon the platform through which the reader interacts with the ebook. The reader typically has access to a table of contents and a search interface, and often has control over some facets of page layout. Many ePub platforms also offer helpful features that may fail to account for some sophisticated document representations.

Search facilities within appropriately encoded ePub context can and should be more context sensitive. For starters, search results that distinguish among first use, definitions, mentions in keyword metadata, mentions in text and titles, mentions in a bibliography or glossary entries, and mentions in an index.

A table of contents is typically represented as a series of nesting lists with titles and references. In print, such a list might include the titles of chapters, sections, and even sub-sections. In electronic media there is no reason to limit the level of detail that we provide in a table of contents, but neither is it necessary to always present every level of the hierarchy. The ability to control presentation of the table of contents would be especially welcome.

DocBook and ePub3 each include the concept of a bibliography, glossary, and index; XHTML and CSS do not. This disjunction between the structural and elemental components of the logical work and those of the delivered product impedes the efforts of the content creator to reliably deliver utility to the consumer. We hope to discover an appropriate set of publishing semantics to allow us to encode the structure of academic works to enable discovery and interaction with the book, as a book, and imbue its logical sub-components with familiar affordances.

3.3. XHTML5, CSS and JavaScript

We observe that there exist both parallels and disjunctions among the various book production models and vocabularies employed by XHTML5, CSS, and JavaScript in an ePub context. Mismatches among these have thwarted our attempts to produce dictionary-style headings for the bibliography, glossary, and index.

Popular ePub3 reader platforms offer nominal author- or user-control over presentation of paged content or text styling. We observe inconsistent application of XHTML and CSS features across popular reader platforms. For example, building collapsible list structures for tables of contents, bibliographies, glossary, and index could be achieved with universal support for the xhtml5:detail element; better yet would be providing those semantics through CSS.

Example 11. CSS Precedence

Polyvalent style sheets are a challenge because of the CSS precedence rules. Given that any element may be attributed to one or more disciplines, the order of stylesheet rules affects which actions are fired. A paragraph that is attributed to IA and Web may be labeled as one or the other, depending on the order in which they appear in the stylesheet. In this example, Web wins because it is the latest, or most recent declaration.

```
p.IA::before { content: "IA"; ...}

p.Web::before { content: "Web"; ...}
```

We have observed that we cannot rely on the availability of JavaScript on all reader platforms. We expected that portable reader devices might not have JavaScript capability; we were disappointed that the Lucifox add-on for Firefox disables embedded JavaScript.

3.4. Semantic Enablement

Semantic enablement is a process that relies upon the content creator to annotate their material with actionable semantic labels, and upon a downstream processor to leverage those semantics for presentation to and interaction with people. The ePub3 platform, for example, mandates that all book archives include a table of contents that is subsequently used by the reader platform to provide a hierarchical list of hypertext links.

We have had to limit our exploitation of semantic enablement while waiting upon new versions of ePub3 and DocBook5. In the interim we have relied upon DocBook's @role and XHTML's @class and @rel to signal semantics downstream. We are now considering the application of RDFa to DocBook 5.1 sources to produce rich XHTML documents containing hybrid meta-data stores leveraging native XHTML5 semantics, supplemented with a surface layer of RDFa, and provisioned with compact JSON triple stores to support JavaScript interactions.

3.5. Distributed Authoring and Publishing

Our ultimate goal is to implement a distributed authoring and publishing system in which new content can be dynamically discovered and logically included in the family of related books. We are exploring the application of DocBook5 annotation and advanced hypertext features to enable two-way linking between TDO and content created by and for the schools in which the book is being used. We are also re-considering Atlas as single-source repository for delivery to multiple media because of the obvious benefits of relying on a generic industrial-strength platform rather than maintaining our own customized one.

4. Acknowledgments

We are grateful for technical assistance provided by Eliot Kimber, Bob Stayton, Liam Quin, George Kerscher, Häkon Lie, Adam Witwer, Nellie McKesson, Fred Chasen, and the oXygen technical support team..

Bibliography

[1] Allemang, D., & Hendler, J. (2011). Semantic web for the working ontologist: effective modeling in RDFS and OWL. Elsevier.

[2] Bitzer, L. F. (1992). The rhetorical situation. Philosophy & rhetoric, 1-14.

[3] Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data-the story so far. International journal on semantic web and information systems, 5(3), 1-22.

[4] Chasen, F., Harnell, J., and Renold, A.J. 2014. Future Press. Unpublished Master's Project, UC Berkeley School of Information.

[5] Couto, M. V., Valente, M. T., & Figueiredo, E. (2011, March). Extracting software product lines: A case study using conditional compilation. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on (pp. 191-200). IEEE.

[6] Feigenspan, J., Papendieck, P., Kästner, C., Frisch, M., & Dachselt, R. "FeatureCommander: colorful# ifdef world." In Proceedings of the 15th International Software Product Line Conference, Volume 2, p. 48. ACM, 2011.

[7] Feigenspan, J., Kästner, C., Apel, S., Liebig, J., Schulze, M., Dachselt, R., Papendieck, M., Leich, T., & Saake, G. "Do background colors improve program comprehension in the# ifdef hell?." Empirical Software Engineering 18, no. 4 (2013): 699-745.

[8] Glushko, R. J., & McGrath, T. (2005). Document engineering. MIT Press.

[9] Glushko, R. J. (Editor) (2013). The Discipline of Organizing. MIT Press.

[10] Glushko, R. J. (Editor) (2014). The Discipline of Organizing. O'Reilly Media.

[11] Grafton, A. (1999). The footnote: A curious history. Harvard University Press.

[12] Kästner, C., Apel, S., & Kuhlemann, M. (2008, May). Granularity in software product lines. In Proceedings of the 30th international conference on Software engineering (pp. 311-320). ACM.

[13] Krueger, C. (2002). Easing the transition to software mass customization. In Software Product-Family Engineering (pp. 282-293). Springer Berlin Heidelberg.

[14] Kilgour, F. G. (1998). The evolution of the book. New York and Oxford: Oxford University Press.

[15] Klein, J. T. (2010). A taxonomy of interdisciplinarity. In R. Frodeman, J. T. Klein & C. Mitcham (Eds.), The Oxford Handbook of Interdisciplinarity (pp. 15-30). Oxford: Oxford University Press.

[16] Melville, Herman. (1851). Moby Dick, or The Whale. London. Richard Bentley.

[17] Nelson, Theodor Holm. "A File Structure for the Complex, the Changing and the Indeterminate." Proceedings of the ACM 20th National Conference (1965), pp. 84-100

[18] Phelps, T. A., and Wilensky, R. 2000. Multivalent documents. Communications of the ACM, 43(6), 82-90

[19] Teufel, S., & Moens, M. (2002). Summarizing scientific articles: experiments with relevance and rhetorical status. Computational linguistics, 28(4), 409-445.

[20] Williams, R. (2005). The Non-designers Design Book: Design and Typographic Principles for the Visual Novice. Non Designer's Design Book.

Generation of a "semantic" eBook: all you need is XML

Vincent Gros
Hachette Livre
<vgros@hachette-livre.fr>

Jean-Claude Moissinac
Institut Mines-Télécom
<jean-claude.moissinac@telecom-paristech.fr>

Luc Audrain
Hachette Livre
<laudrain@hachette-livre.fr>

Abstract

EPUB format is the widely adopted standard compatible with modern reading devices and tablet applications In June 2014, the minor release of EPUB, EPUB 3.0.1, allows microdata and RDFa for the semantic enrichment, in addition to the existing semantic inflection provided by the epub:type attribute. But producing semantic eBooks "by hand" can be time-consuming and expensive for a publisher. Therefore we propose to think about an automated generation based on a XML workflow, extended by inspiration of Schematron processing with XSLT transformations driven by pipeline in XProc. We experiment this approach to produce a French wine guide in EPUB format.

Keywords: Digital publishing, eBook, EPUB, Schematron, semantic enrichment, XML, XProc, XSLT, workflow

1. Introduction

It used to be easy to define what a book was: a collection of pages bound inside a cover. But in the digital age, what really is a book? Probably more than a collection of pages.

According to Vassiliou and Rowley [9], "an eBook is a digital object with textual and/or other content, which arises as a result of integrating the familiar concept of a book with features that can be provided in an electronic environment. EBooks typically have in-use features such as search and cross reference functions, hypertext links, bookmarks, annotations, highlights, multimedia objects and interactive tools."

Therefore we consider here the eBook as a package of logically structured and formatted data semantically enhanced. In this way the information of the book and the content are integrated: the eBook is a self-descriptive set of content. This definition is inspired by the notion of semantic-document described as a combination of PDFs Files and ontologies by Henrik Eriksson [2].

In other words, we suppose an eBook must contain two representations:

- A representation of the logical structure of the book ;
- A representation of the informational structure of the data in the book.

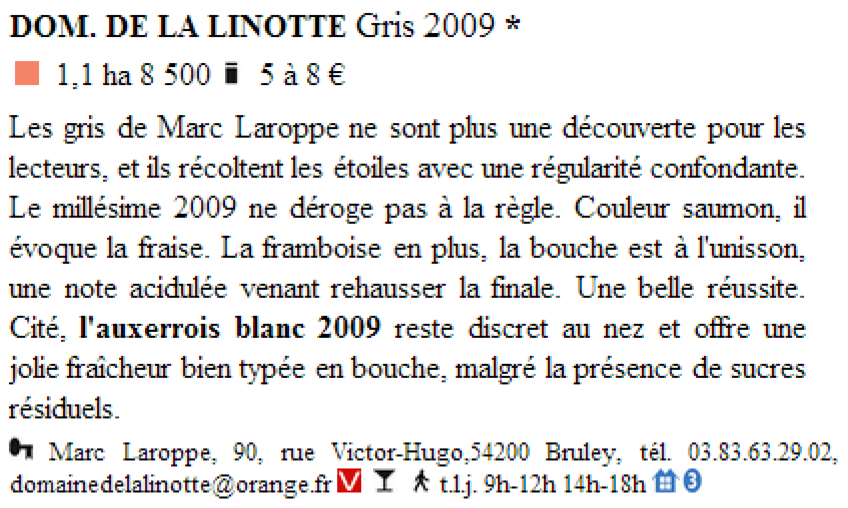
Both representations can be collected at the beginning of an eBook production: in the publishing industry, the production is mainly based on XML source files [1], and in addition Melnik assumes in [5] that every XML document has an implicit RDF[2] model. But here we consider two semantic representations depending on two different mechanisms, which are interlaced with the human readable presentation expressed by the layout. This is now available thanks to the last release of the EPUB standard.

In an industrial context, a publisher can have heterogeneous source files, depending on different DTDs or XSD Schemas. Producing every semantic eBooks as a single operation can be time-consuming and expensive for a publisher. How can we ease semantic EPUB generation in this context? In this paper, we give our first thoughts about an XML workflow from XML source to semantic EPUB.

2. A French wine guide as XML input

In our experimentation, we use a French guide about French, Swiss and Luxembourgian wines. The information in this kind of books is extremely rich and diverse. Within this guide, for example, every wine is described according to its characteristics (the type, the color), the year (of the vintage), the price, etc. There is also an evaluation of the wine (from blank to ***, if the wine is very tasty). For human readers, these kinds of information are indicated by explicit text, by a specific picture, or both (illustrated by the figure 1 from the 2011 edition).

[2] http://www.w3.org/RDF/



DOM. DE LA LINOTTE Gris 2009 *

🟥 1,1 ha 8 500 ▮ 5 à 8 €

Les gris de Marc Laroppe ne sont plus une découverte pour les lecteurs, et ils récoltent les étoiles avec une régularité confondante. Le millésime 2009 ne déroge pas à la règle. Couleur saumon, il évoque la fraise. La framboise en plus, la bouche est à l'unisson, une note acidulée venant rehausser la finale. Une belle réussite. Cité, **l'auxerrois blanc 2009** reste discret au nez et offre une jolie fraîcheur bien typée en bouche, malgré la présence de sucres résiduels.

Marc Laroppe, 90, rue Victor-Hugo,54200 Bruley, tél. 03.83.63.29.02, domainedelalinotte@orange.fr t.l.j. 9h-12h 14h-18h

Figure 1. Display of a wine description (2011)

The informational structure is directly predictable by a human reader from the XML markups chosen by the publisher (see example 1). For example, the tag `PRIX` stands for the prices information[3]. But there are no direct hints for the logical structure of the book (chapter, section, etc.).

Example 1. A wine description in the XML source

```
<VIN>
   <NOMVIN>Dom. de la Linotte</NOMVIN>
   <COMPLVIN>Gris</COMPLVIN>
   <MILLESIM>2009</MILLESIM>
   <APPREC>*</APPREC>
   <CARACT>
          <TYPE TYPE="1" COULEUR="2"/>
          <SURFACE>1,1 ha</SURFACE>
          <NBBOUT>8 500</NBBOUT>
          <CUVE/>
          <PRIX VALEUR="5 à 8 €"/>
   </CARACT>
   <TEXVIN>Les gris de Marc Laroppe ne sont plus une découverte pour les ▶
lecteurs, et ils récoltent les étoiles avec une régularité confondante. […]</▶
TEXVIN>
   <PRODUCT CODE="24759" NOUVEAU="Non">
                <DEBPROD>Marc</DEBPROD>
                <INDPROD>Laroppe</INDPROD>
                <Adresse>90, rue Victor-Hugo</Adresse>
                <CP>54200</CP>
                <INDCOMM>Bruley</INDCOMM>
```

[3]In French, 'prix' means 'price'.

```
                <Tel>03 83 63 29 02</Tel>
        <EMAIL>domainedelalinotte@orange.fr</EMAIL>
                <VENTE/>
                <HOTE PRIX="2"/>
                <VISITES/>
                <DEGUST/>
                <HORAIRES>t.l.j. sf dim. 9h-12h 14h-18h</HORAIRES>
    </PRODUCT>
  </VIN>
```

In the next section we will describe the expected EPUB output.

3. The EPUB output or the "semantic EPUB"

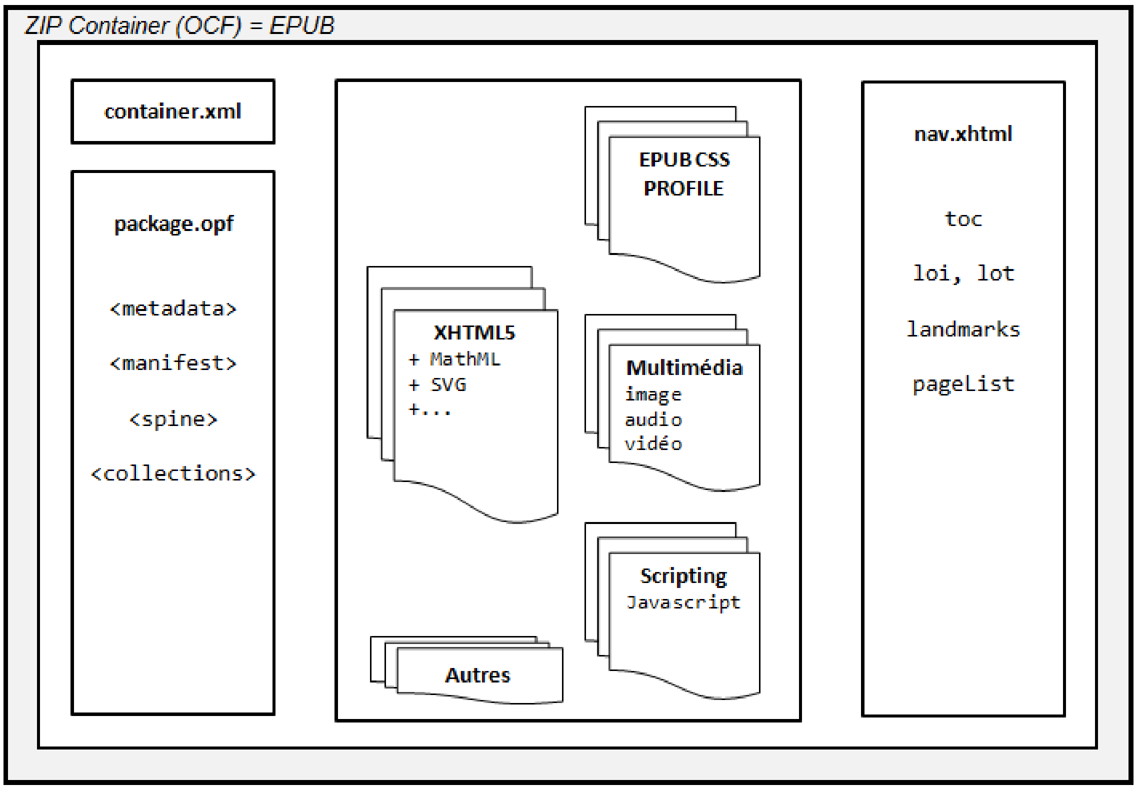


Figure 2. Schema of the EPUB package

The Open EBook Standard was replaced in September 2007 by the Open Publication Structure 2.0, which evolved into the EPUB standard (for Electronic PUBlication). Developed by the IDPF (International Digital Publishing Forum), this format is a standard widely adopted and compatible with modern reading devices and tablet applications. The third major version of the EPUB standard has been published in

October 2011, and is based on the Open Web Platform, mainly HTML5, CSS3, JavaScript and SVG (figure 2).

In June 2014, the minor release of EPUB, named EPUB 3.0.1[4], allows microdata [3] and RDFa [7] for semantic enrichment, in addition to the existing semantic inflection provided by the `epub:type` attribute. We present these mechanisms in the following sections.

3.1. The logical structure using epub:type

One of the limitations of the core HTML markup grammars is that it is not well suited for defining a rich structure, because of its small set of elements.

To consider the logical structure of the book, EPUB includes a specific `epub:type` attribute[5]

. This `epub:type` attribute can be attached to any element in the body of an HTML document, and it accepts any of the terms defined in the EPUB Structural Semantics Vocabulary[6] by default, or other terms by defining a namespace.

For example, example 2 is a basic example of using the semantic inflection.

Example 2. A Content Document in EPUB using semantic inflection.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html>
<html xml:lang="en" lang="en" dir="ltr" xmlns="http://www.w3.org/1999/xhtml" ▶
xmlns:epub="http://www.idpf.org/2007/ops">
 <head>
  <title title="The title of the book">The title of the book</title>
  <link rel="stylesheet" href="../Style/style.css" type="text/css"/>
  <meta charset="utf-8" />
 </head>
 <body epub:type="bodymatter">
 <section epub:type="chapter">
  <header>
   <h1 epub:type="title">Title of the chapter</h1>
   <p epub:type="subtitle">The subtitle</p>
  </header>
  <section>
   <header>
    <h1 epub:type="title">The title of the section</h1>
   </header>
   <p>Here is the first paragraph of the section with some texts, and a ▶
```

[4] http://idpf.org/epub/301

[5]Also Tips n°4 and 5 for accessibility according to Benetech, see http://diagramcenter.org/54-9-tips-for-creating-accessible-epub-3-files.html.

[6] http://www.idpf.org/epub/vocab/structure/

```
noteref<a href="#footnote-1" id="footnote-1-backlink" epub:type="noteref">[1]</▶
a>.</p>
   <p>A second paragraph here.</p>
   <p>A third paragraph.</p>
   <aside id="footnotes" epub:type="footnotes">
    <article id="footnote-1" epub:type="footnote">
     <p class="footnotes">[<a href="#footnote-1-backlink" ▶
id="footnote-1-link">1</a>] One footnote.</p>
    </article>
   </aside>
  </section>
 </section>
 </body>
</html>
```

3.2. Semantic enrichment using RDFa

The recent release 3.0.1 of the EPUB standard allows the integration of RDFa in Content Documents (see the Content Document specification[7]).

For example, a simplified RDFa description of a wine could be as in example 3 (adapted in English, and with addition of `epub:type` attributes):

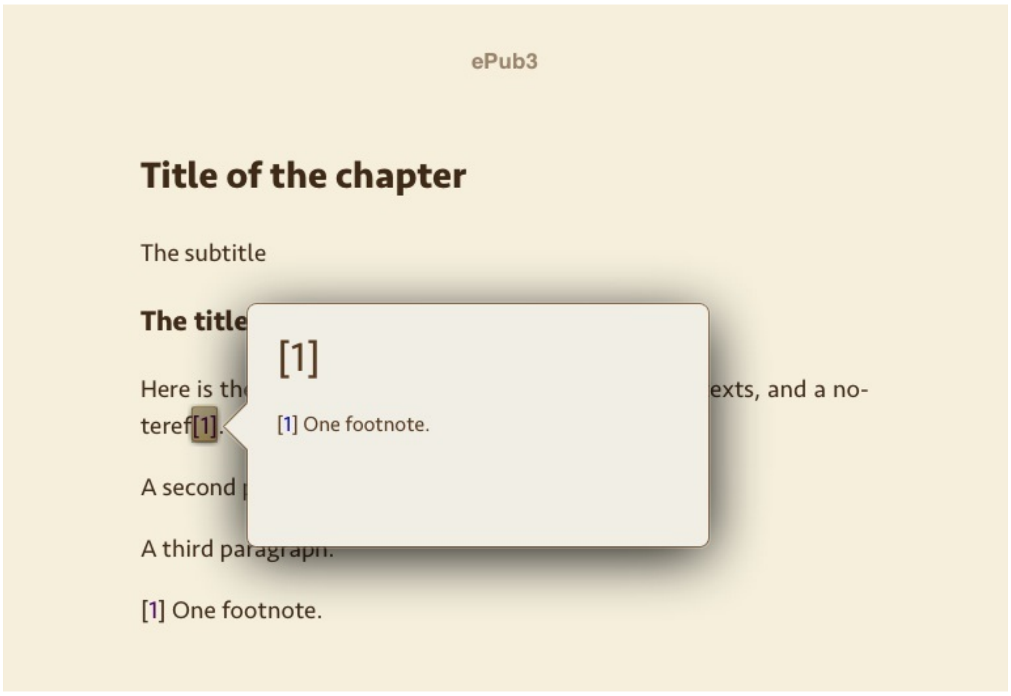
Example 3. An HTML fragment with a RDFa description of a wine.

```
<section resource="http://www.gdv.fr/2011/fr/gdv/8087" typeof="gdv:wine">
 <h1 property="gdv:nameWine" epub:type="title">Dom. De la Linotte<h1>
 <p property="gdv:subnameWine" epub:type="subtitle">Gris</p>
 <div property="gdv:vintage">2009</div>
 <div property="gdv:note">*</div>
 <div property="gdv:type">1</div>
 <div property="gdv:color">1</div>
 <div class="gdv:price">5 à 8€</div>
</section>
```

[7] http://www.idpf.org/epub/301/spec/epub-contentdocs.html#sec-xhtml-semantic-enrichment

3.3. Existing use cases

3.3.1. Interact with the book structure: pop-up footnotes


ePub3

Title of the chapter

The subtitle

The title

Here is th exts, and a no-
teref[1].

A second

A third paragraph.

[1] One footnote.

[1]

[1] One footnote.

Figure 3. Pop-up footnotes in iBooks (iOS)

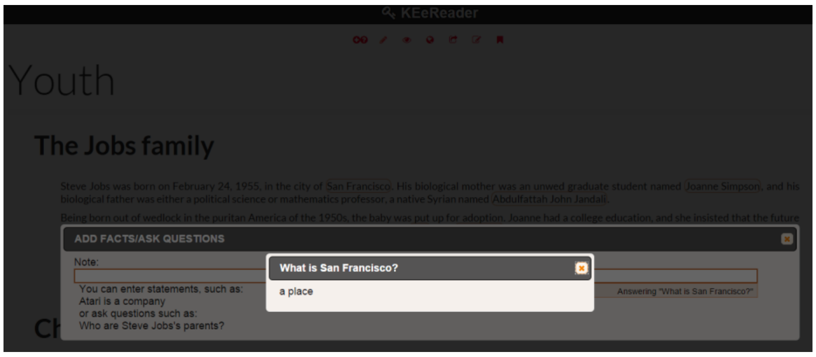
Since May 2012, Apple has enabled pop-up footnotes using the associated values "noteref" and "footnote" of the epub:type attribute (figure 3).

Then, there have also been other reading systems, for example the Azardi EPUB3 reader[8], which propose pop-up footnotes and other applications (for example, center the cover when occurs an`epub:type="cover"`).

3.3.2. Smart reasoning in KEereader

KEereader[9] (for Knowledge Enhanced eReader) is a web reading system developed by Eric Freese. In addition to current reading system features, KEereader provides a smart search functionality based on RDFa markup into the eBooks content. A natural language query parser allows users to ask on the concepts.

[8] http://www.infogridpacific.com/blog/igp-blog-20120115-epubtype-properties.html
[9] http://www.keereader.com



Youth

The Jobs family

Steve Jobs was born on February 24, 1955, in the city of San Francisco. His biological mother was an unwed graduate student named Joanne Simpson, and his biological father was either a political science or mathematics professor, a native Syrian named Abdulfattah John Jandali.

Being born out of wedlock in the puritan America of the 1950s, the baby was put up for adoption. Joanne had a college education, and she insisted that the future

You can enter statements, such as:
Atari is a company
or ask questions such as:
Who are Steve Jobs's parents?

What is San Francisco?

a place

Answering "What is San Francisco?"

Figure 4. Example of an eBook with knowledge marks (example from semanticweb.com)

4. Our approach

4.1. The workflow

The DiVA project was developed at Uppsala University Library and considers the electronic version of a document as the core for multiple output and enhancement of metadata [6]. Our approach is very similar.

Introduce XML into the production process at an early stage can make it the basis of the entire production workflow. The figure 6 represents our simplified XML workflow. In this case, an XML-tagged book is used as the basis of the print and the eBook production. At the bottom, we added an extension to generate a semantic eBook.

We introduce 3 important steps:

- In collaboration with the publisher, a person which can be seen as a Knowledge Engineer considers the validation files (in our work, DTDs or XSD Schemas) to define the mapping rules;
- These Mapping Rules will be used as a base for the generation engine;
- The Generation Engine is the automatic process based on the XML source file, the mapping and the composition rules in order to generate the semantic eBook.

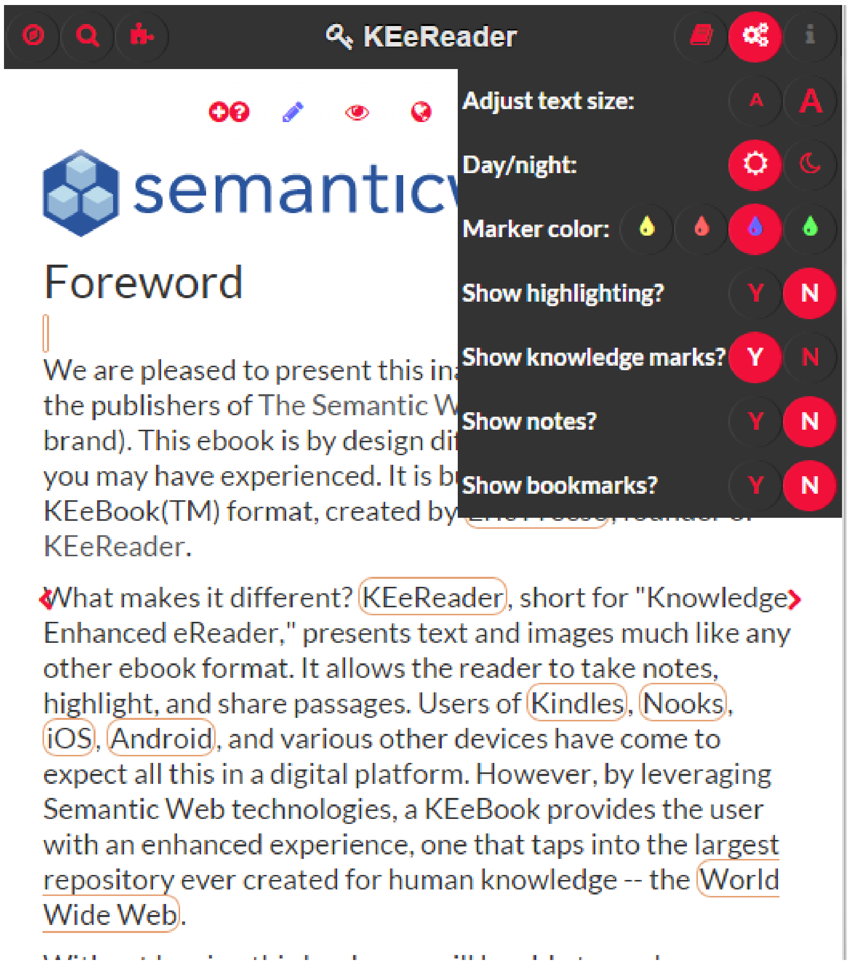
4.2. The mapping rules

Written by the Knowledge Engineer, the mapping rules are set to enable both logical structure and semantic enrichment into the EPUB output.

The format used for these rules must distinguish the rules two types (structural and semantic), in order to let the Generation Engine generate the output accordingly.

The most basic rules directly concern an XML element but others can depend on complex patterns. An element can also be the base of several rules.

Actually we consider 3 components:


KEeReader

Adjust text size: A A

Day/night:

Marker color:

Show highlighting? Y N

Show knowledge marks? Y N

Show notes? Y N

Show bookmarks? Y N

semantic

Foreword

We are pleased to present this in
the publishers of The Semantic W
brand). This ebook is by design di
you may have experienced. It is b
KEeBook(TM) format, created by
KEeReader.

What makes it different? KEeReader, short for "Knowledge Enhanced eReader," presents text and images much like any other ebook format. It allows the reader to take notes, highlight, and share passages. Users of Kindles, Nooks, iOS, Android, and various other devices have come to expect all this in a digital platform. However, by leveraging Semantic Web technologies, a KEeBook provides the user with an enhanced experience, one that taps into the largest repository ever created for human knowledge -- the World Wide Web.

Figure 5. The natural query box (use in the Steve Jobs biography)

- The tag itself;
- Its DOM position; this can be done using XPath ;
- A simple reference to a notable pattern: the first characters of the content of the element, the number of element, etc. This can also be done using XPath.

According to the semantic enrichment, concepts or properties of a concept are defined from the XML source. For example, the `VIN` element applies to the concept `gdv:Wine`[10]

, and its child `NOMVIN` applies to the property `gdv:nameWine`.

The elements of the logical structure of the book are also indicated the same way from the XML source. For example, the `VIN` element applies to the section level and `NOMVIN` is the title of the section.

The mechanisms from the mapping rules to the generation engine are inspired by the Schematron framework and will be exposed in the next section.

[10]'gdv' is the prefix of the wines guide's namespace.

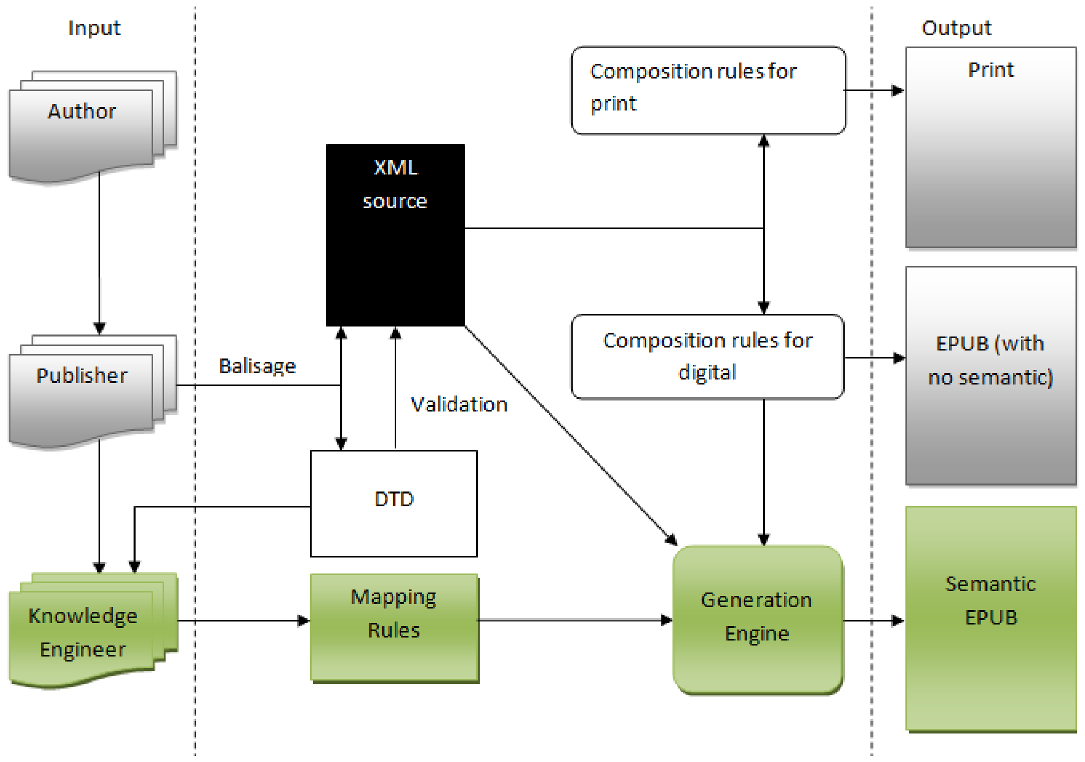


Figure 6. Schema of the extended XML workflow

5. The "Schematron flavor"

5.1. Introduction

The ISO Schematron is a language and toolkit for making assertions about the presence or absence of patterns in XML documents [8]. It can be used instead of DTDs or XSD Schemas or to complement them. Schematron is a validation framework, not a generation language. But one of the Schematron strengths is that it is easily implementable using XSL Transformations. It is suitable for use in an XML Pipeline (for example the W3C XProc[11] pipelining language or Apache Ant[12]).

The core syntax is simple, based on `pattern` elements which contains `rule` depending on a specific `context`, usually expressed in XPath. Every `rule` can contain mixed `assert` or `report` elements depending on the `test` location (also in XPath).

The choice of the Schematron syntax to express semantic rules is first based on the fact that the syntax is relatively simple and human-readable. Rick Jelliffe also wrote about Schematron in [4]: "Its target uses are for software engineering requiring document validation, for scholarly research over patterns in graph-structured data,

[11] http://www.w3.org/TR/xproc/
[12] http://ant.apache.org/

for automatic creation of external markup, and to aid accessibility of documents for people with disabilities".

A classic Schematron process does not augment the information set of an XML document. But as an experimentation, we define a specific Schematron profile to write our mapping rules. There are some unused or underused attributes in which we can put information for the generation

5.2. From Schematron to our approach

Some implementation of Schematron is based on meta-XSLT (XSLT that generates XSLTs), that is the case for the leading Schematron implementation made by Rick Jelliffe called Skeleton[13]. We can draw our implementation from the Skeleton XSLT process, as shown in the figure 6. The generation engine applies the meta-XSLT on the mapping rules (in a Schematron file) to build a generation's XSLT. Then this resulting XSLT is applied on the XML source to generate the content documents of the EPUB package (a folder of resources which can be zipped in an EPUB file).

5.3. From Schematron rules to mapping rules

The Schematron rules are used by the meta-XSLT to build XSLT fragments. First, We present the correspondences between Schematron and the resulting XSLT in the table 1. The `assert` and `report` elements are interpreted simply onto XSLT conditional blocks, whose test is specified by the `test` attribute of both Schematron elements. All `rule` elements within a single `pattern` share the same `mode`, specified by the `name` of the `pattern` element, and also define an `xsl:template` block.

The process of our meta-XSLT is quite different than the Schematron one, because the output is not a validation report, but a set of files that constitute the same content than the source. That said, Schematron elements in a mapping file also correspond to XSLT blocks. For example, the `assert` element define a property of a concept (in the concepts mode corresponding to the informational structure) or a structural book element. Both logical and informational representations apply mechanisms as shown in the table 2 and the table 3.

6. Discussion

A first sample[14]was made in EPUB3 format without semantic enrichment. We use it to do a visual inspection and to confirm the complete readability of the EPUB.

That said, the integration of the tool EpubCheck at the end of the process confirms that the output semantic EPUB is well-formed. Also, we've extend the XML log

[13] http://www.schematron.com/implementation.html
[14] https://code.google.com/p/epub-samples/wiki/SamplesListing#GhV-oeb-page.epub

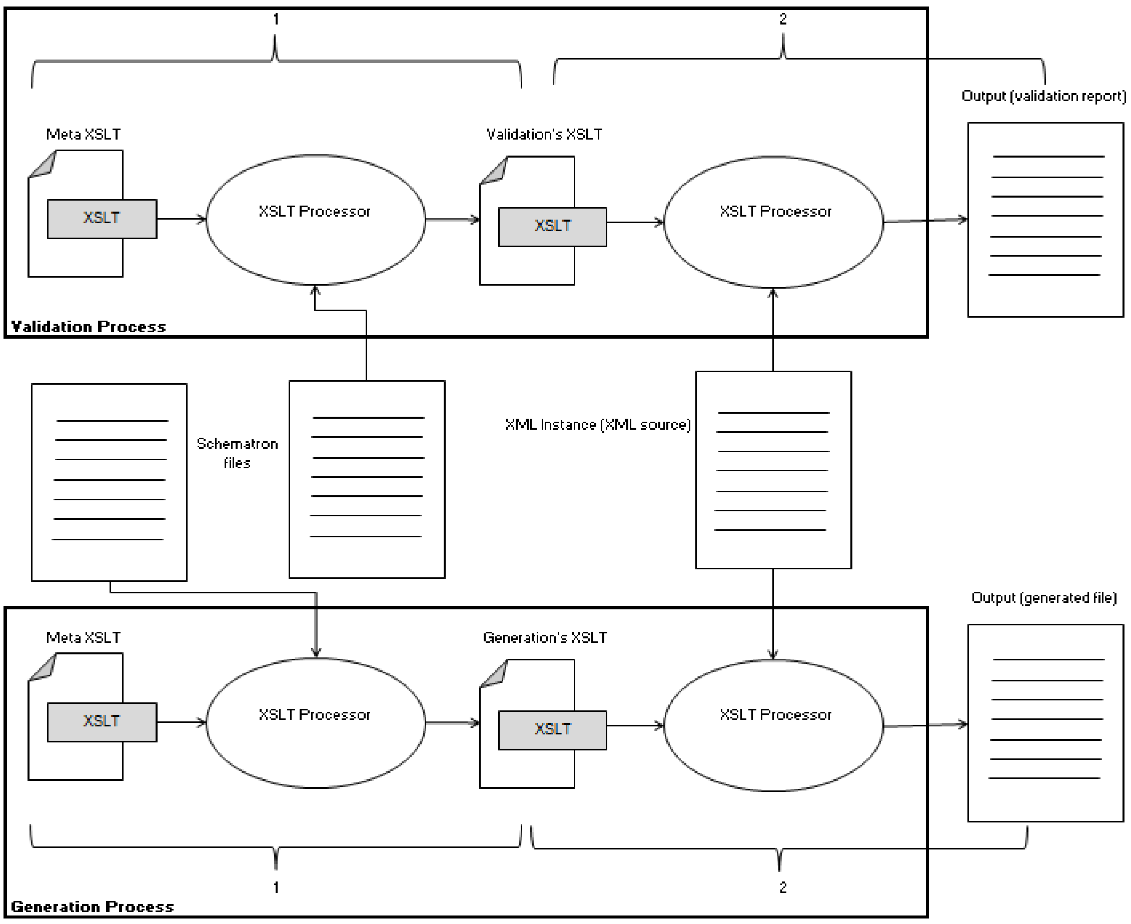


Figure 7. Comparison schema between Schematron process and our generation process

option of this tool (which is open-source) to provide some information: the metadata, the number of character and the list of the `epub:type` attributes.

The result is a semantic EPUB, but not a "full" semantic EPUB. We can have information which requires normalization, for example the opening times of a wine domain, like "t.l.j. sf dim. 9h-12h 14h-18h" (see figure 1 and example 1), which means 'open daily except Sunday, from 9am to noon and 2pm to 6pm'.

Also, the content semantic is based on our own vocabulary, but could reuse well-known vocabularies like FOAF[15] or schema.org.

[15] http://www.foaf-project.org/

Table 1. Implementation of core Schematron elements into XSLT

Schematron Schema	**XSLT**
Assert	Choose

`<sch:assert test="XPATH">message</sch:assert>`	`<xsl:choose>`
	` <xsl:when test="XPATH"/>`
	` <xsl:otherwise>message</xsl:otherwise>`
	`</xsl:choose>`

Report	If

`<sch:report test="XPATH">message</sch:report>`	`<xsl:if test="XPATH">`
	`  message`
	`</xsl:if>`

Rule	Template

`<sch:rule context="XPATH">`
`  <!-- asserts and/or reports here -->`
`</sch:rule>`
	`<xsl:template match="XPATH"`
	`              mode="NAME-OF-PATTERN">`
	`  <!-- xsl:if and/or xsl:choose here -->`
	`</xsl:template>`

Pattern	Pattern

`<sch:pattern name="NAME-OF-PATTERN">`
`  <!-- rules here -->`
`</sch:pattern>`
	`<xsl:apply-templates mode="NAME-OF-PATTERN"/>`

Table 2. Implementation of our Schematron mapping element into XSLT (concepts mode)

Schematron Schema	**XSLT**
Assert	Template

`<sch:assert test="XPATH">message</sch:assert>`	`<xsl:template match="..." mode="concepts">`
	`  <!-- @property -->`
	`</xsl:template>`

Rule	Template

`<sch:rule context="XPATH"`
`          role="NAME-OF-CONCEPT">`
` <!-- asserts here -->`
`</sch:rule>`
	`<xsl:template match="XPATH" mode="concepts">`
	`  <!-- @resource and @typeof -->`
	`  <!-- xsl:if and/or xsl:choose here -->`
	`</xsl:template>`

Pattern	Pattern

`<sch:pattern name="concepts">`
`  <!-- rules here -->`
`</sch:pattern>`
	`<xsl:apply-templates mode="concepts"/>`

Table 3. Implementation of our Schematron mapping element into XSLT (book mode)

Schematron Schema	**XSLT**
Assert	Template
`<sch:assert test="XPATH">message</sch:assert>`	`<xsl:template match="..." mode="book"> <!-- structural item without structural descendants --> </xsl:template>`
Rule	Template
`<sch:rule context="XPATH" role="NAME-OF-STRUCTURAL-ITEM"> <!-- asserts here --> </sch:rule>`	`<xsl:template match="XPATH" mode="book"> <!-- structural item --> <!-- xsl:if and/or xsl:choose here --> </xsl:template>`
Pattern	Pattern
`<sch:pattern name="book"> <!-- rules here --> </sch:pattern>`	`<xsl:apply-templates mode="book"/>`

7. Conclusion

In this paper, we proposed to consider the semantic eBook as the result of all the information existing in an XML-tagged book source: it contains the data and their meaning if available, and the logical structure of the book. In the vision of different and heterogeneous XML sources, we consider an approach in which a Knowledge Engineer defines the Semantic Rules in order to automatically generate the semantic eBook. We applied this approach with Schematron syntax for rules and XSLTs to show the relevance of this approach. The output is an EPUB file which contains all the structural semantic and the content semantic defined in the Schematron schema.

By this approach, an eBook is a whole logical and semantic entity. The next steps of this work are first to improve this approach by more complex rules, and also to think about the use of all the semantics provided in the semantic eBook.

Bibliography

[1] Barron, D. (1989). Why use sgml? Electronic publishing, vol. 2, 3–24.

[2] Eriksson, H. (2007). The semantic-document approach to combining documents and ontologies. Int. J. Hum.-Comput. Stud. 65, 7, pages 624-639.

[3] Hickson, I. (2012). HTML Microdata. W3C Working Draft. http://www.w3.org/TR/2012/WD-microdata-20121025/

[4] Jelliffe, R. (2002). The Schematron Assertion Language 1.6. Academia Sinica Computing Centre.

[5] Melnik, S. (1999). Bridging the gap between XML and RDF. Université de Stanford. http://infolab.stanford.edu/~melnik/rdf/fusion.html

[6] Müller, E, Sandgren, F., Andersson, S., Klosa, U., Hansson, P. (2005). DiVA Publishing System. The Community Source Development Approach. In Proceedings of the 9th International Conference on Electronic Publishing, ELPUB 2005, Leuven-Heverlee, Belgique, 23-27.

[7] Sporny M. (2013). HTML+RDFa 1.1. Support for RDFa in HTML4 and HTML5. W3C Recommendation. http://www.w3.org/TR/2013/REC-html-rdfa-20130822/

[8] Van der Vlist, E. (2007). Schematron. O'Reilly.

[9] Vassiliou, M., Rowley, J. (2008). Progressing the definition of "e-book". Library Hi Tech, vol. 26, num. 3, 355-368.

Building Security Analytics using Native XML Database

Mansi Sheth
Veracode
<msheth@veracode.com>

Abstract

The trove of ever-expanding metadata we are collecting on a daily basis, poses us with the challenge of mining information out of this data-store, to help drive our business analytics solutions. Most non-destructive format of these metadata is in XML formats, so it became crucial to use a technology, which provides sophisticated support for XML specific query technologies. This paper will discuss how Veracode is using Native XML Databases(NxD) tool BaseX, to solve various use cases across multiple departments. It will discuss in depth, how its incorporated, its architecture and eco-system. It will also touch base on lessons learned along the way, including approaches which were tried and didn't work so well.

1. Introduction

As a SaaS security static binary analysis company, we analyze binary code of world's largest organizations. To help secure them better we collect various kinds of metadata about each application we analyze. Some examples of these metadata are third party api/frameworks/libraries usage patterns for each client application, detected entry points, scan quality metrics etc. All these across different languages and technologies. Amount of metadata currently persisted per application averages at around 100MB and we scan on an average around 200 applications per day. To add to this, there are ideas needed to be implemented, data sources to be utilized which will just add data to this BigData store.

Being able to observe what customers are doing with application development is one of the major differentiators of a SaaS service. We envisioned building a security analytics system, by feeding it data from these metadata sources, would help us fuel in evidence-based answer, making us more proactive and innovative.

Enumerating a few straightforward situations, where we could transform Big Data into actionable insights are:

- With new languages and frameworks surfacing every day, we are constantly pushed to be able to find security vulnerabilities resulting from usage of these new technologies. By having a data store of their usage patterns across our clients and also out in the industry, product managers can gain predictive insights on our product strategies, resulting in optimized offerings.

- As part of security research, we routinely do a deep dive on control and taint flow on various languages and frameworks. Some of these could be gargantuan in nature. It would be much more directed approach, to know what parts of a particular language or framework is being used most extensively and prioritize our research accordingly. This approach guarantees, covering maximum breath of framework across our customers in our initial offering of respective framework support.
- Routinely, we come across, new zero day vulnerabilities or an insecured api being exploited in wild. With data and tools, to mine such vulnerable api usage across our clients, it becomes easier isolate vulnerable clients from among hundreds of them, alert and protect them promptly.

Due to the complexity of enterprise applications, binaries we scan are often deeply nested (archives contained within archives and so on), till we reach the files we need to parse for information. Further, all applications have different levels of nesting. This nature of our metadata, we chose to store in XML format as opposed to any other formats like json, text, csv etc. XML structure also gives us advantage of using various advanced query and transformation technologies like XPATH, XQUERY etc, with support across most modern programming languages. Additionally, with many different kinds if metadata, they are persisted in different schema. With XML representation, data retrieval in such varied format becomes trivial.

After analyzing and trying a few rudimentary approaches to extracting required data, it became clear we needed to have a system which could handle such gigantic amounts of data, and return results in timely fashion, a.k.a within minutes or couple of hour and not days.

This paper discusses, by means of examples some sample XML metadata and gains an appreciation for the need to use various advanced XML technologies (XPATH, XQUERY etc) for extracting data efficiently. Further, it briefs about various technologies analyzed and/or tried, but didn't work, lessons learned on the way. Finally, it talks in detail about how Native XML Database tool BaseX is being used, its architecture and eco-system, demonstrating how some of the above use cases are being executed concluding with some statistics.

This paper talks about approaches towards data mining and analysis, which is yet mostly in Proof Of Concept phase. There are lot of performance optimizations, enhancements and security infrastructure which needs to be employed. It just tries to focus on the core technology and how well it works for our security analytics needs.

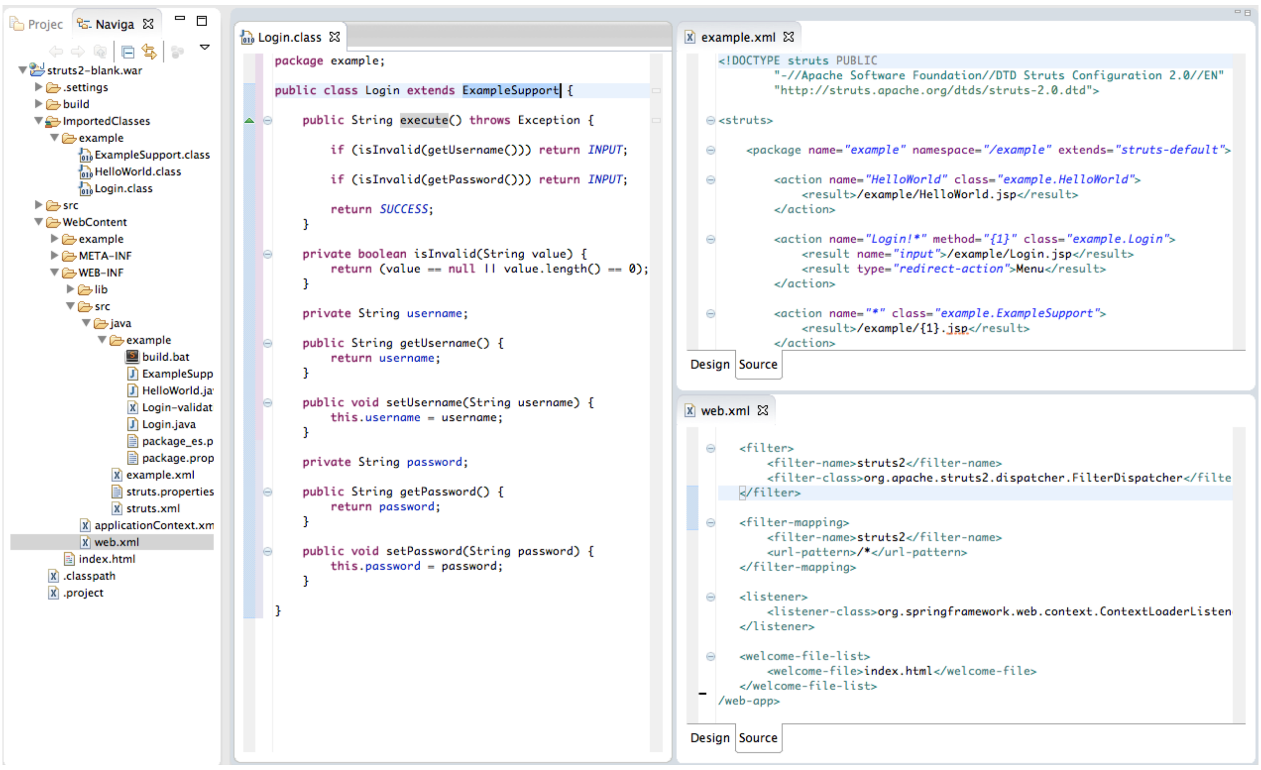
2. What is these metadata and what do we do with it ?

To put this discussion in persceptive, will describe one of our example metadata structure.

2.1. Metadata Structure

One of the biggest client application artifact, we are persisting currently is output of our home grown tool "**Binary Miner**". Main responsibility of "Binary Miner", is to capture exact 3rd party API being used; uptill function level from client binaries.

For example, consider sample struts project struts-blank.war[1], exploded in eclipse:



```java
package example;

public class Login extends ExampleSupport {

    public String execute() throws Exception {

        if (isInvalid(getUsername())) return INPUT;

        if (isInvalid(getPassword())) return INPUT;

        return SUCCESS;
    }

    private boolean isInvalid(String value) {
        return (value == null || value.length() == 0);
    }

    private String username;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```

```xml
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

    <package name="example" namespace="/example" extends="struts-default">

        <action name="HelloWorld" class="example.HelloWorld">
            <result>/example/HelloWorld.jsp</result>
        </action>

        <action name="Login!*" method="{1}" class="example.Login">
            <result name="input">/example/Login.jsp</result>
            <result type="redirect-action">Menu</result>
        </action>

        <action name="*" class="example.ExampleSupport">
            <result>/example/{1}.jsp</result>
        </action>
```

```xml
   <filter>
       <filter-name>struts2</filter-name>
       <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filte
   </filter>

   <filter-mapping>
       <filter-name>struts2</filter-name>
       <url-pattern>/*</url-pattern>
   </filter-mapping>

   <listener>
       <listener-class>org.springframework.web.context.ContextLoaderListen
   </listener>

   <welcome-file-list>
       <welcome-file>index.html</welcome-file>
   </welcome-file-list>
/web-app>
```

Figure 1. struts-blank.war exploded in eclipse, to depict output of binary artifact miner tool.

Binary miner tool, will inspect every binary file typically inside deeply nested archives (.ear -> .war -> .jar -> .class is a very common topology) and extract api usage information from .class, .xml, .jsp etc files. Additionally, it will extract information like compiler version used to compile each class file, inheritance information, external function calls etc. It will also, extract configured attributes from framework configuration files like servlet and filter classes, action classes etc.

[1] https://archive.apache.org/dist/struts/examples/struts2-blank-2.0.1.war

Snapshot output of running binary miner tool on struts-blank.war is as under:

```
<Archives>
 <archive name="struts2-blank-2.0.1.war">
   <class major="0" minor="0" name="example.HelloWorld.jsp" />
   <class major="0" minor="0" name="example.Login.jsp" />
   <class major="0" minor="0" name="example.Menu.jsp" />
   <class major="0" minor="0" name="example.Missing.jsp" />
   <class major="0" minor="0" name="example.Register.jsp" />
   <class major="0" minor="0" name="example.Welcome.jsp" />
   <class major="0" minor="0" name=".index.html" />
   <class major="0" minor="0" name="WEB-INF.applicationContext.xml" />
   <class major="49" minor="0" name="example.ExampleSupport">
     <inheritance name="com.opensymphony.xwork2.ActionSupport" />
     <function modifier="public" name="&lt;init&gt;">
       <apiCalls name="com.opensymphony.xwork2.ActionSupport:&lt;init&gt;" />
     </function>
   </class>
   <class major="49" minor="0" name="example.HelloWorld">
     <inheritance name="example.ExampleSupport" />
     <function modifier="public" name="&lt;init&gt;">
       <apiCalls name="example.ExampleSupport:&lt;init&gt;" />
     </function>
     <function modifier="public" name="execute">
       <apiCalls name="example.HelloWorld:getText" />
       <apiCalls name="example.HelloWorld:setMessage" />
     </function>
   </class>
   <class major="0" minor="0" name="WEB-INF/classes/▶
example.Login-validation.xml" />
   <class major="49" minor="0" name="example.Login">
     <inheritance name="example.ExampleSupport" />
     <function modifier="restricted" name="isInvalid" />
     <function modifier="public" name="&lt;init&gt;">
       <apiCalls name="example.ExampleSupport:&lt;init&gt;" />
     </function>
     <function modifier="public" name="execute">
       <apiCalls name="example.Login:getUsername" />
       <apiCalls name="example.Login:isInvalid" />
       <apiCalls name="example.Login:getPassword" />
       <apiCalls name="example.Login:isInvalid" />
     </function>
   </class>
   .
   .
   .
   <class major="0" minor="0" name=".WEB-INF/classes/struts.properties" />
   <class major="0" minor="0" name="WEB-INF/classes.struts.xml" />
   <archive name="WEB-INF/lib/commons-collections-3.1.jar">
   </archive>
   <archive name="WEB-INF/lib/commons-logging-1.0.4.jar">
   </archive>
   <archive name="WEB-INF/lib/freemarker-2.3.4.jar" />
   <archive name="WEB-INF/lib/ognl-2.6.7.jar" />
   <archive name="WEB-INF/lib/spring-aop-1.2.8.jar" />
   <archive name="WEB-INF/lib/spring-beans-1.2.8.jar" />
   <archive name="WEB-INF/lib/spring-context-1.2.8.jar" />
   <archive name="WEB-INF/lib/spring-core-1.2.8.jar" />
   <archive name="WEB-INF/lib/spring-web-1.2.8.jar" />
   <archive name="WEB-INF/lib/struts2-api-2.0.1.jar">
   </archive>
   <archive name="WEB-INF/lib/struts2-core-2.0.1.jar">
   </archive>
   <archive name="WEB-INF/lib/xwork-2.0-beta-1.jar">
   </archive>
   <class major="0" minor="0" name="WEB-INF/src/java/▶
example.Login-validation.xml"/>
   <class major="0" minor="0" name=".WEB-INF/src/java/example/▶
package.properties"/>
   <class major="0" minor="0" name=".WEB-INF/src/java/example/▶
package_es.properties"/>
   <class major="0" minor="0" name="WEB-INF/src/java.example.xml">
     <function modifier="restricted" name="action/@name">
       <apiCalls name="HelloWorld" />
       <apiCalls name="Login!*" />
       <apiCalls name="*" />
     </function>
   </class>
   <class major="0" minor="0" name=".WEB-INF/src/java/struts.properties" />
   <class major="0" minor="0" name="WEB-INF/src/java.struts.xml" />
   <class major="0" minor="0" name="WEB-INF.web.xml">
     <function modifier="restricted" name="filter-class">
       <apiCalls name="org.apache.struts2.dispatcher.FilterDispatcher" />
     </function>
   </class>
   .
   .
   .
 </archive>
</Archives>
```

If we map the exploded files of struts-blank application with the above output, we can see how individual elements are enumerated in the output. For example, compiled version of "Login.class" file is mined for all its function definitions and all api calls within it. Similarly, tool is configured to extract properties from configuration files. Some of the properties extracted in above example is filter-class defined in web.xml and action elements from struts configuration files.

Seldom real world applications are this simple. They would typically be much bigger and have a relatively much more nested structure making representation in hierarchical structured data like XML most viable choice. Advanced XML querying technologies (XPATH, XQUERY, XSLT) are supported in most of the modern languages, making it much easier to use out of box tools for mining and building an analytic solution.

We started persisting, above discussed metadata across few different enterprise languages. This resulted in GBs of data being persisted on a weekly basis, making our requirements to have a highly scalable solution and retrieving results within a reasonable time frame, almost a must have requirement.

2.2. Common Use cases

Results for some of the most fundamental queries needed by various groups in the company are enlisted below:

- List of all classes and functions within package xyz.abc used across all applications
- List of all applications using an insecured function call (for example f_insecured()) from package abc.xyz.pqr
- Create a prevalence chart. This should give a snapshot of configured namespaces and its prevalence across entire client base.

3. Data mining approaches tried and didn't work

Very quickly, we solved the problem of persisting these valuable metadata spanning across different schema. Soon, enough we started getting constrained to simplistic analyses because of the sheer volumes of data.

There has been an explosion of technologies available to deal with the challenge of data mining. There were a few approaches for extracting information, which we prototyped or brainstormed before we considered Native XML Databases. This should give few chuckles, to practitioners out of our war stories and know how to avoid similar approaches.

- **Batch Processing Metadata XML Files:**

  First and most rudimentary approach tried, was run some XPATH against all metadata files. We did this by creating a lxml based python script, which takes XML file and XPATH as input and results from input file, by apply XPATH on that file.. Below is a code snippet:

```
from lxml import etree
from optparse import OptionParser
import sys

def evaluate_xpath(self):
      try:
              tree = etree.parse(self.xml_file)
              for each_element in tree.xpath(self.xpath):
                      print each_element
      except:
              print "Error in xpath"
```

Next, create a bash script file, which will run XPATH against all metadata files.

```
python ~/Documents/bin/evaluateXml.py -x xml_file1.xml -p '<xpath>'
python ~/Documents/bin/evaluateXml.py -x xml_file2.xml -p '<xpath>'
python ~/Documents/bin/evaluateXml.py -x xml_file3.xml -p '<xpath>'
python ~/Documents/bin/evaluateXml.py -x xml_file4.xml -p '<xpath>'
python ~/Documents/bin/evaluateXml.py -x xml_file5.xml -p '<xpath>'
.
.
.
```

Pros:

- When result set was smaller, post processing it was much manageable, making results useable.

Cons:

- Number and size of metadata files grew exponentially very quickly, making this approach unfeasible.
- Took more than an hour for results, which we got within couple of minutes thru using Native XML Databases. Making a strong case,we needed to deal with a solution, which didn't parse XML files for every single query.

- **Relational Database**

  We started inserting all above API call information in a relational database and tried to query it.

  Pros:

- When no of records in the database, were within the magnitude of millions, and we needed basic api call sites information, this approach worked.

Cons:

- As no of records in a relational table grew, query started taking hours for relatively smaller resultset to return.

  - While, it helped getting thru few of our use cases, it destructed XML's original hierarchical structured format. This made, retrieving any result sets involving nested structure of an enterprise application impossible. For e.g. tracing packaging structure of an application.

- **Hadoop and its eco-system:**

  Due to inherent features of hadoop to aid with distributed processing of large data sets and also its association with BigData, hadoop and its ecosystem became a natural choice to explore.

  Exploring hadoop topologies and technologies around it, found using Apache's Mahout and Apache Pig to be a good candidate to solve XML processing, taking advantage of distributed and scalability features of hdfs file system. However, both these technologies, at it core were treating XML files as flat streams of data and retrieving data based on external configurations of start and end tags.

  Basic pseudo code of this approach is shown in below listing. We tried by customizing XMLInputFilter class of Apache Mahout library.

```
public class XmlInputFormat extends TextInputFormat {
    public static final String START_TAG_KEY = "<class>";
    public static final String END_TAG_KEY = "</class>";



    public static class XmlRecordReader extends
            RecordReader<LongWritable, Text> {
            // initialization code
            .
            .


        @Override
        public boolean nextKeyValue() throws IOException, ▶
InterruptedException {
            // stream file and position cursor between start and end tag.
        }


        .
        .
        .

        @Override
        public Text getCurrentValue() throws IOException, ▶
InterruptedException {
            // using some helper functions around file pointer position, ▶
return
```

```
                // the queried value.

            }
        }
    }
```

Pros:

- By using hadoop's hdfs file system, could have taken advantage of scalability and distributed features for processing humongous amounts of data.

Cons:

- For basic single node conditioned query, it might have worked. But once, we had multiple hierarchical nodes as conditions (classes within jar files, which may or may not be under ear/ or war) in our query above code would have been more cluttered, making it soon unmanageable.
- Couldn't take advantage of any of XML querying technologies like XPATH or FLWOR, since at its core XML files would still be treated as unstructured text data and processing is based using file streaming.
- Multi-node advantage would still be lost, with not taking advantage of parsed XML files and using it in its native format.

- **Hadoop, AVRO and HIVE:**

  Next, we explored Apache AVRO serialization system. We investigated, while importing XML artifacts, parse and convert it into AVRO format. Once, we have XML files represented in AVRO format, we can create required schemas for data retrieval. We could use Apache HIVE, Pig or traditional mapper reducer to retrieve data.

  Pros:

- This approach looked promising, since it didn't parse thousands of huge XML files, for every single query. Making data retrieval in reasonable time frame.
- Advantage of using Hadoop eco-system giving us advantages of distributed processing and flexibility of advanced technology usage of Pig, Hive and Map Reduce for data retrieval and analysis.

Cons:

- While, promising it still had some data loss, with AVRO schema representation, raising doubts if all our current and future use cases would be satisfied.
- Not taking advantage of XML querying techniques, and learning curve of yet another SQL-like querying technology.

- **Various NoSQL Databases:**

  This was more of an exploratory exercise, with studying different kinds of NoSql databases and database products under each category. Found, Document oriented NoSQL category of databases, might be most suited for our needs.

However, while digging deeper realized, these databases, would be best suited for json representation. It could store data as xml's native format (strings or text), but if we need to use these stored records as XML and use any kind of basic XML querying technology, it would be parsed for every single query. This quickly dismissed most of the NoSQL category of databases, expect XML oriented Databases.

After considering and exploring above technologies and options, weighing advantages and disadvantages of each approach, we had a clear idea of what features we needed for our security analytic solutions.

- To gain maximum performance gain, parsed xml files are being used for every single query.
- Utilize existing XML technologies for querying
- Resultsets would most often be in millions. Such a data retrieval should be manageable and within reasonable time frame of minutes and not hours or days.
- Scalable
- REST interface, for easier usage.

We explored various document-oriented NoSQL databases, which handles data in XML format. There are mainly 2 major category of XML Databases:

- XML Enabled: Here, data may be mapped as XML to underlying traditional data-stores, but needn't be stored as XML structure. Again leading to our previous concerns.
- Native XML Database: The internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage. These databases are optimized for storage (taking into consideration, data layout, indexing etc) and retrieval (query support, access control, transaction etc) of XML databases.

After this exercise of exploring best options for our data mining and interpretation needs, we decided Native XML Databases (NXD) might be our most promising choice.

4. Native XML Databases

After comparing few Native XML Databases, found BaseX to be easiest to use. The winning features of BaseX were its powerful XQUERY processor and a light weight footprint. Also, access to stored database resources and to the XQuery engine via REST made any kind of extension extremely easy.

Our current deployment ecosystem around using Native XML Database is as under:

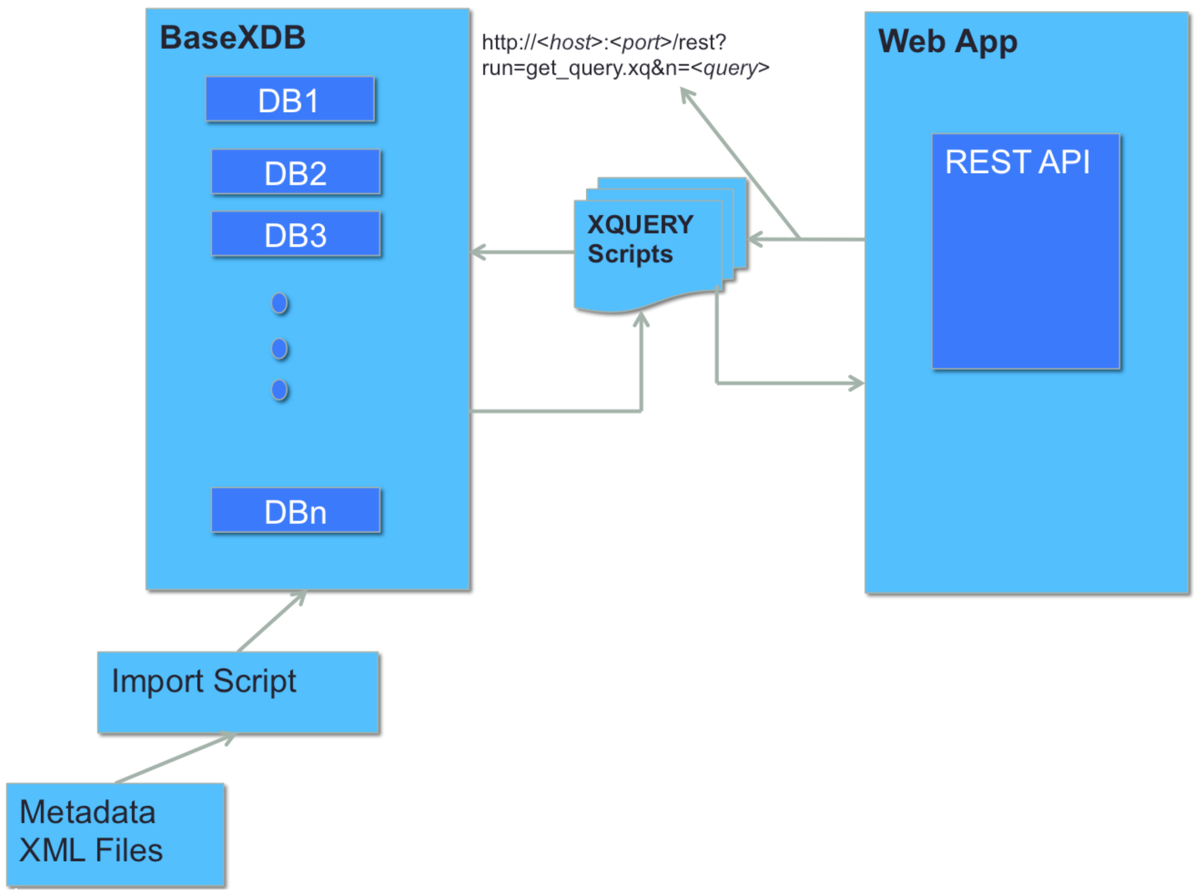


Figure 2. BaseX eco-system at Veracode

Steps performed in reaching to a point of mining results using BaseX eco-system were:

- Collect all metadata files under a single directory "**BI Output Dir**" (Metadata XML Files). These would be all files with various metadata from different data sources, various languages/technologies, different schema etc.
- Import XML documents using an import script "**Import Script**", into BaseX DB. This import script is written using BaseX's java language bindings to execute BaseX DB commands. In BaseX, each DB has limit on no of nodes, but not on no of documents managed across all its DBs. Also, with BaseX, we essentially have no limit on no of databases, as long as they all are under directory *$DBPATH;* configured in baseX configuration files. We could easily scale using symlinks when current drive space falls short. Keeping this limitations in mind, configured a limit of 50 documents per database. After all required files are imported, this script also creates attribute index across all databases.
- Once, we have our BaseX DB populated, created a few generic XQUERY scripts. XQUERY examples are discussed in below sub-section "**Sample XQUERY Scripts**"

- Enabled REST services and configured default web server based on settings discussed in BaseX Web Application[2]
- XQUERY files located on the server, could be evaluated by using "run" operation thru GET request as shown in below example.

  ```
  curl -ig 'http://<host>:<port>/rest?run=<some_xquery_script>.xq&n=<xpath>'
  ```

  In above example, "*some_xquery.xq*" file is located under $*WEBPATH* directory configured in baseX configuration file. Above GET request, passes external binding parameter to "*some_query.xq*", which runs *xpath* against the all documents in all databases under BaseX DB, and returns result set.
- To get ease access to this information, to every one in the company, including individuals who are not very xpath/xquery savvy, we developed a simple web application which conceives above REST API calls and renders requested information.

4.1. Sample XQUERY Scripts

After analyzing our most common use cases, devised a few general purpose XQUERY scripts which will adhere to most of our uses. These XQUERY scripts binds external query parameters supplied by user, generates dynamic queries from these external parameters, runs it against the against the all documents in all databases under BaseX DB and return results accordingly.

Some of these queries are discussed below:

-
  ```
  (: get_query.xq script :)
  (: external binding parameter, to be entered as "n" variable :)
  declare variable $n as xs:string external;

  declare option output:item-separator "&#xa;";
          (: Each element would be in new line :)

  (: Run input query, on every XML document in every Database:)
  for $db in db:list()
         (: Assign dynamic variables to generate query, to be used in eval :)
          let $query := "declare variable $db external; "
                       || "db:open($db)" || $n
          return xquery:eval($query,map { 'db': $db, 'query': $n })
  ```

  Above XQUERY script will execute user controlled input XPATH, run that XPATH against every document in all databases, collect and send the result back to the user. Similarly, we have XQUERY script which will return document name where XPATH matches a node etc. We call these XQUERY scripts thru REST interface as under:

[2] http://docs.basex.org/wiki/Web_Application

```
curl -ig 'http://localhost:8984/rest?run=get_query.xq&n=
/Archives/*/descendant::class/▶
descendant::apiCalls[contains(@name,"javax.servlet.http")]
/@name/string()'
```

Above query is expected to return all api calls within javax.servlet.http namespace, in every document in all databases, effectively giving us all call sites used from this namespace across all our customer binaries. This is invaluable data point in prioritizing our efforts for supporting a new framework.

- On a very similar line, we have XQUERY script which will gives us no of documents across all databases, where a particular namespace is being used.
  The script is as under:

```
(: external binding parameter, to be entered as "n" variable :)
declare variable $n as xs:string external;
declare option output:item-separator "&#xa;"; (: Each element would be in ▶
new line :)

let $singlequote := "'"

(: Dynamic XPATH query, for $n namespace :)
let $cmd :=
           concat("/Archives/*/descendant::class[not(starts-with(@name,",
           $singlequote,$n,$singlequote,"))]/▶
descendant::*[starts-with(@name,",
           $singlequote,$n,$singlequote,")]")

let $apiPath :=
for $db in db:list()
let $query :=
  "declare variable $db external; " ||
  "db:open($db)" || $cmd
(: Execute above XPATH on every document in all databases.
This query returns matching nodes. :)
return xquery:eval($query,
        map { 'db': $db, 'query': $cmd })

let $clients :=
for $elem in $apiPath
return db:path($elem) (: $clients holds file name of all documents with ▶
above nodes:)

(: Return count of no of distinct file names, with matching nodes.:)
return concat($n,"      ",count(distinct-values($clients)))
```

Above script takes a namespace (for e.g. javax.servlet.http) as input and gives no of documents across all databases, where a hit was found, giving us exact no of customers using a particular namespace. This helps us collect prevalence data for each namespace which essentially means for each framework. For e.g. We can find out if Struts is more predominant over Spring MVC and so on and so forth.

5. Statistics

We ran a couple of experiments, with different approaches tried and using BaseX ecosystem discussed above on different hardware configurations, using same query. Below is the observation:

Table 1. Statistics of mining data using different approaches and hardware configurations

Hardware Specifications	Mining Approach	Database size	Amount of time taken	ResultSet
MacBook Pro, 16 GB RAM, 2.7 GHz Intel Core i7, SSD	Python Automator script	43 GB	1 Hour, 45 mins	NA
MacBook Pro, 16 GB RAM, 2.7 GHz Intel Core i7, SSD	BaseX eco-system thru REST	43 GB	< 6 mins	1.3 million records
Amazon EC2 instance, 32 GB RAM	BaseX eco-system thru REST	43 GB	< 5 mins	1.3 million records
Amazon EC2 instance, 32 GB RAM	BaseX eco-system thru REST	154 GB	21 mins	5.3 million records
Amazon EC2 instance, 32 GB RAM	BaseX eco-system thru REST	310 GB	42 minutes	11 million records
Amazon EC2 instance, 32 GB RAM	BaseX eco-system thru REST	450 GB	64 minutes	16 million records

We were psyched with the possibility of this sheer size of data being processed within an hour.

6. Conclusions

Our ability to collect data from various data sources, has significantly out-paced our capability to process, analyze, store or do any kind of predictive analysis on these datasets. Using right tool for data mining from proliferation of tools is exceedingly important, to explore the unbounded potential of BigData.

Parts of our BigData could be best represented in XML structure. This paper advocates Native XML Databases, as one of the most pragmatic option for XML data storing and mining for our use cases, due to its inherent nature of being optimized for storing and retrieving XML structured data. In this work, we talk in detail about our concrete experiences of how using NXD we could build our security analytic solution for predictive analysis, incidence response, framework security research etc.

We also discuss, our journey of finding the right approach and lessons learned along the way. From the huge taxonomy of BigData mining tools and technologies, we realized most of these were treating XML structured data as either flat streams of data or needed to be canonicalized for alternative representations like SQL or json, and not being exclusively optimized for XML structures. We sincerely hope this paper helps others researching approaches of data mining and information analysis, specially if BigData is represented in XML data stores.

None of the work described here would have been possible without the support and encouragement of Veracode management and specially security research team members here at Veracode. Many thanks to BaseX User group, specially to Christian Grun for his constant and timely guidance. I would also like to thank XMLPrague committee for the invitation to write and present this paper.

Bibliography

[1] BaseX Official Web Site http://basex.org/home

[2] XQUERY Priscilla Walmsley O'Reilly Media

[3] Efficient processing of large and complex XML documents in Hadoop Sujoe Bose Hadoop Summit 2013

[4] List of NoSQL Databases: http://nosql-database.org

[5] Data base ranking: http://db-engines.com/en/ranking

[6] Mining Big Data: Current Status, and forecast to the Future Wei Fan Albert Bifet http://www.sigkdd.org/sites/default/files/issues/14-2-2012-12/V14-02-01-Fan.pdf

[7] Scaling Big Data Mining Infrastructure: The Twitter Experience Jimmy Lin Dmitriy Ryaboy http://lintool.github.io/MapReduce-course-2013s/material/Lin_Ryaboy_2012.pdf

[8] Lecture 40 - XML Databases Dr.S.Srinath https://www.youtube.com/watch?v=GhvZMspVCbI

[9] Using Map and Reduce for Querying Distributed XML Data Lukas Lewandowski http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/Lewandowski12.pdf

Node search preceding node construction: XQuery inviting non-XML technologies

Hans-Jürgen Rennau
Traveltainment GmbH
`<hrennau@yahoo.de>`

Abstract

We propose an approach how to complement XPath navigation with a node search which does not require node construction. Node search is based on a set of external properties (a "p-face") which a node may assume in the context of a node collection. Being external, these properties can be retrieved without node construction, and being stored outside the nodes they can be maintained and queried by non-XML technologies, e.g. relational and NOSQL databases. A small set of concepts, carefully aligned with the XQuery data model, allows the seamless integration of various non-XML technologies driving node selection, without introducing any dependencies of XQuery code on any particular technology. A first implementation of the concepts is presented.

Keywords: XML, XPath, XQuery, XML infospace

1. Introduction

XPath navigation is based on node properties (document-uri, node-name, parent, children, attributes, typed-value, …). If the documents have not already been translated into an internal representation (e.g. by loading them into an XML database), access to node properties requires the parsing of document text into node trees, which is a very expensive operation. The consequence is a severe scaling problem, as you quickly run out of available memory and available time. Navigation requires node construction, and there is no general concept of a *search* as an operation to be distinguished from navigation and possibly preceding node construction.

The W3C data models of XML (infoset [2], XDM [7]) do not recognize entities "above" the single document. True, XPath and XQuery ([4], [5]) take into account the existence of collections, which are sequences of nodes. A collection may have a URI or be assigned the status of being the "default collection", but is has no properties apart from that.

If we recognize collections as entities in their own right, we can use them as *context* within which to associate member nodes with additional, collection-scoped properties. For example, documents representing logged events might be associated with a property “timestamp”. A collection might thus associate its member nodes with additional properties in the same way as a map associates its entries with a map key. And such additional properties serve the same purpose as map keys: to enable a selection among the contained units which does not require inspection of the units themselves. As the additional node properties are conferred by – and scoped to - the containing collection, let us call them *external properties*. Each external property has a name and a value. The value may echo document contents literally (be a copy of the value found in a particular attribute or element), be derived from document contents (e.g. counts and existence flags), or be independent of contents (e.g. the timestamp of document receipt). Excepting the last case, a property value can be viewed as the value of an XQuery expression evaluated in the context of the node in question.

External properties are stored outside the node itself, and property access does not require node construction. The external properties which a collection confers to its member nodes can thus be used to filter the collection without constructing any member nodes. This setup means a potential gain in efficiency, as in some scenarios the amount of node construction may be reduced drastically. Consider this document:

```
<resources>
  <r uri="file:/a/b/c1" t="2014-02-05T08:16:48" status="green"/>
  <r uri="file:/a/b/c2" t="2014-02-06T13:09:03" status="red"/>
  <r uri="file:/a/b/c3" t="2014-02-06T13:11:51" status="green"/>
  …
</resources>
```

It can be interpreted as the definition of a collection which associates XML documents – identified by their URI – with two external properties named `t` and `status`. The first might be the document creation time, and the second some kind of processing status. The creation time has perhaps been extracted from a particular location within the document, and the status has perhaps been determined by applying an XQuery expression to document contents. If an intended processing concerns only those resources whose creation time is after 2014-02-05 and whose status is “red”, a very simple processing of the <resources> element (`$rs`) will yield the required input, without accessing any resources not needed:

```
$rs/r[@t ge "2014-02-06"][@status eq "red"]/@uri/doc(.)
```

The example used an XML document to represent a collection conferring external properties to its member nodes. If we constrain external properties to be name/value pairs and values to be atomic sequences, other technologies may be used as well, e.g. relational or NOSQL databases. Such technologies may be leveraged when filtering the node collection in order to restrict node construction to those members actually required by the processing.

This paper explores the concept of a node search based on external node properties. The goal is a new XPath function offering node search, which is generic and yet enables the hidden integration of non-XML technologies into the operation of node search. We report a first implementation of the new concepts which is part of an open source framework for the development of XQuery command-line tools [6].

2. Basic concepts

First we introduce several basic concepts found useful in the design of a generic node search API which can be implemented using non-XML technologies.

2.1. external property

A mapping of a node to a property name and a property value. The name is constrained to be an NCName, the value is a sequence of atomic items, which may be empty. The mapping is defined in the context of a particular node collection. Different collections can associate the same node with different sets of external properties.

2.2. p-face

All external properties of a given node, conferred to it by a given node collection. Within a p-face, property names are unique. A p-face can therefore also be viewed as a collection-scoped mapping of names (= property names) to atomic sequences (= property values).

2.3. p-model

A specification how the nodes of a collection are mapped to their p-faces. A possible form is a set of property names associated with XQuery expressions, implying that the property value is the atomized result of evaluating the expression in the context of the node in question. Another form is a single XQuery expression resulting in a map of property names and values. A third form declines the possibility to derive the p-face from node contents and stipulates that a p-face is set from without, added to the node when it is inserted into the collection. The first two forms may also allow *additional* properties with unpredictable values and possibly also unpredictable names.

Note: a more formal definition of the p-model might be achieved by introducing the concept of a node insertion function and representing the p-model as a function argument.

2.4. node descriptor

A string from which a node can be constructed. A node descriptor may be the serialized node content, a URI or some other, proprietary kind of pointer providing node access.

2.5. p-node

Entity of information containing a node descriptor and a p-face.

2.6. p-collection

A sequence of p-nodes. By implication, also a sorted mapping of nodes to p-faces.

2.7. p-test

A condition imposed on an external property, defined in terms of (1) a property name, (2) a test value, (3) an operator relating property value and test value (e.g. =, <, >, …). A test value is a sequence of one or more atomic items. Example of a p-test, represented as text:

```
creationDate > 2013-12-31
```

2.8. p-filter

A condition imposed on a p-face, defined in terms of one or more p-tests and logical operators "and", "or", "not". Example of a p-face, represented as text:

```
creationDate > 2013-12-31 && status = green
```

2.9. node search

The operation of applying a p-filter to a p-collection. The result consists of those collection members whose p-face matches the p-filter.

3. Node search API

The new search API which we propose consists of a single XPath function `filteredCollection`. The input identifies a p-collection and specifies a p-filter. The output is the sequence of nodes contained by the p-collection and matching the p-filter. The p-collection is identified by a URI, and the p-filter can be supplied either as a structured representation (a `<pfilter>` element), or as an equivalent filter descriptor string. Informally, the function can be presented with two different signatures:

```
node()* filteredCollection( $pcollection as xs:anyURI,
                            $pfilter as element(pc:pfilter)?)
node()* filteredCollection( $pcollection as xs:anyURI,
                            $pfilter as xs:string?)
```

Formally, there is only one signature whose second parameter is typed as `item()?`.

The structured representation is an element tree composed of elements representing p-tests (`<p>`) and logical operations (`<and>`, `<or>`, `<not>`). Example:

```
<pfilter xmlns="http://www.infospace.org/pcollection">
   <and>
      <p name="status" value="green,closed" sep=";"/>
      <or>
         <p name="date" op=">" value="2013-12-31"/>
         <p name="supplier" op="~" value="X*"/>
      </or>
   </and>
</pfilter>
```

Besides the obvious operators (=, !=, <, <=, >, >=) there is a further operator (~) which represents pattern matching. If a test value comprises several items, they can be alternatively represented as a concatenated string (accompanied by attribute @sep which specifies the item separator) or by placing each item in an `<item>` child element of the `<p>` element.

Every filter element can be represented by an equivalent descriptor string composed of name/operator/value triples and logical operators (`&&`, `||`, `not`). Parentheses are used for controlling operator precedence and for delimiting a sequence of comma-separated test value items. Example:

```
status = (green,closed) && (date > 2013-12-31 || supplier ~ X*)
```

It is important to note that the function signature is generic: it avoids any dependency on the technologies involved in the internal representation of the collection, the filtering operation and the construction of selected nodes. The nodes may be constructed by parsing files, but they may also be obtained by parsing strings retrieved from a database (relational or NOSQL), or perhaps they are retrieved as nodes from an XML database. Likewise, the association between nodes and their external properties may be represented by XML structures, by relational tables, by NOSQL documents or yet in different ways. It is the function implementation which must handle the involved artifacts appropriately.

4. Node management API

The new node search API requires the definition and maintenance of p-collections. Key operations are:

- the creation of p-collections (instantiating artifacts)

- the loading of nodes into these collections (including their external properties)
- the removal of nodes from these collections
- the deletion of p-collections

Like node search, node management should be supported by a generic API which hides any implementation details and in particular the technologies actually used. We postpone the formal definition of this API to a later section, as it requires some further conceptualizations which we shall introduce first.

5. To bridge the gap

The concept of p-faced collections is based on the abstract notion of name/value pairs and their association with nodes, avoiding any assumptions about the implementation – the artifacts used to represent these pairs and associations. While it is not difficult to design a particular implementation, this would amount to an application-level solution only, because we still lack concepts how to integrate alternative implementations into a single and coherent model. We face a certain gap between the abstract notion of p-faced collections and any concrete implementation approach, a lack of terms which would allow us to speak about those implementations in a unified way. To bridge this gap, we propose three further concepts: *NCAT* (node catalog), denoting the artifact(s) which represent the collection contents; *NCAT model*, the information required to manage and use those artifacts; and *NODL* document (Node cOllection Description Language), a machine-readable description of a p-collection, including its p-model and its NCAT model.

5.1. NCAT

The artifact (or set of artifacts) used to represent a p-faced collection is called an NCAT (node catalog). Examples of an NCAT are an XML file, a set of relational database tables and a NOSQL database.

Software implementing the filtering of a p-collection must access the NCAT and apply to it operations of selection and retrieval. This presupposes information about the NCAT. Such information can be conceptualized as the NCAT model.

5.2. NCAT model

A p-collection has a logical structure which can be summarized as a sequence of entries containing a node and its external properties. An NCAT model defines a pattern how to map this logical structure to a technology-dependent representation (an NCAT). The model also specifies the configuration data required in order to manage and filter the NCAT. An example of such configuration data might be the connection data of a relational database which harbours the NCAT. It should also be noted that the model may define dependencies on the p-model, that is, the names and types of external properties used within the p-collection. An example would be a dependency of the number and names of relational tables to be used, and of the names and types of their columns.

5.3. NODL

Software tasked with filtering or managing a p-collection deals with its NCAT and must be supplied with information about it. The information should be received in a standardized and machine-readable form, which is a NODL document (Node cOllection Description Language). A NODL specifies the NCAT model and supplies any configuration data which it defines. A NODL also includes a description of the p-model, as details of the NCAT may depend on the number, names and types of external properties.

Note. The term NODL was chosen in order to stress the conceptual analogy to WSDL and WADL.

6. NODL documents

A NODL has the following general structure:

```
<nodl xmlns="http://www.infospace.org/pcollection">
   <collection>…
   <pface>…
   <nodeDescriptor>…
   <ncat>…
</nodl>
```

6.1. The `<collection>` element

Attributes of the `<collection>` element supply general information about the p-collection:

- @name - a collection name
- @uri - a collection URI
- @format - a list of formats in which collection members may be delivered
- @doc - a short description

6.2. The `<pface>` element

The `<pface>` element supplies the p-model. Each `<property>` child element defines an external property in terms of a name, a data type including cardinality, an optional length constraint and an XQuery expression yielding the property value. Some examples:

```
      <property name="cr" type="xs:dateTime"
                expr="ShoppingCart/@CreationDate"/>
      <property name="bookingID" type="xs:string*" maxLength="40"
                expr="//BookingId"/>
      <property name="travellerCount" type="xs:integer"
                expr="//Travellers/count(Traveller)"/>
```

The `<pface>` element can also have an `<anyProperty>` child element. When present, it signals that nodes may have additional external properties with unpredictable names and values.

6.3. The `<nodeDescriptor>` element

The `<nodeDescriptor>` element specifies how the NCAT represents collection nodes – as URI, as serialized node text, or as a non-URI accessor string.

6.4. The `<ncat>` element

The `<ncat>` element identifies the NCAT model and provides its configuration data. The element has a single child element whose name and contents depend on the NCAT model used. Currently, two NCAT models are supported, so that the `<ncat>` element contains either an `<xmlNcat>` or a `<sqlNcat>` element. The contents of these elements are described in the following section.

7. First NCAT models

This section summarizes first versions of two NCAT models, defining the structure of NCATs based on XML documents and relational databases, respectively.

7.1. The XML NCAT model

The model defines the structure of an NCAT implemented as an XML document. The model is best illustrated by a little example of such an NCAT:

```
   <pnodes xmlns="http://www.infospace.org/pcollection" name="scarts"
         uri="" formats="xml" nodeDescriptor="uri" count="32048">
      <pnode node_uri="..." t="2014-11-13T12:29:56" sID="02Z9ROU5">
         <tourOp>AB</tourOp>
         <check>
            <item>Address</item>
            <item>DoubleBook</item>
            <item>WatchList</item>
         </check>
      </pnode>
      <pnode node_uri="..." ...>...</pnode>
```

```
    <pnode node_uri="..." ...>...</pnode>
    ...
</pnodes>
```

The NCAT is implemented by a `<pnodes>` element containing `<pnode>` elements which represent the collection nodes. Every `<pnode>` element has a @node_uri attribute providing a URI which can be resolved to the node in question. The external properties are represented by further attributes and/or child elements with names equal to the property name. The first entry of the NCAT shown above thus represents a node which has four external properties: `t`, `sID`, `toupOp` and `check`. A single-valued property is represented by either an attribute or a child element with simple content. A multiple-valued property is represented by a child element which has one `<item>` child per value item.

Note that a program processing such an NCAT cannot predict whether to find the property values in attributes or elements. This does not mean any ambiguity, as property names are unique within each p-face. Given a p-node $pnode, the following expression retrieves the value of property `foo` reliably:

```
$pnode/(@foo, foo[not(item)], foo/item)
```

If the filtering of p-nodes resulted in a sequence $pnodes of selected p-nodes, the following expression returns the selected nodes themselves:

```
$pnodes/@node_uri/doc(resolve-uri(., base-uri(..)))
```

7.1.1. Configuration data

Configuration data specify the document URI of the NCAT document and, optionally, the names of properties which should always be stored as elements, rather than attributes. By default, single-valued properties are stored as attributes.

7.1.2. NODL representation

When the XML NCAT model is used, it is represented by an `<xmlNcat>` element with empty content, a mandatory @documentURI attribute and an optional @asElems attribute. The latter contains a whitespace-separated list of property names or name patterns. Example:

```
<xmlNcat documentURI="/ncats/xsds.ncat" asElems="*remark* airport"/>
```

Note. The current version of the XML NCAT model does not support any other representation of the node than by URI. This restriction does not apply to the SQL NCAT model, because it can make much sense to store serialized documents within the NCAT itself if it *is* a database.

7.2. The SQL NCAT model

Disclaimer
The model as described and implemented has only been used with MySQL databases, version 5.6. An extension enabling its use with other database systems is planned.

The NCAT is implemented by a set of tables harboured by a single database. Given a p-model, the following tables are used (where ${cname} denotes the collection name found at `$nodl/collection/@name)`:

${cname}_ncat	The main table, containing the node descriptor (e.g. node URI) and all single-valued properties.
${cname}_ncat_${pname}	One such table is used for every property $pname whose cardinality allows multiple values.
${cname}_ncat___dyn	This table is only used if the p-model allows properties with arbitrary names and values (thus if `$nodl/pface/anyProperty` exists).

The main table ${cname}_ncat contains the following columns:

nkey	an integer primary key identifying the node
node_uri	the node URI (only if `$nodl/nodeDescriptor/@kind` = 'uri')
node_text	the serialized node (only if `$nodl/nodeDescriptor/@kind` = 'text')
node_access	the node accessor (only if `$nodl/nodeDescriptor/@kind` = 'accessor')
${pname}	for every property $pname whose cardinality precludes multiple values: the property value

If one of the columns `node_uri` or `node_access` exists, it is associated with an index enforcing unique values. Every column $pname (which represents a single-valued property) is associated with an index whose length is the minimum of the length constraint found in the p-model `($nodl/pface/property[@name eq $pname]/@maxLength)` and the value 200.

Every table ${cname}_ncat_${pname} represents a multiple-valued property $pname. It has these columns:

nkey	a foreign key referencing the `nkey` column in the main table
pkey	an integer primary key

${pname}	the value of an item of property $pname

The column $pname is associated with an index whose length is the minimum of the length constraint found in the p-model and the value 200.

The table ${cname}_ncat__dyn is used only if the p-model admits arbitrary property names. It has the following columns:

nkey	a foreign key referencing the `nkey` column in the main table
pkey	an integer primary key
pname	a property name
pvalue	a property value item

The columns `pname` and `pvalue` are both associated with an index whose length is 100 and 200, respectively.

7.2.1. Configuration data

Configuration data specify the RDBMS, the database connection (host, user name, password) and the database name.

7.2.2. NODL representation

When the SQL NCAT model is used, it is represented by an `<sqlNcat>` element with empty content and the following attributes: @rdbms, @rdbmsVersion, @host, @user, @password, @db. Example:

```
<sqlNcat rdbms="MySQL" rdbmsVersion="5.6" host="localhost"
       user="guest" password="guest123" db="pcol"/>
```

8. First implementation

This section summarizes a first implementation of the concepts introduced in this paper. The implementation is part of `ttools` [6], an open source framework for the development of XQuery command-line tools.

Like the framework as a whole, the implementation of p-collections is pure XQuery – no changes of the XQuery processor code were made. Support for the SQL NCAT model currently requires use of the BaseX processor [1], as access to relational databases is implemented using the BaseX extension functions `sql:connect` and `sql:execute`. Support for other processors offering equivalent extension functions will be added shortly.

8.1. Node search API

Applications developed with `ttools` have access to the function `filteredCollection` discussed in section Node search API. Example code:

```
tt:filteredCollection($nodl, "airport=(DUS,CGN) && lowcost=true")
```

Assuming that $nodl is a `<nodl>` element node, the function call returns all nodes found in the collection described by $nodl and which have within this collection an external property `airport` with a value equal "DUS" or "CGN" and also an external property `lowcost` with a value `true`. If $nodl uses an XML NCAT model, both of the following entries would contribute nodes to the result:

```
<pnode node_uri="…" lowcost="true" airport="DUS"/>
<pnode node_uri="…" lowcost="true">
   <airport>
      <item>DUS</item>
      <item>FRA</item>
   </airport>
</pnode>
```

If $nodl uses an SQL NCAT model, the matching entries would be determined by a SQL select command with a `where` clause referencing a `lowcost` and an `airport` column. The details of the command text depend on the collection name and on the cardinalities of the external properties (implied by `$nodl/pface/property/@type`). Assuming that the collection name is `offers`, `lowcost` is single-valued and `airport` is multiple-valued, the following command would be used behind the scenes:

```
SELECT node_uri FROM offers_ncat AS t1
LEFT JOIN offers_ncat_airport AS t2 ON t1.nkey = t2.nkey
WHERE `lowcost` = 'true' and `airport` in ('DUS', 'CGN')
```

8.2. Input parameter type `docSEARCH`

Using the `ttools` framework, application code does not receive external variables as supplied by the invocation, but the results of a pre-processing which the framework applies to the input. A `ttools` based application has one single external variable (named `request`), which represents the tool invocation and which is parsed into an operation name and a set of named input parameters. Here comes an example of a `request` value, invoking an operation called `typeReport` and supplying two input parameters, `comp` and `xsds`:

```
typeReport?comp=ctype, xsds=/projects/xsd/*.xsd
```

The pre-processing is controlled by annotations contained by the XQuery modules, which associate parameter names with a type and possibly further information. Application code retrieves a parameter by passing its name to a `getParam` function, and it receives a value constructed from the input text (a substring of the external variable `request`) in accordance with the type annotation. Given the following annotations:

```
<operation name="typeReport">
   <param name="xsds" type="docDFD"/>
   <param name="comp" type="xs:string">
</operation>
```

the application could retrieve the parameter values like this:

```
let $comp as xs:string := tt:getParam($request, "comp")
let $xsds as document-node()* := tt:getParam($request, "xsds")
```

Note that the delivered value of parameter `xsds` is a sequence of document nodes, whereas the supplied value consists of a file name pattern. The transformation of supplied parameter text to delivered parameter value is controlled by the type annotation (`docDFD`).

`ttools`' support for p-collections is reflected by the parameter type `docSEARCH` which is available for use in parameter annotations. Supplied parameter values must consist of a URI and a p-filter, separated by a question mark:

```
nodlURI?pfilter
```

The delivered value is the sequence of document nodes obtained by applying the p-filter to the p-collection defined by the NODL document found at the specified URI. For example, given these annotations:

```
<operation name="typeReport">
   <param name="xsds" type="docSEARCH"
   <param name="comp" type="xs:string"/>
</operation>
```

and the following invocation:

```
typeReport?comp=ctype, xsds=/nodls/xsds.nodl?ctype~*lang*
```

the application code would receive the filtered p-collection implied by the parameter value:

```
let $xsds as document-node()* := tt:getParam($request, "xsds")
```

Note that the application code does not even call `tt:filteredCollection` – the parameter value is *delivered* as the filtered collection specified, to know: the sequence of all documents which are contained by the p-collection defined by the NODL found at `/nodls/xsds.nodl` and which have an external property `ctype` with a value containing the string "lang".

8.3. Node management API

Applications developed with ttools have access to a node management API consisting of the following functions:

```
 tt:createNcat ($nodl as element(pc:nodl))
 tt:feedNcat   ($nodl as element(pc:nodl), $nodes as node()*)
 tt:feedNcat   ($nodl as element(pc:nodl), $dirFilter as xs:string+)
tt:copyNcat   ($nodl as element(pc:nodl), $pfilter as element(pc:pfilter)?,
               $toNodl as element(pc:nodl)
 tt:deleteNcat ($nodl as element(pc:nodl))
```

Preconditions for their use are:

1. The application was created using a flavour parameter set to basex79 or basex80
2. The supplied NODL document uses either an XML NCAT model or an SQL NCAT model
3. If the NODL uses an SQL NCAT model:
   a. the RDBMS MySQL is installed (version >= 5.6)
   b. the official JDBC driver for MySQL (MySQL Connector/J) has been downloaded and placed in the lib directory of the BaseX installation

Function tt:feedNcat can be supplied with *directory filters*, which are strings with a ttools-defined syntax. Using directory filters, one may specify complex filters on file system contents in a very concise way, specifying directories which are searched recursively or shallowly, positive and/or negative file name patterns and optional constraints controlling the inclusion/exclusion of subdirectories encountered during recursive search.

9. A small sample application

This section describes the creation of a small sample application which illustrates the use of p-collections. Let us assume we want to create a command-line tool which offers various reports about XSD documents specified by the command-line call. Our application will support the possibility to specify the input XSDs as a filtered p-collection. The actual reporting is kept trivial to avoid distraction from the main point, which is the use of p-collections.

After downloading the ttools framework from [6], we use it to create a command-line application named xspy. Its first version shall have a single module of application code (items.mod.xq) and support a single operation tns. The operation will report the target namespaces found in a set of XSDs.

Having installed ttools in a directory /ttools, we want to create an initial version of our new application in a directory /apps/xspy. We achieve this with the following call:

```
basex -b "request=new?dir=/apps/xspy,mod=items,ops=tns, flavor=basex79"
   /ttools/ttools.xq
```

The call creates the application directory and fills it with XQuery modules supplied by the framework, as well as a generated first version of the application module `items.mod.xq`.

What we shall do now is:

1. create a p-collection containing XSDs, which are associated with a useful set of external properties
2. write some application code producing a simple report on target namespaces
3. invoke the application, supplying it with a filtered p-collection

In order to create a p-collection containing XSDs, we first create a NODL document similar to the following:

```
<nodl xmlns="http://www.infospace.org/pcollection">
   <collection name="xsds" uri="" formats="xml" doc="A collection of XSDs."/>
   <pface>
     <property name="tns" type="xs:string?" maxLength="100"
               expr="//xs:schema/@targetNamespace"/>
     <property name="stype" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:simpleType[@name]/@name"/>
     <property name="ctype" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:complexType[@name]/@name"/>
     <property name="elem" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:element/@name"/>
     <property name="att" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:attribute/@name"/>
     <property name="group" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:group/@name"/>
     <property name="agroup" type="xs:string*" maxLength="100"
               expr="//xs:schema/xs:attributeGroup/@name"/>
     <property name="enum" type="xs:string*" maxLength="100"
               expr="//xs:enumeration/@value"/>
     <anyProperty/>
   </pface>
   <nodeDescriptor kind="uri"/>
   <ncat>
      <sqlNcat rdbms="MySQL" rdbmsVersion="5.6" host="localhost"
               user="guest" password="guest123" db="pcol"/>
   </ncat>
</nodl>
```

The p-collection will thus support the following external properties:

- tns – the target namespace
- stype – the names of contained simple type definitions
- ctype – the names of contained complex type definitions

- elem – the names of contained global element declarations
- att – the names of contained global attribute declarations
- group – the names of contained model group definitions
- agroup – the names of contained attribute group definitions
- enum – the values of contained enumeration values

Due to the `<sqlNcat>` element, the NCAT will be created in the form of relational tables inserted into a database `pcol`. Note that the table names are implied by the collection name (`xsds`) and the names of those external properties which may be multiple-valued (including all properties whose type has an occurrence indicator '*' or '+') .

The NCAT is created by the following builtin operation of our new application:

```
basex -b request=_createNcat?nodl=/nodls/xsds.nodl /apps/xspy/xspy.xq
```

Now we fill the NCAT with all XSDs found in or under the directory `/xsds/niem-2.1`:

```
basex -b "request=_feedNcat?nodl=/nodls/xsds.nodl,
          dirs=|/xsds/niem-2.1/*.xsd" /apps/xspy/xspy.xq
```

While the feed proceeds, we may observe how the MySQL tables are filled:

```
mysql -uguest -pguest123
mysql>use pcol
mysql>select node_uri from xsds_ncat;
=>
+---------------------------------------------------------------+
| node_uri                                                      |
+---------------------------------------------------------------+
| file:/xsds/niem-2.1/ansi-nist/2.0/ansi-nist.xsd              |
| file:/xsds/niem-2.1/ansi_d20/2.0/ansi_d20.xsd                |
| file:/xsds/niem-2.1/apco/2.1/apco.xsd                        |
...
+---------------------------------------------------------------+

mysql> select stype from xsds_ncat_stype;
=>
+--------------------------------------------------------+
| stype                                                  |
+--------------------------------------------------------+
| AccidentSeverityCodeSimpleType                         |
| AlarmAcknowledgementCodeSimpleType                     |
| AlarmDescriptionCodeSimpleType                         |
| AlarmEventCategoryCodeSimpleType                       |
| AlarmEventLocationCategoryCodeSimpleType               |
...
+--------------------------------------------------------+
```

Now we write some application code. We edit the generated function `f:tns` so that it returns a simple target namespace report: listing all target namespaces encountered in the input XSDs, and for each namespace the document URIs of all XSDs using it. We replace the generated dummy version of function `f:tns` by the following code:

```
declare function f:tns($request as element()) as element() {
    let $docs := tt:getParams($request, 'fdocs')
    let $tns :=
        for $xsd in $docs
        group by $tns := $xsd/xs:schema/@targetNamespace
        let $uris :=
            for $uri in $xsd/document-uri(.)
            order by lower-case($uri)
            return <xsd uri="{$uri}"/>
        order by lower-case($tns)
        return <tns uri="{$tns}">{$uris}</tns>
    return
        <z:tns countDocs="{count($docs)}">{$tns}</z:tns>
};
```

Note that the filtered XSDs are obtained simply by retrieving the input parameter `fdocs`:

```
let $docs := tt:getParams($request, 'fdocs')
```

This is explained by the fact that the generated annotations (which we neither changed not removed) declare the operation to have a parameter `fdocs` of type `docSEARCH`:

```
<operation name="tns" type="node()" func="tns">
   <param name="fdocs" type="docSEARCH*" sep="SC" pgroup="input"/>
   ...
```

The remainder of the function body creates a list of target namespaces found in the filtered collection. After these changes, the call

```
basex -b request=tns?fdocs=/nodls/xsds.nodl?stype~*country*
   /apps/xspy/xspy.xq
```

produces a report of all target namespaces containing simple type definitions with a name matching `*country*`:

```
<z:tns xmlns:z="http://www.ttools/org/sql/ns/xquery-functions" countDocs="5">
  <tns uri="http://niem.gov/niem/domains/jxdm/4.1">
    <xsd uri="file:///C:/xsds/niem-2.1/domains/jxdm/4.1/jxdm.xsd"/>
  </tns>
  <tns uri="http://niem.gov/niem/domains/screening/2.1">
    <xsd uri="file:///C:/xsds/niem-2.1/domains/screening/2.1/screening.xsd"/>
  </tns>
```

```
    <tns uri="http://niem.gov/niem/fips_10-4/2.0">
      <xsd uri="file:///C:/xsds/niem-2.1/fips_10-4/2.0/fips_10-4.xsd"/>
    </tns>
    <tns uri="http://niem.gov/niem/iso_3166/2.0">
      <xsd uri="file:///C:/xsds/niem-2.1/iso_3166/2.0/iso_3166.xsd"/>
    </tns>
    <tns uri="urn:oasis:names:tc:ciq:xal:3">
      <xsd uri="file:///C:/xsds/niem-2.1/external/have/1.0/xAL-types.xsd"/>
    </tns>
  </z:tns>
```

Note that to produce this report, only the five documents which are reported are parsed. Behind the scenes, their selection is accomplished by the following SQL query:

```
SELECT node_uri FROM xsds_ncat AS t1
   LEFT JOIN xsds_ncat_stype AS t2 ON t1.nkey = t2.nkey
   WHERE `stype` LIKE '%country%'
```

Also note that calling the application, we specified the p-collection by supplying its NODL. The NODL *encapsulates all implementation details*, of which we need not be aware in order to use the collection. To complete this little tutorial, we create a copy of the NODL (`xsds2.nodl`), in which we replace the `<sqlNcat>` element by an `<xmlNcat>` element:

```
<ncat>
   <xmlNcat documentURI="/ncats/xsds.ncat"/>
</ncat>
```

We create and populate the XML based NCAT exactly as we created the SQL based NCAT:

```
basex -b "request=_createNcat?nodl=/nodls/xsds2.nodl" /apps/xspy/xspy.xq
basex -b "request=_feedNcat?nodl=/nodls/xsds2.nodl,
          dirs=|xsds/niem-2.1/*.xsd" /apps/xspy/xspy.xq
```

Using the new NODL instead of the previous one, we obtain the same results, although now behind the scenes the documents are selected by evaluating the XML based NCAT.

10. Discussion

This work attempted to prove the concept of p-collections as a way of complementing XPath navigation with a node search which does not require node construction. In many scenarios, the use of XML databases seems to be a better alternative, mainly as the stored documents can be selected in a “deep” way which may respond to anything found anywhere within the documents, and in an “open” way as there is no limit to the possibilities of defining conditions. These are in fact the very strengths of XPath and XQuery, and in comparison the filtering of p-collections appears somewhat primitive, limited to the "flat" set of external properties and to the combination of atomic property tests via the logical operations of `and`, `or` and `not`.

Nevertheless, we believe that p-collections bear some promise. Major advantages are:

- non-XML technologies can be leveraged (and all their power, maturity and scalability)
- the possibilities of adding further technologies are not limited
- technology choices are on the level of single collections – for any collection the best fit can be used
- XQuery code using p-collections is independent of technology choices
- p-collections are simple to manage thanks to a generic node management API

The use of p-collections thus allows something which could be called "distributed technologies" and which, in spite of its severe limitations, may seriously compete, in specific scenarios, with the "monolithic" alternative of working with a particular XML database system, no matter how powerful.

The current version of the concepts constrains the names of external properties to be NCNames. We refrained from admitting QNames, rather than NCNames, as they would introduce a certain increase of complexity - how to represent them elegantly in non-XML resources and queries? But should the step yet be taken, external properties could be RDF triples, and the potential of this perspective remains to be explored.

Bibliography

[1] BaseX: XML database and XQuery processor. http://basex.org

[2] Cowan, John & Tobin R., eds. XML Information Set (Second Edition). W3C Recommendation 4 February 2004. http://www.w3.org/TR/xml-infoset/

[3] Rennau, Hans-Jürgen. "XQuery topic tools - concept, user interface, development framework." Presented at Balisage: The Markup Conference 2014, Washington, DC, August 5 - 8, 2014. In Proceedings of Balisage: The Markup Conference 2014. Balisage Series on Markup Technologies, vol. 13 (2014). doi: 10.4242/BalisageVol13.Rennau01.http://www.balisage.net/Proceedings/vol8/html/Rennau01/BalisageVol8-Rennau01.html.

[4] Robie, Jonathan et al, eds. XML Path Language (XPath) 3.0 W3C Recommendation 8 April 2014. http://www.w3.org/TR/xpath-30/

[5] Robie, Jonathan et al, eds. XQuery 3.0: An XML Query Language. W3C Recommendation 8 April 2014. http://www.w3.org/TR/xquery-30/

[6] Topic tools - a lightweight framework for developing powerful command-line tools with XQuery. https://github.com/hrennau/TopicTools

[7] Walsh, Norman et al, eds. XQuery and XPath Data Model (3.0). W3C Recommendation 8 April 2014. http://www.w3.org/TR/xpath-datamodel/

Native XML Databases: Death or Coming of Age?

Craig Brown
<cmb@qti.qualcomm.com>
Xavier Franc
<xfranc@qti.qualcomm.com>
Michael Paddon
<mwp@qti.qualcomm.com>

Abstract

Back in the early 2000s Native XML Databases (NXDbs) promised a bright future. Today, adoption falls short of those promises and some have even suggested that the technology is in decline. We analyze this "trough of disillusionment" with reference to the Gartner Hype Cycle and compare it to the history of relational databases.

Despite the slow maturation of NXDbs, XML itself is now well established and here to stay. The need for repository systems able to store and query large document collections is likely to increase as ever more XML data is generated. XML is an important technology in the field of publishing and digital preservation, and is especially effective when it comes to handling a mix of text and data.

Native XML Databases might represent only a niche market, but may prove to be irreplaceable in some applications.

To illustrate this, we present a use case in which an NXDb turned out to be a good solution for a massive data warehouse problem. Why other solutions fell short is discussed, as well as the specific features that made the product we selected, the Qualcomm® Qizx™ database[1]*, suitable. To conclude, we provide our thoughts about which features are desirable or even possibly indispensable in an NXDb from the perspective of making native XML Databases more attractive to application designers.*

Keywords: XML, database, query optimization, scalability

[1]Qualcomm Qizx is a product of Qualcomm Technologies, Inc. Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Qizx is a trademark of Qualcomm Incorporated. All trademarks of Qualcomm Incorporated are used with permission.

1. Where are all the XML Databases?

Native XML Databases (NXDbs) were a hot topic 10-15 years ago. Many believe the buzz peaked around 2003-2004. The use case seemed as obvious then as now: “if you have more than a handful of XML documents that you need to store, you should store them in a native XML Database” [7]. And yet as of 2014, we have not seen mainstream adoption of the technology.

W3C standards development has continued slowly and steadily, however, and important standards such as XQuery Update [2] were only completed in the past 5 years.

The number of published research papers in the NXDb field appears to have steadily decreased since the last decade. However research into non-relational databases continues to bloom, and many of the fundamental results are directly applicable to NXDbs. This suggests that the term “NoSQL” may have displaced “NXDb” in grant applications but that current research continues to be relevant to XML Databases.

The number of available NXDb products reached a peak of about 40 at the end of the last decade, and some are no longer maintained or commercially available [7]. Bourret's website has listed no new products over the preceding 5 years. An informal survey by the authors suggests there are currently a few actively developed open-source products and a handful of commercial offerings.

RDBMS vendors have quickly reacted to the threat of NXDbs by offering XML extensions in their products, typically as an "XML column" feature. This has blurred the distinction between relational and XML Databases, and created uncertainty in the market about the benefits of purely native XML technologies. To date, this has been effective at defending the market share of incumbent products against potential disruption.

The very expression "XML Database" has nearly disappeared from marketing literature, in favor of *"NoSQL"*. The ongoing investment in the NoSQL field points to continuing and growing demand for non-relational data models (key/value, document-oriented, graph) and yet XML Databases do not seem to benefit from this trend.

A perception exists that NXDbs lack maturity for use by enterprises. There is a kernel of truth to this: relational database products are decades old and contain many more features than the first generation of XML Databases. Of course, they also contain much more legacy technical debt. As XML offerings mature, more enterprise-ready features may be expected.

There is no well-known theory about how to model data in a non-relational way. Some relational database architects shy away from XML to represent data in new applications since there are few good examples to emulate. The presence of hierarchy and the lack of traditional schema normalization also can seem unsettling. This is a chicken and egg problem that we last saw in the 1980s. One of the authors recalls when developers were wary of the newfangled relational databases versus tried and trusted raw ISAM files.

XML standards are often regarded as too complex. Some enterprises and developers are slow to embrace radically new technologies. Adoption of XQuery [1] is a good example. As a functional language, it is relatively difficult for a programmer from an imperative background to pick up. Nevertheless, there are benefits to be realized once that learning curve has been climbed. For instance, the natural composability of XQuery functions allows the easy creation of domain specific query languages in a way that might not be obvious to an imperative programmer.

JSON has taken much of the focus away from XML when it comes to representing semi-structured data. JSON is perceived as much lighter and easier to master, especially in web environments. However, XML already has many of the components that enterprises will demand long term such as standardized schema/query/update/processing languages, which JSON still lacks.

2. Are we in the "Trough of Disillusionment"?

According to the Gartner Hype Cycle [6], after reaching the “Peak of Inflated Expectations,” the visibility of new technologies frequently goes through a “Trough of Disillusionment” before possibly scaling the “Slope of Enlightenment” or maturation phase.

The history of relational databases is instructive: the first paper was written in 1969, lots of research and investment occurred through the 1970s; finally credible products emerged, starting around the early 1980s. The market went on to mature, and by the early 1990s many early entrants had retired their products [8]. The survivors of this shakeout enjoyed strong growth from the mid 1990s onwards. Looked at through this lens, XML databases are still very young. If the past is a guide, one would expect credible products to be emerging sometime this decade. XML Databases might only now be starting up Gartner's slope of enlightenment. If that is the case, then one would expect a future phase of mass adoption.

Obviously the NXDb situation is not exactly the same, since RDBMS had no incumbent to challenge at the time, but this observation confirms that the actual pace of adoption of a new technology can be much slower than initially imagined by its proponents.

This begs the question: why would anyone choose an NXDb over the alternatives? Some key indicators that an NXDb may be the most suitable platform are:

- Where XML is used as representation in the first place.
- Data where the schema is significantly evolving over time.
- Information merged from multiple, disparate sources (with schema evolution being common here, too).
- Complex hierarchical relationships in data.

The authors encountered a use case exhibiting all of these characteristics. We present it as an example of the sorts of problems that we believe will drive NXDb adoption in the future, and recount our experiences in building a solution.

3. Use Case: a World Patent Database

It is common for an inventor (or the company they work for) to apply for patents to protect the fruits of their research.

An invention may only be patented if (amongst other criteria) it is novel. Anyone applying for a patent is therefore likely, at some point, to consider searching through existing patents (and related publications) to see if prior art exists. But there are millions of active patents nowadays, and without effective tools that task can become extremely time-consuming and costly.

This is a quintessential big data problem. There is an enormous amount of information available in a vast number of documents published by patent offices around the world. Some is textual. Some is structural. While simplistic key word searches are applicable to this data, modern machine learning techniques (e.g. clustering and classification) are also valuable. Our use case essentially consists of a massive document warehouse that must be queried in many different ways, not all of which are foreseeable. Some of these queries are made by humans looking for a small result set. Others are made by machines and may return extraordinary amounts of data.

Patents (and related documents such as published filings) are extremely hierarchical, and exhibit an evolving schema. XML is a natural format for their representation, transmission and storage, and we were not surprised to discover this was indeed common practice. Figure 1 depicts some structure from a sample document, from which a sense of the schema may be inferred. Since representation is a solved problem, our challenge reduces to effectively storing, correctly indexing and efficiently querying a huge corpus of complex XML documents.

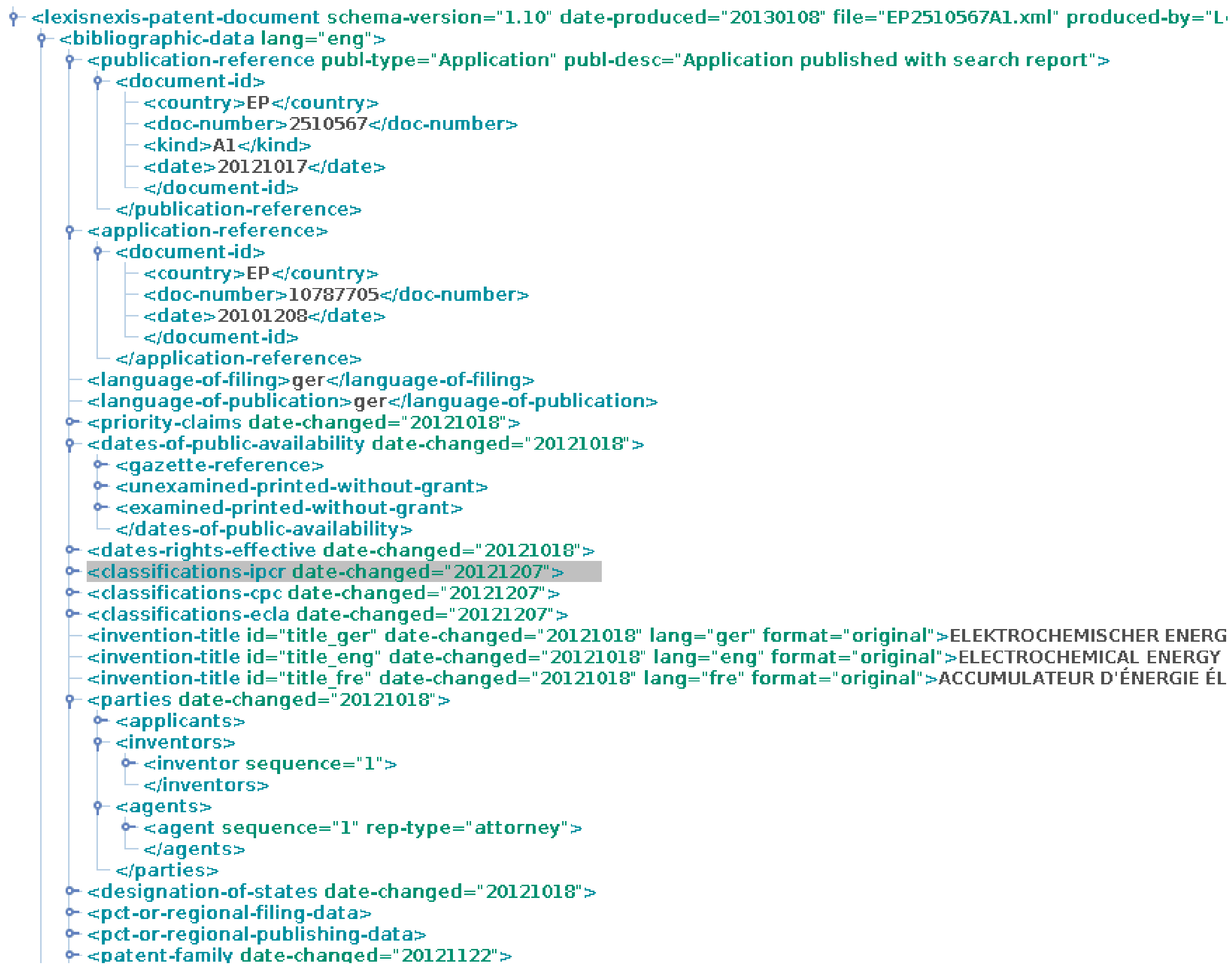
How large? If one takes all the published patents and filings from just three patent offices (USPTO, EPO and WIPO/PCT) there are nearly 20 million documents. That's over 2.5 terabytes of data (about 125 kilobytes per document on average) and growing fast. Over 850,000 new or updated documents are issued per month.

How big is 2.5 terabytes? Merely to copy from hard disk to hard disk (at circa 100 Mb/s) takes about 7 hours; In our tests, parsing that much XML in order to extract fields took days on a contemporary CPU.

All patent documents use a common schema in principle, but there are variations according to time and originating office. This schema defines about 200 elements and 50 attributes and includes MathML. From time to time new elements are added. Any solution must deal gracefully with this change, including in updates to old documents.

The types of queries that must be supported for our use case are:

- Large batch queries for statistics and machine learning.
- Small queries generated by interactive user interfaces (with many concurrent users).
- Custom ad hoc queries for data exploration. Such queries can be complex with multiple conditions and constraints.



```
<lexisnexis-patent-document schema-version="1.10" date-produced="20130108" file="EP2510567A1.xml" produced-by="L
  <bibliographic-data lang="eng">
    <publication-reference publ-type="Application" publ-desc="Application published with search report">
      <document-id>
        <country>EP</country>
        <doc-number>2510567</doc-number>
        <kind>A1</kind>
        <date>20121017</date>
        </document-id>
      </publication-reference>
    <application-reference>
      <document-id>
        <country>EP</country>
        <doc-number>10787705</doc-number>
        <date>20101208</date>
        </document-id>
      </application-reference>
    <language-of-filing>ger</language-of-filing>
    <language-of-publication>ger</language-of-publication>
    <priority-claims date-changed="20121018">
    <dates-of-public-availability date-changed="20121018">
      <gazette-reference>
      <unexamined-printed-without-grant>
      <examined-printed-without-grant>
      </dates-of-public-availability>
    <dates-rights-effective date-changed="20121018">
    <classifications-ipcr date-changed="20121207">
    <classifications-cpc date-changed="20121207">
    <classifications-ecla date-changed="20121207">
    <invention-title id="title_ger" date-changed="20121018" lang="ger" format="original">ELEKTROCHEMISCHER ENERG
    <invention-title id="title_eng" date-changed="20121018" lang="eng" format="original">ELECTROCHEMICAL ENERGY
    <invention-title id="title_fre" date-changed="20121018" lang="fre" format="original">ACCUMULATEUR D'ÉNERGIE ÉL
    <parties date-changed="20121018">
      <applicants>
      <inventors>
        <inventor sequence="1">
        </inventors>
      <agents>
        <agent sequence="1" rep-type="attorney">
        </agents>
      </parties>
    <designation-of-states date-changed="20121018">
    <pct-or-regional-filing-data>
    <pct-or-regional-publishing-data>
    <patent-family date-changed="20121122">
```

Figure 1. A patent document sample (styled view with some nodes folded)

4. The Journey to an NXDb

In order to find a solution for our use case, we embarked on a journey in which we experimented with many different database technologies. It was this process that eventually led us to NXDbs, and in particular Qizx, as a viable solution.

4.1. Key/Value stores: the joins killed us

Key/Value stores basically offer to retrieve some application-defined data (value) from a key. It is often also possible to scan keys matching a particular condition. Some KVS allow retrieval by content, using secondary indexes.

There are quite a few key/value storage systems available. From the perspective of the difficulties we encountered, they are all essentially similar as the root cause is an "impedance mismatch" between the primitives and the problem to be solved.

- The simplest approach is that the key is a field of interest inside a document and the value is a reference (DocId) to a document. We discuss in the next section what it takes to extract and maintain 'fields of interest'.
- In all cases, it is necessary to create key/value tables ahead to time to support all possible queries. As a consequence custom queries are impractical on large data sets.
- Due to rudimentary query primitives, most joins must be done by business logic: this is time consuming to implement, error prone, and inefficient (much data has to be read and conveyed from storage to client).
- Most joins or constraint checks (e.g structure based) require pulling and parsing entire documents: this is terribly slow. Extracting specific parts of documents is also cumbersome.
- No free text search: necessity to add a third-party full-text engine.
- Some sort of structure index could have been implemented as key/values. Then we would have built a kind of native XML database!

4.2. Relational Databases: schema maintenance killed us

Using a Relational DBMS seemed a viable approach, building on a mature technology and query language. Mapping XML nodes to a relational schema requires data to be decomposed into tables and fields. This process is generally called *shredding*.

Some databases can automatically perform the decomposition task, based on a XML schema:

- The problem here is to have an up-to-date schema.
- Automatic shredding is notoriously slow.
- The resulting relational schema is quite complex. Writing queries against such a complex schema is really hard.
- We could just have used an "XML column" in a relational database. In that case however, indexing of our content would have been limited or non-existent, and querying would be impractical beyond a point.

Alternatively, it is possible to define a partial relational schema: this means defining entities (or types) of interest in the XML document (e.g. patent, inventor, claim), and for each entity, defining named properties as a function of the contents.

- Defining a complete relational schema is an enormous challenge, that requires hundreds of tables and thousands of constraints.
- Without automatic shredding, custom code is required to parse documents, extract and index fields: this is expensive and error prone.
- There is always a new type or field that has not been seen yet: that induces a Sisyphean task of continual rewrites of database schema, parse/store software with days/weeks of reloading.

4.3. Full-text search engine with non-text fields

Many full-text search engines offer the possibility of handling several "Fields" which can even have non-text value types like numbers or dates. A typical example is Lucene. An example of this approach applied to patents is Google Patents[2]. It offers a limited search interface with links providing a kind of drill-down functionality.

As with relational databases, each document has to be decomposed into a set of fields.

- Each change in the field-set implies rescanning all the documents and re-indexing, a task usually requiring several days of processing.
- It is unwieldy or impossible to deal with entities in the document that do not decompose cleanly into flat fields.
- We lose the ability to leverage hierarchy in any but the most basic ways. Often this is very important. We might want to query all the names in a patent filing. Tomorrow, we might want just the inventors' names. The relative position of elements carries important information.

4.4. Other non-XML systems

We also considered a NoSQL document-oriented database such as MongoDB, as it can handle semi-structured data in the form of JSON.

However there was a major issue: JSON cannot represent XML with mixed content.

4.5. Native XML Databases

Storing XML documents directly as XML, being able to retrieve the documents in their original form, then querying this data using a powerful and standardized

[2] http://patent.google.com

query language (XQuery), seemed a natural and promising way to solve our problem. But the devil was in the details.

To determine which NXDbs were suitable to our problem, we ran a set of trials.

Open-source products were examined initially, since by nature they have evaluation versions without limitations.

The problems we encountered fell in several categories:

- **Stability**: several products terminated with software crashes or data corruption issues under intense upload strain. Durability was fundamental to us since it can take weeks to import really large datasets.
- **Scalability**: we encountered two classes of problem as the database grew large:
  - Update performance dropped off severely. A slow decay in speed is naturally expected (typically `1/log(n)`) since many algorithms involved have a `n×log(n)` complexity at best. But in some trials we observed such a rapid fall in performance that it became clear that we would be unable to update documents as fast as they were being produced.
  - Some internal limit was reached: for example in one trial the database could not store more than 4 billion nodes (2^{32}).
- **Query speed**: this was always going to be a major challenge for a multi-terabyte document corpus. We observed that many products became unable to return queries fast enough for our needs when our collection grew beyond a few million documents. In some cases, this may have been improved by defining a number of custom indexes - but that would have created an ongoing issue of future ad hoc queries requiring new indexes.

Our trials led us to conclude that, at that time, none of the open source candidates were a good fit for our use case. Different products had different pros and cons, but none met our exact needs. We note that many of these products have continued to rapidly mature in the 2-3 years since our study. The reader is encouraged to run their own trials for their own use cases.

We then began to assess commercial offerings. A key requirement for us was to avoid proprietary query languages and consequential vendor lock-in.

4.6. Finally, Qizx

Qizx was one of the commercial offerings we chose to trial. Qizx is an NXDb with several interesting features that was initially developed by a small European company, and has since been acquired by Qualcomm Technologies, Inc.

Qizx is a lightweight product, in the sense that it has a focused feature set, is parsimonious with resources and runs fast. It has good documentation, tools to make evaluation easy, and transparent licensing. In addition it has the following particular characteristics:

- Stability and Scalability: Qizx is able to run for more than one week ingesting 2 terabytes of XML.
- **Automatic indexing**: There is no need to define ad hoc indexes in order for new queries to be executed efficiently. Qizx creates all necessary indexes and uses them transparently. We found that this feature is a game-changer.
- Fast, standard queries: Without requiring any particular effort, except writing good XQuery, Qizx has excellent response times against large collections. It supports XQuery Full-Text and XQuery Update.
- Application-specific key-value properties (basic data or XML fragments) can be attached to documents, and used for querying. This support for metadata makes it possible to perform pre-computation on documents and store the results without touching the original data.
- Data type conversion: During the storing/indexing phase, Qizx automatically recognizes basic types such as numbers and dates in attribute or element content, and indexes them both in text form and in the recognized type. For example an element such as `<item price='1.0'/>` can be found by a query such as `//item[@price = 1]` (numeric value) as quickly as `//item[@price = '1.0']` (text value).

Qizx turned out to be an excellent fit for our use case.

5. Reflections

Our journey provided us first-hand experience as to why a native XML database can be a superior solution for handling a massive XML corpus with complex structure. We also discovered just how difficult it was to find the right technology for our needs despite there being quite a few options to choose from. That might explain why some give up and write off native XML solutions! That nearly happened to us, but we are pleased that we persisted.

Along the way, we learned a lot about what might make an NXDb more attractive to an application architect.

Automatic indexing

The problem with most non-XML technologies is that they require re-scanning of the entire dataset in order to extract the new fields or to build new specialized indexes, a process that can take days of computation in our case. This is not only very costly, it can be Sisyphean at development time. It can also imply database downtime if such changes are required during the production phase.

Storing in XML form provides the benefit that all data is available for query and that no data is lost in the translation phase. But this is not sufficient: If new specialized indexes have to be defined because of new queries then some of the benefit is lost, because adding new indexes also requires a re-indexing phase where all the data present in the database has to be scanned. The cost is likely smaller than rescanning data from source XML, but it is still significant.

Therefore it appears that the feature 'automatic indexing' is fundamental: it means "store and index" once, then execute all the queries needed without any other overhead. If in addition the queries run fast, then the benefit over other solutions is tremendous.

Actually there are already several NXDbs that support such a feature in some way. In our opinion, all native XML Databases should support some form of this feature, because 1) XML makes it possible and 2) without this feature, a native XML Database is much less useful.

Use of XML Schema (or not)

XML Schema can be used in several ways:

- For validating documents on creation or update. This is obviously a desirable feature.
- For determining the data type of element or attribute content. But this is a rather heavy approach for users. Requiring a schema in order to determine the types of pieces of XML data is also somewhat at odds with the "self-describing" promise of XML.
- To optimize queries. This idea has been advocated since the beginnings of XQuery. We believe that it is of limited effectiveness:
  - It is highly complex, and indeed it has never been implemented as far as we know.
  - It requires a correct association between document and schema, without version issues. This is error prone and expensive to manage for database users.

  In contrast, optimization methods that use the knowledge of the *actual* structure of documents, inferred from the data and without need for a schema have been proposed [4][5].

  The next version of Qizx will implement this type of optimization and deliver significant speedups. Our preliminary testing suggests a 2 to 10 times speedup in typical cases.

Fast queries

XML Databases should be optimized for fast querying, not for massive updating, because they are likely to be used for applications where data does not change much but rather is accumulated (historical data, publications, logs).

Ideally, an XML Database supporting automatic indexing should be able to compile and optimize any standard XQuery using the automatic indexes and then return results at optimal speed.

In practice, this is not always possible. Some queries cannot be optimized. In addition XQuery is a rich and complex language that makes optimization difficult.

For example, a direct exploitation of automatic indexes in Qizx allows optimizing a query such as:

```
//inventor[ .//name = 'John Smith' ]
```

But if the inventor name in the database has not the exact specified form, the query will fail. So we can rewrite it like this:

```
//inventor[ normalized-name(.) = 'john smith' ]
```

where normalized-name() is some function that returns a normalized form of the name of an inventor. It can be as simple as lowercase() or more complex (the inventor name can also appear as a pair of elements first-name, last-name).

Unfortunately, the query compiler in Qizx currently cannot fully optimize this query using indexes only. Therefore it must back off on the predicate and optimize only `//inventor`, which is much slower on a large database.

It is interesting to notice that automatic indexes offer great possibilities of optimizations:

- For example it is possible to apply the function to all index keys and find those that match the normalized name, then merge the corresponding results. It can be costly, but if it is done once and cached in a map, then subsequent queries will execute much faster.
- By building specialized indexes derived from basic indexes, at much lower cost.

Full-text search

Because it is the nature of XML to mix text and data, it seems very natural and useful to offer full-text searches on XML documents.

Full-text capabilities have been standardized in XQuery some years after the core language. The XQuery Full-Text standard [3] offers a very rich set of features, but strangely lacks an essential functionality: how to quickly get the N most relevant documents or nodes for a given full-text query (provided by any full-text search engine).

This can be expressed in XQuery:

```
subsequence( for $hit score $score in ...full-text-query...
             order by $score descending
             return $hit,
             1, N)
```

Yet this is not only embarrassingly cumbersome, but also quite difficult to optimize by a database system.

The only solution for XML Database implementers is to resort to proprietary functions. It seems very desirable to define a standard function that would do the same thing in a simple way.

Faceted search

Another capability that is very useful for real applications but has no support yet in XQuery standards is *faceted search*.

There are many definitions of faceted search (and a myriad patents around it...), but here is an outline of this feature, applicable to XML Databases:

1. A faceted search returns an *Entity*, which is typically an XML element; for example an entity would be `patent, inventor, claim, legal-event`...
2. For each Entity type, a set of *Facets* is defined. Facets are properties of the entity that can be used either for searching or for sorting.

   Each facet has a name and is computed from the Entity, for example as a relative path, but more generally by any function.

   The value can be transformed for presentation to a search user: for example dates can be mapped to a set of months or years.
3. A search specifies an Entity type and a subset of facets with values (or more generally with predicates on values). Other facets can be used for sorting the results.

   Search can also be combined with full-text search and sorted by relevance.
4. For each facet not used in the search, the search function computes the distinct values of that facet over the result set with occurrence counts.

   The facet values can be presented to a user in order to "drill down" by using particular facet values to refine the search.

Standardizing faceted search for XML represents a fairly significant step forward that we believe the XML community can collaborate to achieve.

6. Conclusions

In conclusion, we believe that there is a strong argument to make that NXDbs have been languishing in the Gartner “Trough of Disillusionment” for some time. However relevant research has continued and the surviving products have continued to mature. In the experience of the authors, NXDbs now outperform competing technologies when used in their areas of strength.

Our trials suggest that products which support automatic indexing and effective query optimization are the most likely to be attractive to application architects. Simple and efficient full-text search and faceted search are also highly desirable.

Products with these features will arguably be well placed to take advantage of future market growth.

Bibliography

[1] XQuery 3.0: An XML Query Language. W3C Recommendation, 08 April 2014. http://www.w3.org/TR/xquery-30/

[2] XQuery Update Facility 1.0. W3C Recommendation, 17 March 2011. http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/

[3] XQuery and XPath Full Text 1.0. W3C Recommendation, 17 March 2011. http://www.w3.org/TR/2011/REC-xpath-full-text-10-20110317/

[4] DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Roy Goldman , Jennifer Widom - Stanford University 1997

[5] Faster path indexes for search in XML data Nils Grimsmo, Norwegian University of Science and Technology, Trondheim. 2007 - ISBN: 978-1-920682-56-9

[6] Gartner Hype Cycle. http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp

[7] XML and Databases, R.Bourret. http://www.rpbourret.com/index.htm

[8] Timeline of Database History. http://quickbase.intuit.com/articles/timeline-of-database-history

A Unified Approach to Design and Implement data-centric and document-centric XML Web Applications

Christine Vanoirbeek
<christine.vanoirbeek@epfl.ch>
Houda Chabbi Drissi
<Houda.Chabbi@hefr.ch>
Stéphane Sire
<s.sire@oppidoc.fr>

Abstract

The paper addresses the development of XML-based web applications that indifferently deal with structured document-centric or data-centric information. It focuses on a fundamental aspect – the validation of information – that provides substantial benefits at two levels: (i) the enhancement for the developers to design and implement error-free applications and (ii) the capacity for the end users to provide information that meets the requirements of the underlying application information model. The paper proposes an approach based on RELAX NG and Schematron. It also describes an implementation of the validation process in the eXist-db database environment using the Oppidum XQuery development framework.

Keywords: XML Web Application, Document-centric and Data-centric information, Information validation

1. Introduction

Document-centric and data-centric applications are still mostly considered in opposition; we argue in favor of a uniform « reconciling » XML representation of both types of information to build Web-based information management applications.

Modeling document or data with an XML approach raises a number of questions that are inherent to the nature of information conveyed by those two types of information and the way to design appropriate models that take into consideration their respective specificities. Despite the apparent opposition, we believe that it exists a lot of applications where data and document-oriented XML information are intrinsically tied and can undoubtedly take mutual benefit from a uniform modeling approach.

For historical reasons, *document-centric information* often refers to structured documents (potentially constrained by a DTD as initially promoted by SGML), i.e. a structured content produced by an author in the purpose of publication. It also generally evokes a single piece of information that has a paper counterpart. In this sense, a so-called XML document-centric content exhibits a number of specific features.

The features are essentially the following ones:

- The order of elements is important;
- The structure is quite irregular and results in XML trees that are not balanced;
- The granularity is potentially very high (as for examples, a paragraph that takes into account the structure of a bibliographic reference or the insertion of hyperlinks);
- Mixed content is frequently used.

However, the perception of a structured document has changed over the time. Initially designed for publishing purposes, structured documents are increasingly considered as pieces of data within global information systems. XML schemas are progressively used to define document models; they introduce new concepts such as data types that do not exist in DTDs. Such documents are designed and produced in a way that facilitates automatic processing operations on them.

Data-centric information refers to XML content most often generated from relational databases and produced by applications. It commonly represents data collections with complex structures that mainly content non textual data. The adoption of XML to deal with data has been essentially motivated by the stack of technologies brought by the document markup approach to facilitate exchange of information or to produce views on XML data flows by mean of XSLT (a typical example being the production of HTML content or PDF documents) but also to query and manipulate XML data flows using XQuery.

Data-centric information presents the main following features:

- The order of elements is not important;
- The XML trees are quite balanced with repetitive structures;
- No use of mixed content.

Nowadays, more and more complex Web applications deal with both data-centric and document-centric information, such as enterprise intranet solutions. This category of applications relies on a strong modeling of information stored in databases in order to provide the expected functionality.

One way to address the development of such applications would be to use a relational model for the data-centric part and an XML model for the document-centric part. This seems, at a first glance, a good idea since new generations of relational databases (which implements the standard SQL2006) are XML-enabled. However, jointly using the two models, systematically raises a fundamental problem for the designer. How to guarantee the consistency between the potentially duplicated information in the two representations?

We argue in favor of using the XML model to encompass, in a unified way, the modeling of document-centric and data-centric information as a wide range of data structures, including those of the relational and object database models, can be described with the XML format [1].

The paper is organized as follows. First, the paper introduces a motivational example in the purpose of emphasizing the benefits that could be taken from an XML unifying approach. Second, it discusses modeling issues to tackle both document-centric and data-centric information. Third, it describes an approach based on the combined use of RELAX NG and Schematron. Forth, it presents the technical solution implemented in the eXistdb environment. Finally, the paper concludes with perspectives for future works.

2. Motivational example

As a typical example, let us consider an application dedicated to the management of research projects (such as European projects). During the preparation step, the partners of a project will work on the elaboration of the proposal i.e. the preparation of a document describing the project. They will also provide of a set of administrative data.

The description of the project contains purely document-centric information, such as the description of the project objectives, the classical description of the state of the art, the biography of key partners, etc.

The specification of workpackages of the project is more data-centric but especially well adapted to a hierarchical representation; each workpackage contains a title, a global description, a number of tasks, each of them potentially subdivided into sub-tasks and finally a list of deliverables. Each task consists in a description, a list of involved partners, a start and end time. The description of the workpackage itself is clearly document-centric information.

The set of administrative data is purely data-centric information. It consists in a list of records gathering information about organizations and individuals, such as addresses, contacts, emails, etc.

Adopting an XML approach to represent the purely document-centric information is simple common sense. The XML approach is particularly well adapted for publishing purposes or exploitation of information through dynamic views in a browser. For instance, once the project accepted, information about partners could be automatically made available on the website of the project.

Adopting an XML approach to address the representation of a workpackage makes sense for three main reasons: (i) its hierarchical structure is well adapted to XML (while possibly but less appropriately represented with a relational model), (ii) the modeling of textual parts is obvious and (iii) precisely structuring the content allows to generate derived views (such as a Gantt chart or a table summarizing the list of deliverables), avoiding information redundancy in the document.

Adopting a relational approach to represent administrative details is quite natural. No need of hierarchy as the information is flat and can be easily stored into relations. Using the foreign key, we can link all the information together.

The implementation of a an application such as a project management raises a number of questions in terms of the database to be used for storing, managing and querying the mentioned different kinds of information.

Three main options are available:

- The use of a relational database
- The use of an XML-enabled relational database
- The use of an XML native database (XND)

Use of a relational database

Many research, since 2001 [4], has been done around the techniques (of shredding or mapping) to be used to store XML documents into a relational database. Let us note, that the challenge is not only how to store the hierarchical structure, but also how to allow the execution of XQuery queries and XUpdate in an optimized way. The fact is, that each XML query has to be translated into a SQL query. The relational optimizers are not always aware about the XML structure and then produce poor schedule plans[7]. This is why the new generation of XML-enabled relational databases uses native XML storage and dedicated indexes have been introduced (as in oracle).

Use of an XML-enabled database

Using an XML-enabled relational database (implementing the SQL2006 standard) seems, at a first glance, a good idea. It allows to use a relational model for the data-centric part and an XML model for the document-centric part.

However, which mechanism should be used to maintain consistency between potentially duplicated instances of information? For instance, what would be the impact on the application development if the phone number of a key partner changes? An answer to this problem can be to duplicate the information and create triggers to maintain the consistency.

What about the normalization of such a relational schema? First normal form is no longer guaranteed as the XML type attributes are not atomic.

Whatever mechanism is used to address this problem, we can already note the following points:

- When querying these relational schemas, we use the SQL/XML. So, we can write a query which mix between a pure relational query (SQL) and a XQuery query. So we need to know the two languages. And let us note that even though SQL/XML is a standard, it is not used in all XML-enabled DBMS.

- The SQL/XML standard does not support the FTQuery part of the XQuery language, which is important in the applications that we address.

Use of an XML native database

Using an XND is the third option. We explain in the next section why such databases are fully adapted to represent both document-centric and data-centric information. XNDs use XML documents as unit of storage - of course not necessarily stored as text files - and often provide the concept of collection, a logical model for grouping and/or repeating documents. Moreover the XQuery language allows the development of complete web applications.

Our objective is to provide the developer of a Web application with the possibility to validate the entire database against a global XML model of its application. The goal is to facilitate - and accelerate - the development process by detecting harmful XQuery code. The decomposition of the data model of the application into collections and resources complicates somehow the matching against a global XML model, since there may be several decompositions for a given XML model. This is an issue we will address in the rest of the article.

3. Modeling issues

From a conceptual point of view, a number of issues are to be discussed about the potential offered by the main existing languages (XML Schema, RELAX NG and Schematron) to constrain XML content. DTDs are obviously discarded as they do not support data typing. The following considerations aim at justifying our proposal to use RELAX NG and Schematron [8].

3.1. Specification of content type

Content type is of major concern to jointly address processing of both document-centric and data-centric information. Regarding *atomic values* (the XML text nodes), it is important to potentially consider some of them as processable units of information. It seems obvious for data-centric information where atomic values are intended to store field-like information that will be processed by an application. But it is also true for enhancing processing of a document-centric content. A typical example could be the modeling of a cooking recipe; it is by nature a highly document-oriented content but typing information about the number of persons it concerns as well as the quantity of requested ingredients would allow, for instance, to automatically generate a list of ingredients to be bought accordingly to a varying number of persons.

Adopting the W3C XML Schema predefined datatypes is suitable for a wide variety of applications. Using RELAX NG is a reasonable choice to avoid dependance on a specific datatypes system as the requirements of applications may evolve over time and, maybe, necessitates the use of another datatypes system.

An important issue concerns *mixed content* that is often presented as a typical feature of document-centric information. Mixed content is a reminiscence of SGML and has been introduced to accommodate representation of structured document content at a period of time where either memory or storage occupation were more important than nowadays. For compatibility reasons, this concept has been maintained in XML. As a consequence, allowing character data to be interspersed with tagged elements unfortunately complicates the implementation of API provided to deal with XML content [3].

We promote to discard the use of mixed content to better capture the structure of XML content and to enhance its processing in the framework of Web applications that target a unified structural approach to avoid redundancy between document-centric and data-centric content.

As an example, let us consider the description of the objectives of a research project. It is clearly a document-centric element that contains, among other things, a number of paragraphs. We propose the following representation of a paragraph.

Example 1. The content model of a paragraph

```
<Parag>
  <Fragment>The project addresses </Fragment>
  <Fragment FragmentKind="important">the development
          of Web applications </Fragment>
  <Fragment>based on XML technologies.
         Information about standards is available on the </Fragment>
  <Link>
    <LinkText>W3C</LinkText>
    <LinkRef>http://www.w3.org/</LinkRef>
  </Link>
  <Fragment> site.</Fragment>
</Parag>
```

The content model of a paragraph does not include any pseudo-element; each of its sub-elements being limited to either a Fragment or a Link. A Fragment may optionally have a FragmentKind attribute that is used in rendering purposes. In our model, Fragment elements are of string type (and not token type) as the leading and trailing spaces should not be removed.

Our position about mixed content is of course tied to the authoring capabilities offered to the final users for providing XML content [9]. This issue is addressed in Section 5.

A potential rendering of the paragraph of Example 1 is illustrated in Figure 1.

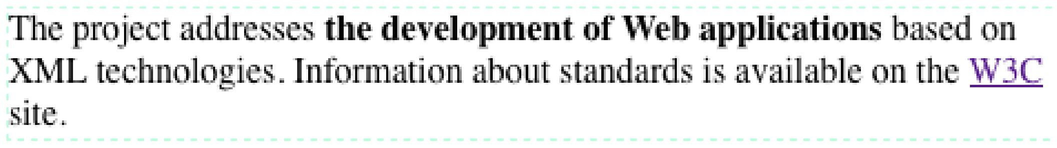

The project addresses **the development of Web applications** based on XML technologies. Information about standards is available on the W3C site.

Figure 1. Rendering of a paragraph in a browser

Specifying the definition of *compound elements* pleads in favor of RELAX NG for two main reasons: (i) the named pattern approach is much more flexible than XML Schema approach - it is important in terms of design - and (ii) the constructors proposed by RELAX NG are perfectly adapted to model either document-centric or data-centric content.

Document-centric content organization relies on the use of 3 main constructors (inherited from SGML): the sequence, the choice and the aggregate constructors. RELAX NG proposes the two first ones and the interleave constructor is more powerful that the aggregate constructor proposed in SGML.

Data-centric content relies on the extensive use of unordered content. Surprisingly, XML Schema offers very weak support in this respect. By default, ordering elements is the default option when designing a schema; the all constructor is very restrictive.

3.2. Constraining content

Both document-centric and data-centric approaches require the expression of constraints for guaranteeing that information is structured in such a way that it can be processed by an application, the goal being to provide the functionality satisfying at the same time editing, rendering and computing issues. Let us consider the categories of these constraints and how they stand in the two paradigms.

3.3. Mandatory information

Information may be mandatory in the document-centric world for two reasons: in a editorial purpose (as an example, a section without title does not make sense) or in a computing purpose (as an example, the absence of of number of persons in a cooking recipe will impact the safe use of a function that provides the list of ingredients to be bought). The content mandatory constraint is a well-known concept in the data-centric world.

Making elements mandatory is a point that may be addressed by RELAX NG as well as XML Schema but a related important issue is to guarantee that an element satisfies a content model that is conforming the expected processable format. This may be achieved by specifying constraints on the content, an issue that may be addressed by using, for instance, the potential offered by the W3C XML Schema datatypes. Once again, this issue is in strong relation with authoring capabilities offered to the final users but also to the designers of an application who need to be provided with appropriate means to express such constraints.

3.4. Uniqueness and referential integrity

In the document-centric world, references to unique elements were initially intended to express cross references in a document (such as references to a section in a document or reference to a figure). In this purpose, the SGML DTDs introduced the concept of ID and IDREF, based on the use of attributes to express such constraints. This concept has been retained by XML Schema that also extents the possibilities through the mechanism of key and keyref reflecting a more relational database oriented view. One of the the main difference is the scope of the identifier.

However key and keyref mechanism presents certain limitations that can be avoided by using the assertion mechanism introduced in XML Schema 1.1 [2]. Using assertions, keyref are essentially expressed with XPath expressions. It reinforces our approach which consists in specifying such constraints via Schematron rules.

3.5. Cardinality constraints

A conceptual model of data usually specifies cardinality constraints indicating how to limit the number of entity occurrences that are associated in a relationship. Using a relational database, cardinalities `0-1`, `1-1`, `1-N` and `N-N` can be directly represented by the paradigm of primary keys and foreign keys. But, all the other cardinalities can not be directly expressed by the relational structure.

Adopting the XML model as the native way to model information, the cardinality constraints may be expressed by the combined use of composition relations and specification of occurrences. XML schema - but not RELAX NG - allows to be more precise and to explicitly indicate minimum and maximum values for element occurrences. For instance, it could be used for limiting the number of topic keywords occurrence in a paper if is requested by a publisher.

Jointly using RELAX NG and Schematron provides the advantage to benefit from a more expressive way to specify composition of elements and to define precise cardinality constraints by mean of Schematron rules.

In summary, our approach consists in describing the data structure with RELAX NG - potentially associated with an external data types system, such as the XML Schema data types - and in expressing any other type of constraints with Schematron rules.

4. Modeling approach based on RELAX NG and Schematron

In this section, we present our approach to design the XML model of a Web application targeted to be implemented in a XND. Let us consider our motivational example, introduced at Section 2, to illustrate our approach on a concrete case.

4.1. Modeling the structure of a research proposal with RELAX NG

The graphical view of the RELAX NG schema shown on Figure 2 illustrates the global structure that could be adopted to model the information processed by a project management (PM) Web application. Our simplified PM application gathers (i) information about several projects, each of them having an `Id`, a `Name` and containing the description of the `Proposal` and (ii) information about `Partners`; a list of `Person` potentially involved in one or more projects.

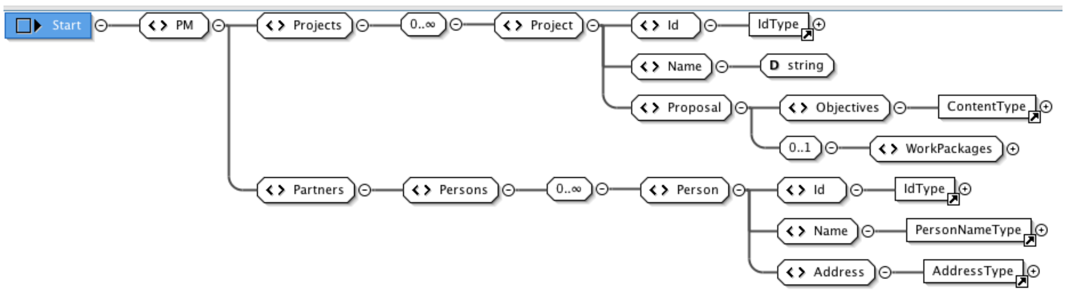


Figure 2. Global schema of the PM application

The description of the objectives of the project is purely document-centric and matches the model illustrated in Figure 3 that proposes a simple content model to represent textual data composed of either `Title`, `Paragraph` or `List` elements. The content model for a `Paragraph` is the one presented in Section 3.

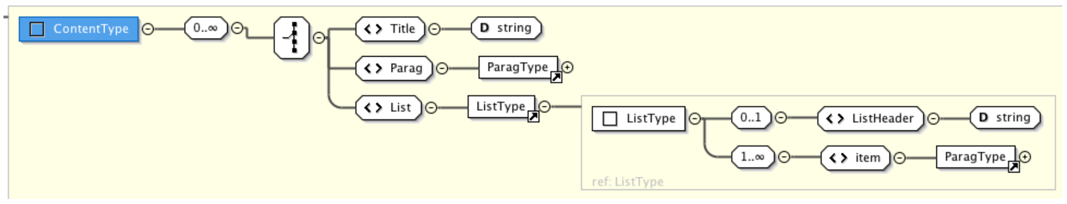


Figure 3. Content model of textual data

The model of a `WorkPackage` is illustrated in Figure 4. It emphasizes the fact that it contains interleaved relations between document-centric information and data-centric information.

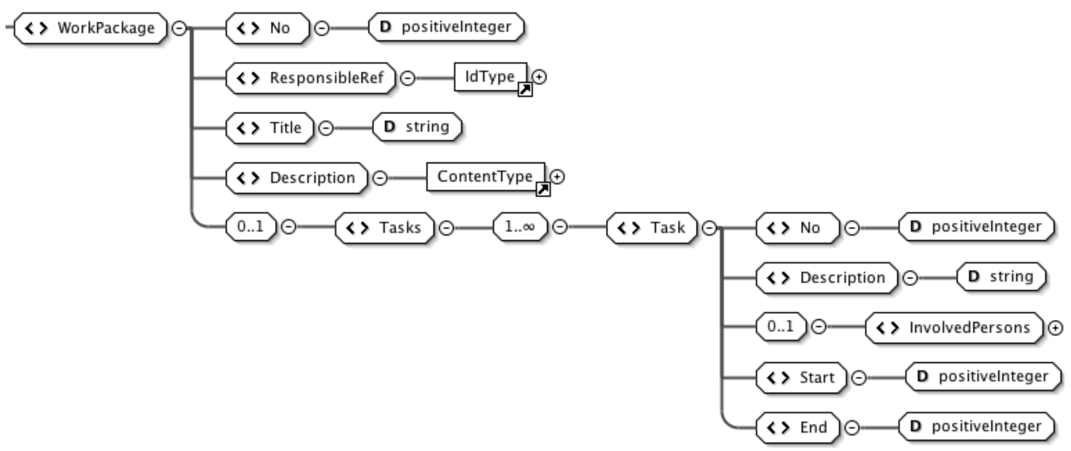


Figure 4. Content model of a workpackage

Finally, information about a `Person` is a purely data-centric information.

4.2. Annotation of the RELAX NG Schema

Annotating the RELAX NG Schema is twofold. It aims at providing the developer with the possibility to clearly distinguish between the design of the conceptual model of information to be processed by an application but also aims at providing specification about the physical layer.

In a XND, the physical layer mostly concerns the issue of grouping XML documents into collections; it is not independent from the conceptual model. Our approach consists in providing the developer with the possibility to annotate the global schema of its application - the conceptual model - by specifying the way to organize the information into collections and documents.

In this purpose, our proposal is to enhance the RELAX NG schema with annotations (making use of a dedicated namespace prefixed as `ide`) to specify which elements are to be stored as documents in the database. The extract of an annotated schema shown in Example 2 illustrates this mechanism. In that case the `collection-name` annotation tells that the developer's choice is to store information about each project individually in a specific XML document in a `partners` collection, while she wants to store information about all the persons in a single XML document called `persons.xml`.

Example 2. Global schema with annotations

```
<element name="Partners" ide:store="collection" ide:collection-name="partners">
  <element name="Persons" ide:store="resource" ide:resource-name="persons">
    <zeroOrMore>
      <element name="Person">
        <element name="Id">
```

```
          <ref name="IdType"/>
        </element>
        <element name="Name">
          <ref name="PersonNameType"/>
        </element>
        <element name="Address">
          <ref name="AddressType"/>
        </element>
      </element>
    </zeroOrMore>
  </element>
</element>
```

The annotations can be exploited to generate all the RELAX NG sub-schemas to be used to check an XML document against its appropriate sub-schema. This can be done, for instance, by using an XSLT transformation that we will describe in Section 5 which makes use of another `store` attribute annotation.

4.3. Specifying constraints with Schematron

The following Schematron rules provide examples of constraints that can be checked in the context of our example.

They reflect the unicity and referential constraints but also illustrate an application specific constraint; for example, the fact that managers of projects limits the involvement of persons in no more than 3 tasks.

Example 3. Specifying contraints with Schematron

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <pattern name="unicity-constraints">
    <rule context="PM">
      <assert test="count(distinct-values(.//Project/Id))
                 = count(.//Project/Id) ">Duplicate Ids for projects</assert>
    </rule>
    <rule context="WorkPackages">
      <assert test="count(distinct-values(WorkPackage/No))
                   = count(WorkPackage/No)">Duplicate No for workpackage</▶
assert>
    </rule>
    <rule context="Tasks">
      <assert test="count(distinct-values(Task/No))
                   = count(Task/No)">Duplicate Id for tasks</assert>
    </rule>
    <rule context="Persons">
      <assert test="count(distinct-values(Person/Id))
                   = count(Person/Id)">Duplicate Id for persons</assert>
    </rule>
  </pattern>
  <pattern name="referential-integrity">
    <rule context="WorkPackage">
      <assert test="ResponsibleRef = ancestor::PM/Persons/Person/Id">Reference ▶
to unknown ResponsibleRef</assert>
       <assert test="Tasks/Task/InvolvedPersons/InvolvedPersonRef
                    = ancestor::PM/Persons/Person/Id">Reference to unknown ▶
InvolvedPersonRef</assert>
    </rule>
  </pattern>
  <pattern name="application-specific-constraints">
    <rule context="Project">
      <assert test="count(distinct-values(.//Tasks/Task//InvolvedPersonRef)) ▶
&lt;=3">A person cannot be involved in more than 3 tasks</assert>
    </rule>
  </pattern>
</schema>
```

5. Implementation

This section describes an implementation of our approach in order to provide the developer with validation tools to check the data consistency of an application during the development step.

Our work relies on the use of Oppidum [6], a ligthweight MVC development framework that takes benefit of a full stack of XML technologies to query, manipulate and provide views in a browser of XML information stored in the eXistdb XML native database. The companions of Oppidum are AXEL and AXEL Forms [5], two JavaScript libraries that offer the users the capability to equally author, in a browser, document or data-oriented XML information that conform to a model.

5.1. Developing the validation mechanism

The Oppidum framework is intended to build Web applications relying on a RESTful approach. For that purpose the application must be entirely defined in terms of actions on resources. This is achieved by defining declaratively each action in a mapping file.

To experiment our mechanism, we implemented an action called `generate-all-schema`. It is defined in our mapping file as shown on Example 4. This code indicates, in the MVC paradigm, which model to use (the model is always an XQuery script) and which view applies (the view is an XSLT transformation working on the result of the XQuery script).

Example 4. Mapping entry for activating the generation of sub-schema

```
<action name="generate-all-schema">
  <model src="modules/ide/generate-all-schema.xql"/>
  <view src="modules/ide/display.xsl"/>
</action>
```

The `generate-all-schema` XQuery script shown as Example 4 generates all the requested sub-schemas and stores them in the database for further use by the developer. It uses conventional locations in the database, for instance to be integrated in an IDE to propose easy access to validation services while developing and testing the application. It relies on annotations included in the global schema of the application. The input and output collection paths are encoded in variables in the `global` namespace. The generation of sub-schemas is achieved by invoking an XSLT transformation shown as Example 6. That transformation generates one sub-schema for each element annotated as `ide:store="resource"` in the global schema. The set of definitions of the RELAX NG schema are factorized in a resource called `definitions.rng`.

Example 5. XQuery script to generate and store sub-schemas from the global schema of the application

```
declare function local:gen-definitions-for-writing ( $grammar as element() ) {
  <grammar xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="http:/▶
/www.w3.org/2001/XMLSchema-datatypes">
    { for $d in $grammar//define return $d }
  </grammar>
};

declare function local:store-grammars ( $grammars as element() ) {
  <Root xmlns="http://relaxng.org/ns/structure/1.0">
    {
    for $g in $grammars/grammar return
    let $filename := concat(string ($g/@name), '.rng')
    let $schema :=
      <grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
        <include href="definitions.rng"/>
        { $g/start }
      </grammar>
    return
      xmldb:store($globals:schema-collection, $filename, $schema)
    }
  </Root>
};

let $global-schema := fn:doc($globals:global-schema-resource)
let $global-transfo := fn:doc($globals:generate-all-schema-transfo)
let $all-schemas := transform:transform ($global-schema, $global-transfo, ())
return (
  xmldb:store($globals:schema-collection, "definitions.rng", ▶
local:gen-definitions-for-writing($all-schemas//Definitions),
  local:store-grammars($all-schemas//Grammars)
  )
```

By convention, the global schema of an application is stored in a sub-collection named `schema`, this collection being itself stored in a collection named `validation`. The generated sub-schemas are stored in the `schema` collection. The Schematron rules are stored in a sub-collection of `validation` named `rules`.

Example 6. XSLT transformation to generate the sub-schemas from the global schema based on annotations

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ▶
xmlns:ide="http://oppidum.com/ide" version="2.0">
  <xsl:output method="xml"/>

  <xsl:template match="/">
    <Root>
      <Grammars>
       <xsl:apply-templates select="//*[local-name() = 'element'][@ide:store ▶
='resource']"/>
      </Grammars>
      <Definitions>
        <grammar xmlns="http://relaxng.org/ns/structure/1.0" ▶
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
          <xsl:apply-templates select="//*[local-name() = 'define']"/>
        </grammar>
      </Definitions>
    </Root>
  </xsl:template>

  <xsl:template match="*[local-name() ='element'][@ide:store = 'resource']">
    <grammar xmlns="http://relaxng.org/ns/structure/1.0" ▶
name="{@ide:resource-name}" datatypeLibrary="http://www.w3.org/2001/▶
XMLSchema-datatypes">
      <start>
        <element name="{@name}">
          <xsl:apply-templates select="child::*[not(attribute::ide:store)]"/>
        </element>
      </start>
    </grammar>
  </xsl:template>
  <xsl:template match="*[local-name() ▶
='oneOrMore'][parent::*[attribute::ide:store]][child::*[attribute::ide:store]]"/>

  <xsl:template match="*[local-name() ▶
='zeroOrMore'][parent::*[attribute::ide:store]][child::*[attribute::ide:store]]"/▶
>

  <xsl:template match="*[local-name() = 'define']">
    <xsl:copy-of select="."/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:element name="{local-name()}" namespace="http://relaxng.org/ns/▶
structure/1.0">
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates select="*"/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Running the `generate-all-schema` XQuery script, using `pm.rng` as the global schema of the application, will generate and store three documents in the `schema` collection: `definitions.rng`, `project.rng` and `persons.rng`. The Example 5 shows a corresponding extract of the annotated global schema where the `store` attribute defines the sub-schemas architecture.

As a result, the possibility is given to the developer to validate the data contained in the database by using the validation functions available in the eXist-db environment to check data consistency either against RELAX NG schema or Schematron rules.

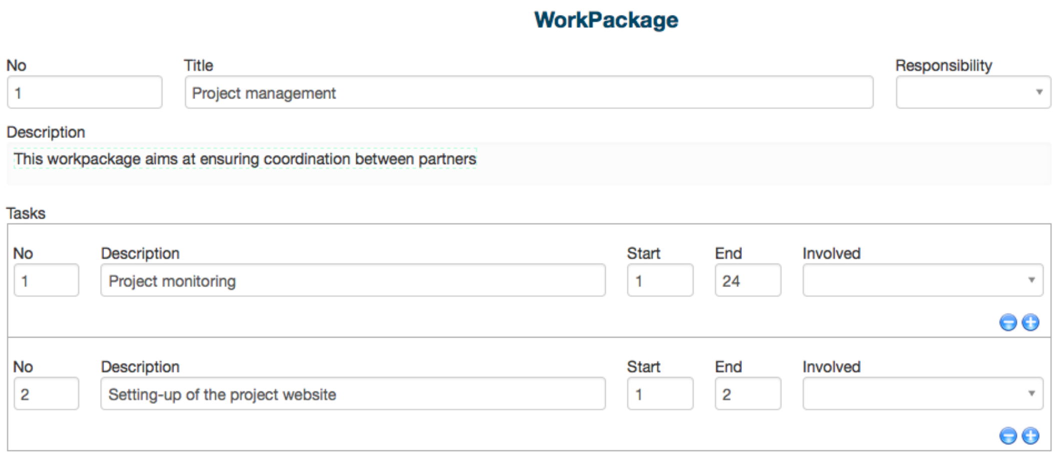
The Example 2 also shows other annotations related to the physical implementation of the data model as a collection (`collection-name` attribute) or as a single document resource (`resource-name`) which has been discussed in Section 4. That information is used to offer validation services anchored on the sub-schemas.

5.2. Authoring issues

The development of XML Web applications relying on the validation of both document-centric and data-centric content need to be sustained by intuitive authoring facilities for end users to populate the database

The AXEL and AXEL Forms JavaScript libraries have been developed in this purpose; jointly used they provide powerful capabilities to indifferently author document-centric and data-centric content in the browser.

Let us consider, as an example, the proposed user interface shown in Figure 5 to edit information about a workpackage. This editing interface contains fields (as usual in forms) to provide factual data such as the workpackage number and title. It also allows to edit, in a "document-oriented" way the description of the workpackage. In that sense the produced XML content will fit our model for representing document-centric information such as modeled in the RELAX NG schema presented in Section 4.


WorkPackage

No	Title	Responsibility
1	Project management	

Description

This workpackage aims at ensuring coordination between partners

Tasks

No	Description	Start	End	Involved
1	Project monitoring	1	24	
2	Setting-up of the project website	1	2	

Figure 5. Editing a workpackage

6. Conclusion

The paper has addressed the issue of the design and implementation of XML Web based applications with a modeling approach that conciliates document-centric and data-centric information. It relies on the adoption of RELAX NG and Schematron to explicitly distinguish between the structural aspects of information and the definition of constraints. It has presented a method to support the developer in designing a unique global schema for an application while validating the database content in a decentralized way during the development process.

The work currently performed about validation is intended to be augmented and integrated in the - currently under construction - Integrated Development Environment of the Oppidum framework.

The process of annotating the global schema of an application is currently under investigation to address other potentialities to be offered to a developer using the Oppidum framework.

The first one is to use the annotated schema to automatically build the structure of collections in the database for a given application. This is more challenging as it may appear since, for instance, the information to be stored about an application does not only consist in XML resources. In our example of project management, we could consider, for instance, to store appendices in any format or pictures of persons in order to publish a project website.

The second one addresses the issue of generating document editing templates to be used by AXEL and AXEL Forms. Our goal is firstly to reduce the extra work needed to create templates that match the constraints on data, and secondly to guarantee the consistency between the global schema of the application and the control checking performed client-side on user's data input when using these libraries.

Bibliography

[1] Serge Abiteboul – Ioana Manolescu – Philippe Rigaux – Marie-Christine Rousset – Pierre Senellart: Web Data Management. Cambridge University Press 2011

[2] Anne Brüggemann-Klein – Mustapha Maalej – Marouane Sayih: XML Schema Identity Constraints Revisited. proceedings of XML Prague 2014, pp 123-145, 2014

[3] Catherine Pugin: Integrated Modeling and Transformation for semi-structured Documents. PhD Thesis, University of Fribourg, Switzerland, 2009

[4] Jayavel Shanmugasundaram and all.: A General Technique for Querying XML Documents using a Relational Database System. ACM SIGMOD, Volume 30, Issue 3, September 2001 Pages 20 - 26

[5] Stéphane Sire: AXEL and AXEL Forms. http://www.oppidoc.fr/en/technologies

[6] Stéphane Sire – Christine Vanoirbeek: Small Data in the Large with Oppidum. proceedings of XML London 2013, pp. 69-79, 2013

[7] Igor Tatarinov and all.: Storing and Querying Ordered XML Using a Relational Database System. ACM SIGMOD'2002, June 4-6, Madison, Wisconsin, USA

[8] Eric van der Vlist: Relax NG, compared. http://www.xml.com/pub/a/2002/01/23/relaxng.html

[9] Christine Vanoirbeek – Vincent Quint – Stéphane Sire – Cécile Roisin: A Lightweight Framework for Authoring Multimedia Content on the Web. journal Multimedia Tools and Applications, 2012.

Graphical User Interface Tool for Designing Model-Based User Interfaces with UIML

Anne Brüggemann-Klein
Technische Universität München
`<brueggem@in.tum.de>`
Lyuben Dimitrov
Technische Universität München
`<dimitrov@in.tum.de>`
Marouane Sayih
Technische Universität München
`<sayih@in.tum.de>`

Abstract

Graphical user interfaces can be designed using various types of editors. Almost all types are based on one of the following two principles: What You See is What You Mean (WYSIWYM) or What You See is What You Get (WYSIWYG). The WYSIWYG tools are the most sophisticated ones and are usually constrained to concrete platforms or formats. On the other hand, WYSIWYM editors come with a presentation neutral way of capturing the semantics of the content, rather than the exact presentation. This principle provides a significant advantage when authoring XML-based user interface documents in terms of device and platform independence. The WYSIWYM paradigm alleviates the stress of exactly displaying the information that is being conveyed, and thus does not require maintaining precise formatting between the Editing View and the Final Result.

In this paper we present a graphical user interface editor based on the WYSIWYM principle for designing model-based user interfaces with the User Interface Markup Language (UIML) as a modeling language, XSLT for transformation and XForms for presentation. We provide a proof of concept and show the applicability of using XML technologies for designing graphical user interfaces for End-user Development.

Keywords: User Interface Markup Language, Graphical User Interface (GUI), WYSIWYM, XML technology

1. Introduction

Graphical user interfaces (GUI), as their name suggests, form a visual interface between user and computer by which the user is capable of communicating with the system using various interaction methods (e.g. reading, writing, clicking, moving and etc.). The development of graphical user interfaces has a considerable value and potential in the field of web engineering and especially in designing large and complex platforms with high demands on usability. Also an important aspect in software development and particularly in the field of web engineering is the capability to involve users or clients at an early stage of the development: The graphical user interface can form a common language between users and developers.

Several related XML technologies such as XForms, XAML, and XUL, have been designed to support different activities across the lifecycle of graphical user interface development. These technologies allow developers to design and implement GUIs effectively. Besides, there exist some tools that provide an easy way to model UI with these languages. One such XML language is the User Interface Markup Language (UIML) which is used to describe requirements, design and implement user interfaces. Our research unit Engineering Publishing Technology Group (EPT-Group) at the Technical University München (TUM) has worked with numerous XML technologies related to XML-based web application development [BMS14-1] [BMS14-2] [BMS14-3] [Kuh14], where these XML technologies are used throughout most or all stages of the application development process. Our research group will continue to use other instruments and languages in this area of research. Part of the quest is to base the development processes on XML technologies rather than just the final systems.

In this paper, we present a graphical UI editor named uimlBuddy for creating graphical user interfaces modeled with XML technologies. Almost all types of editors, especially graphical editors, are based on one of the following two principles: WYSIWYG or WYSIWYM. Although, the most popular way of editing documents is to use tools based on the concept of WYSIWYG, we use the WYSIWYM principle. This principle is still a visual representation but lifts off the requirement that the view between editing the GUI and presenting the GUI in the system are strictly identical: Editing view has to express a less strict formatting and focus on capturing the semantics and the intention of how the system GUI should work. This principle is better suited for organizing UI in web-based platforms, because we cannot guarantee that the UI created by the editor depicts the final UI (i.e., XForms, XHTML) in all browsers and across all devices (different screen sizes, form factors, etc.). Thus, in terms of platform and device independence, the WYSIWYM works in our favor, since it puts emphasis on the actual information being conveyed, instead of the precise formatting.

The proposed UIML editor was implemented in Java by Lyuben Dimitrov as part of his Master thesis project. The goal for this editor is to present an easy and straightforward way to create valid and well formatted UIML documents. Our objective is to demonstrate that UIML provides a practical mechanism for describing and representing user interface and to facilitate the usage of UIML for end user development.

The remainder of the paper is organized into five sections as follows: In the section called "User Interface Markup Language (UIML)", we give a short overview of the UIML Language that can be used to provide an XML representation of any user Interface. In the sections called "Proposed modifications in the MIM and the UIML" and "MIM model according to UIML specifications", we explain the proposed simplifications in the Meta Interface Model (MIM) to facilitate the uimlBuddy proof of concept implementation. In the section called "The concept of uimlBuddy", we describe the concept of the editor. In the section called "The Editor uimlBuddy", we explain what are the features and operations offered by the uimlBuddy editor. Finally, in the section called "Conclusion and future work", we draw some conclusions and raise ideas for future work.

2. Model-based User Interface Development

Model-based User Interface Development or as it is commonly referred to as Model-driven User Interface Development (MDUID) is a concept that first emerged around the 1980s [Mye95]. Main motivation factor that led to MDUID is that creation of user interfaces became a tedious, time consuming and costly task [MRo00]. Developers and designers of UI have to consider several important aspects in the creation of UIs. Most notable of them are related to:

- The vast proliferation of new devices available on the market nowadays.
- Support for different platforms and environments each with its own constraints.
- Abundance of programming languages, libraries and frameworks for development of UI.
- The gap between end users with respect to their backgrounds, age, preferences and skills.

Several frameworks exist which try to reduce the development effort by targeting these multiple contexts and multiple targets. One of them is the Cameleon Reference Framework (CRF), which is a meta architecture decomposing the process of UI design into several different components [CRF09] as seen in Figure 1:

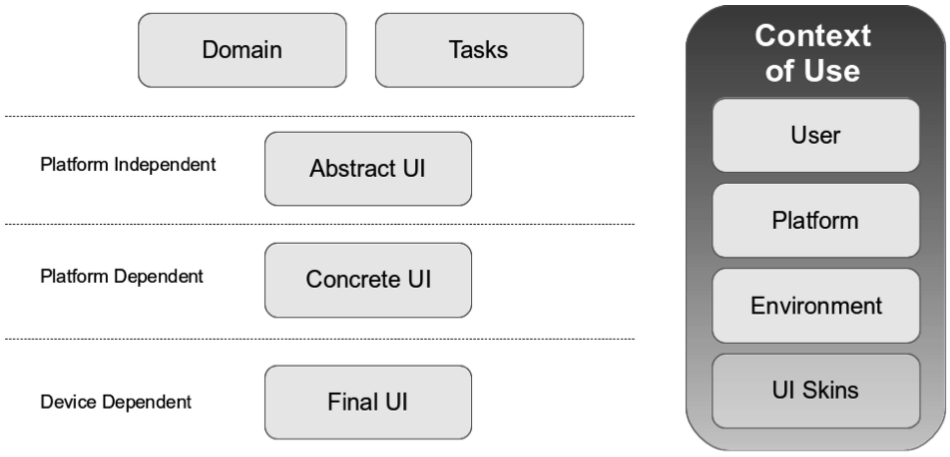


Figure 1. The Cameleon Reference Framework [CRF09]

The framework includes several different levels of abstraction:

- Task level reflects the chain of tasks that need to be carried out in a concrete order to achieve the desired user interaction result.
- The Abstract User Interface (AUI) describes the UI in terms of abstract interaction objects that are independent of any platform or modality.
- The Concrete User Interface (CUI) describes the user interface with concrete interaction objects that are modality dependent, but independent in terms of platform.
- The Final User Interface (FUI) is now platform-dependent source code that can be represented in any language (HTML, Java). The FUI layer is the final result with which the user interacts.

Transformations from one level of abstraction to another can be performed in both directions (forward and reverse engineering) [CRF09].

2.1. User Interface Markup Language (UIML)

The UIML language is an XML-based language for defining and describing user interfaces in declarative terms and abstract form. It is supported and standardized by the Organization for the Advancement of Structured Information Standards (OASIS). In the field of Software Engineering, it is widely accepted that the user interface should be decoupled from the logic. The UIML is modeled by the Meta-Interface Model (MIM), which divides the user interface into three different components: presentation, interface, and logic. These components are defined according to [Pha00] as:

- The logic component gives user interface a canonical way to communicate with an application.

- The presentation component allows the user interface to be displayed independently of the underlying platform.
- The interface component offers the interaction between the user and the application.

As a consequence, the user interface can be reused across multiple applications while preserving its consistency. Furthermore, this decoupling allows developing the UI and the logic separately.

Improving on early models, the MIM subdivides the <interface> component into four additional subcomponents: structure, style, content, and behavior. Take a look at the UIML skeleton below:

```
<?xml version="1.0"?>
<uiml>
 <interface>
  <structure> …
   <part id="…" class="…"/>
  </structure>
  <style> … </style>
  <content> … </content>
  <behavior> … </behavior>
 </interface>
 <peers>
  <presentation/>
  <logic/>
  <peers>
</uiml>
```

- Structure: defines the physical organization of the UI. Each <structure> contains different <part> elements, where each part represents a concrete UI element. Each part is also associated to a certain "class" that denotes the type of UI element (e.g. label, button, etc.).
- Style: specifies the appearance of the elements (e.g. text, color, size).
- Content: specifies the content of each part (if any) such as string literals, images and etc. Useful when you want to further abstract away the UI by not binding it to static content. For example a UI in both English and German can be practically reused just by providing content for the German language.
- Behavior: specifies the actions that are triggered under special circumstances (button click).

The <logic> element is responsible for linking the concrete final UI vocabulary with the final Logic and each element in the interface component. The <presentation> element either defines the mappings to names of classes of the concrete UI toolkit (Java AWT, XHTML, etc.) or it points to an external file where this vocabulary is defined. This concrete UI metaphor is linked with an abstract UIML part element

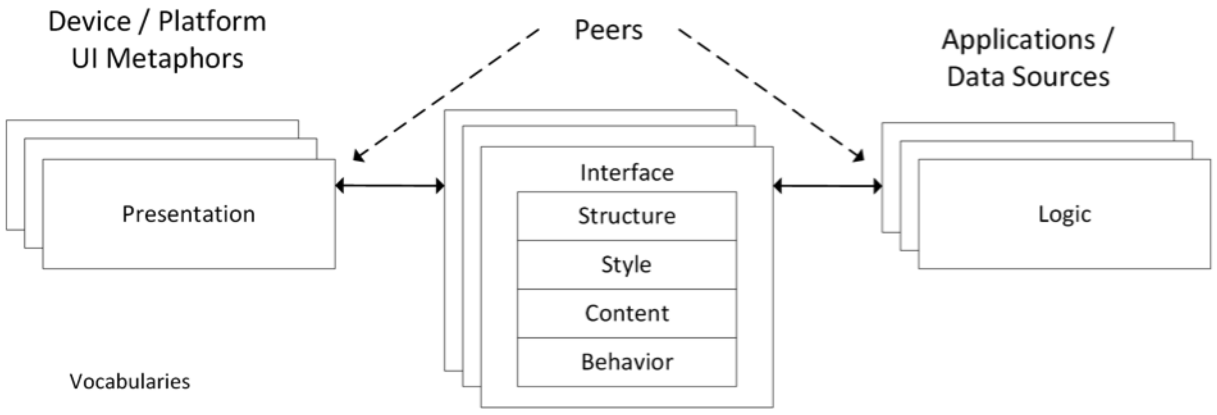


Figure 2. Meta-Interface Model [UIML4]

for the final UI. The Logic is also an external file, where the behavior of each part element is matched with the concrete software implementation for the final target platform. The advantages of this are that the Interface component stays abstract (i.e., free of any concrete implementation details), while the presentation and logic components can be substituted for different platforms and target UIs.

In order to obtain the FUI, the UIML document has to be transformed. Our target language is XForms and we have decided to use XSL Transformations that render the UIML into XForms. If we have to relate this approach to the Cameleon Reference Framework, it would look like this:

The actual process of transforming an UIML document into the final platform is divided into several steps, as shown in Figure 4below:

1. The UIML document is processed with the custom XSLT Transformations, where each individual part is handled by a different XSL template that assigns its corresponding style, behavior and content. The UIML document can be regarded as the Abstract User Interface (AUI).
2. After the transformation, we produce an XForms document with a declarative model containing data types, instances, data submission parameters, and other declarations, if required. The View layer contains the interface controls that were matched and transformed in Step 1 via XSLT. This is regarded as the Concrete User Interface (CUI). Additionally, references to CSS and necessary JavaScript files are added and used later on in styling the resulting HTML element).
3. Since browsers do not natively support XForms, the resulting CUI has to be transformed once again with the help of an XForms implementation of choice (e.g., orbeonForms, betterFORMS or XLSTForms).
4. Finally, the HTML is generated by the XForms implementation and the HTML elements are displayed and styled according to the CSS stylesheets provided in Step 2. This is regarded as the Final User Interface (FUI).

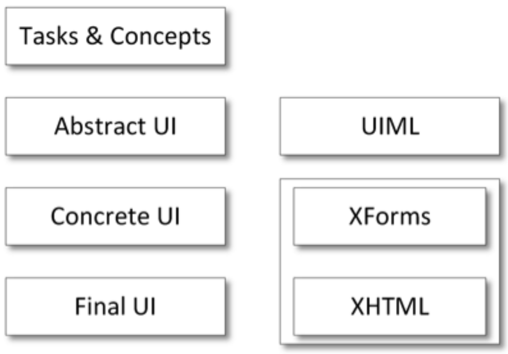


Figure 3. Our approach with respect to CRF

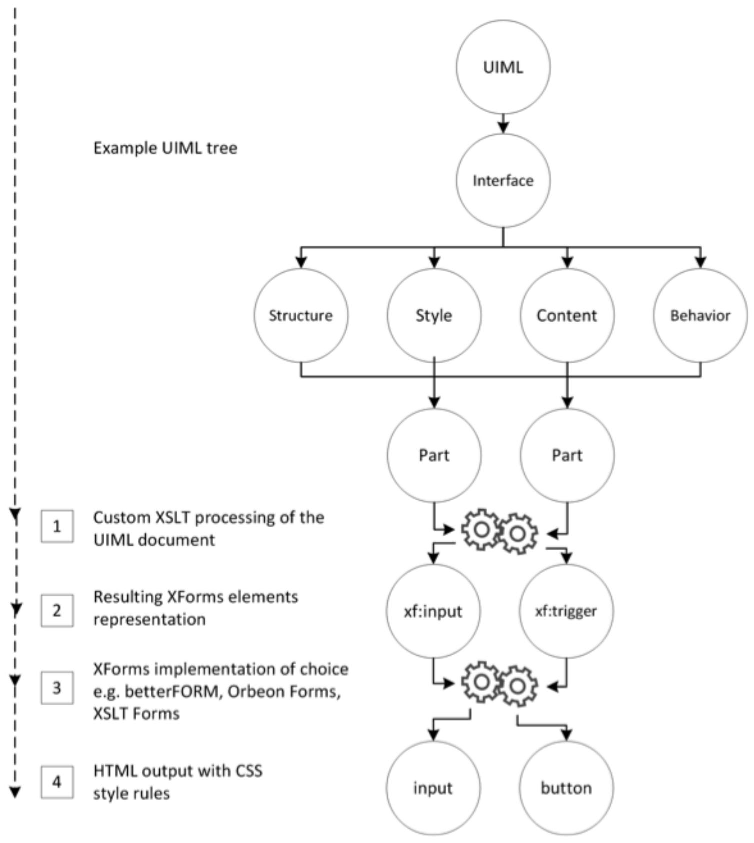

Example UIML tree
UIML
Interface
Structure
Style
Content
Behavior
Part
Part
1 Custom XSLT processing of the UIML document
2 Resulting XForms elements representation
3 XForms implementation of choice e.g. betterFORM, Orbeon Forms, XSLT Forms
4 HTML output with CSS style rules
xf:input
xf:trigger
input
button

Figure 4. UIML Transformation Process

The undertaken approach of styling UIML documents has its benefits and drawbacks. The main disadvantage is that it requires a prior knowledge of the available CSS styles. On the other hand, the main advantage is that the XSLs can be edited to reference any style sheet in the final UI including an absolute URL that points to a remote CSS file. This means that the styling is completely decoupled from the process, but the uimlBuddy editor provides the option to directly specify the styles while modeling the UI. In this way, the user is free to use any modern CSS framework of his/her choice with flexible updates and improvements.

2.2. Proposed Modifications to the MIM and UIML

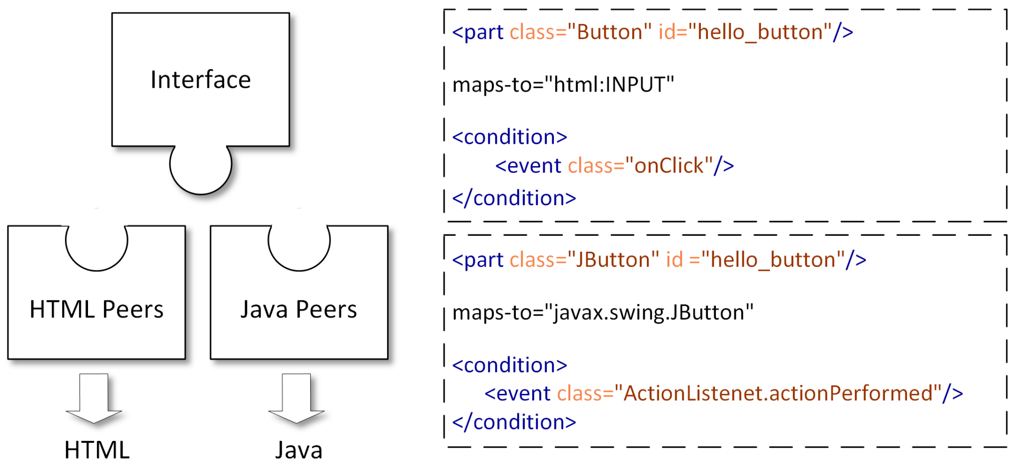
In this section, we explain modifications we applied to UIML and the MIM and why we propose them. The underlying reason for the modifications is the desire to simplify and facilitate the XSL Transformation scenarios and subsequently, the uimlBuddy editor implementation. In brief, the original Meta-Information Model assumed by UIML is modified such that we don't use the 'peers' element with its sub elements. The second change is related to the way of mapping style, content and behavior to structure elements. We have simplified it down to the use of a single unique id attribute. And last but not least, we are getting rid of the idea of precedence in UIML. These modifications are explained in detail in the next section.

2.2.1. Modifications to the Meta-Interface Model

The MIM Model in Figure 2 shows another element called "peers" in addition to the logic and presentation elements. This "peers" element specifies the widgets in the target platform and the methods in scripts, programs, or objects in the application logic, which are associated with the user interface [Pha00].

In Figure 5, we have two different presentations – one for Java and one for HTML (only one presentation is used at the same time with the Interface element). Let's assume that we have a part button defined in the Structure element. In the Java Presentation, this part would be mapped as a "JButton" if we use Java Swing, while in the HTML Presentation it would be mapped as <input type="button">. Besides, in the Behavior element, each rule has a condition, such as "ButtonPressed". In the Java presentation, this would be mapped to "ActionListener.actionPerformed", while in the HTML Presentation the mapping will be to "onClick". The Logic element will map to an exact name with the external application logic. To paraphrase, it acts as the glue between the UI and application logic in the resulting final application. In other words, in the Behavior element, after a condition is met, an event is triggered (such as "onClick") and an "action" is taken. This action contains a call with the name of the method in the external application logic.

Since we decided to use XSLT to transform an UIML document into an XForms document, for the sake of the proof of concept, we target only one specific FUI. Thus, we exclude the <peers> element and its subelements: <presentation> and <logic>. In our scenario, the <presentation> of XForms is defined within the very XSLT files. The logic element maps to exact method names that are present in the "behavior" (under the "call" elements) to the application logic, which is external of the UIML document as shown in Figure 2. If we eliminate that mapping and use the method name from the "behavior" call element in the XSL transformation dir-



```
<part class="Button" id="hello_button"/>

maps-to="html:INPUT"

<condition>
      <event class="onClick"/>
</condition>
```

```
<part class="JButton" id ="hello_button"/>

maps-to="javax.swing.JButton"

<condition>
    <event class="ActionListenet.actionPerformed"/>
</condition>
```

Figure 5. UIML Presentations

ectly, we can simplify the writing/generation of UIML documents and the transformation scenarios themselves. The disadvantage of this technique is that if another platform should be considered, a new XSLT will have to be constructed to map to a new vocabulary. Furthermore, the current state of the XSLT files does not support all different types of UI metaphors. If a new UI control is included at a later stage, this implies new XSLT templates have to be further created in order to support the new control.

The proposed modifications in the MIM are shown in Figure 6:

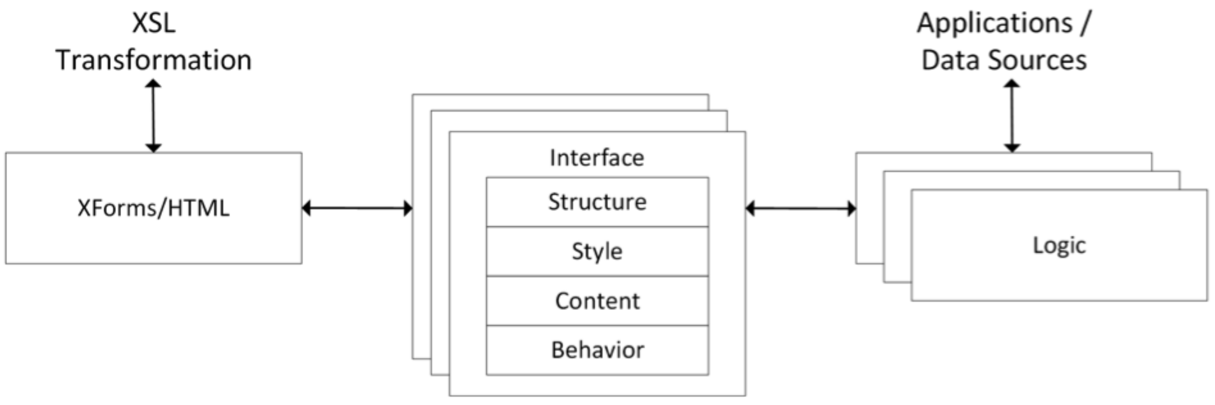


Figure 6. Modified MI Model

2.2.2. Modifications to UIML Specification

Since we focus on the Interface element, we propose two changes to the UIML specification that simplify the XSLT scenarios from developer's perspective and also make the resulting UIML document more concise and easy to read and understand. We demonstrate that in the next two subsections.

2.2.2.1. Style, Content and Behavior Mapping

The following example illustrates how the style, content and behavior of each part is distributed across the interface component:

```
<structure>
 <part id="affirmativeChoice" class="Button"/>
 <part id="negativeChoice" class="Button"/>
</structure>
<style>
 <property part-name="affirmativeChoice"
  name="backgroundColor">green</property>
 <property part-name="affirmativeChoice" name="label">
  <reference constant-name="affirmativeLabel"/>
 </property>
 <property part-name="negativeChoice"
  name="backgroundColor">red</property>
 <property part-name="negativeChoice" name="label">
  <reference constant-name="negativeLabel"/>
 </property>
</style>
<content>
 <constant id="affirmativeLabel" value="Yes"/>
 <constant id="negativeLabel" value="No"/>
</content>
```

As shown in the above code listing, there are two parts – each has a unique "id". The <property> elements match the corresponding <part> element's id with the "part-name" attribute. Furthermore, the "label" property element contains a <reference> element that refers to the "id" of another <constant> element. This way of combining information for a single part element is not really required when compiling UIML documents with XSL Transformations. We propose to change all sub element references only to a single, unique "id" attribute and also eliminate the <reference> element inside the <property> element (in other words, match the content directly according to <part> "id" attribute).

The above example can be simplified to the following script:

```
<structure>
 <part id="affirmativeChoice" class="Button"/>
 <part id="negativeChoice" class="Button"/>
</structure>
<style>
 <property id="affirmativeChoice"
  name="backgroundColor">green</property>
 <property id="negativeChoice"
  name="backgroundColor">red</property>
</style>
```

```
<content>
 <constant id="affirmativeChoice" label="Yes"/>
 <constant id="negativeChoice" label="No"/>
</content>
```

To sum up the changes:

1. The only way to map <part> elements to its styles, contents and behavior is by using their attribute "id".
2. Content elements are no longer referenced from Style elements. The <reference> elements are eliminated. Instead, we directly target the content by the unique "id".
3. Attribute names in a Content element are specifically targeted to what they represent. For example, if the content is Label, the <constant> element would have a "label" attribute name, if it were an Image, the <constant> element would have a "source" (or image path) attribute name.

Overall, this modification is targeted more at simplifying the XSLT matching rules than the uimlBuddy editor implementation. If further UI elements are added to the XLS Transformations, the effort to create the transformation scenarios will be reduced. Nevertheless, in the notion of XHTML and XForms documents, the required UIML document is shorter and simpler to read and write. This lowers the high threshold of Model-based User Interface Development (MBUID) for new designers and developers. Besides, parsing an already existing document and visually rendering it in the editor uimlBuddy is simplified which can lead to a better tool that increases the ceiling of the MBUID approach.

2.2.2.2. Style precedence

According to the UIML Committee Draft version 4.0, styles, contents and behaviors can be either nested into a structure part or referenced from external interface sub elements. It is important to distinguish between those different styles, because they can have different precedence and order of execution. This modification aims to handle the precedence and conflict resolution issues. The following example considers this issue:

```
<?xml version="1.0"?>
<uiml>
 <interface>
  <structure>
   …
   <part id="Button1" class="Button">
    <style>
     <property name="backgroundColor">blue</property>
    </style>
    …
  </structure>
  <style>
   <property id="Button1" name="backgroundColor">
    orange</property>
   <property id="name_input" name="backgroundColor">
    yellow</property>
  </style>
 </interface>
</uiml>
```

Here, Button1 part is given three different colors – blue, orange and yellow. A rendering or interpreting engine must resolve this conflict by following some precedence rules ordered from highest to lowest priority:

- <property> elements nested inside the <part>
- <property> elements with a valid part-name attribute located in a separate style section
- <property> elements with a valid part-class attribute

The XSLT Specifications [Kay07] already have other rules to handle styling issue. These are the XSLT rules, which apply in order:

- Templates in the primary style sheet have higher precedence than the imported templates.
- Templates with a higher value in their "priority" attribute have higher precedence, while templates without a priority attribute are assigned a default priority.
- Templates with more specific patterns take precedence.

For example, suppose we have the following style sheets:

- Stylesheet A imports stylesheets B and C in that order;
- Stylesheet B imports stylesheet D;
- Stylesheet C imports stylesheet E.

Then the order of import precedence (lowest first) is D, B, E, C, A. If the previous three steps leave more than one template in consideration, it would be considered as an error, but XSLT processors can recover by defaulting to the last one in the file.

But to achieve the precedence implied by nesting in UIML, XSLT files would require duplication of the matching rules in the main style sheet and then in the imported stylesheets, which is code repetition. We decided to further simplify them by using only one way. We choose to use references with valid id attributes in separate style, content and behavior sections and disallow nesting styles, contents and behaviors in the structure parts. However, when using the uimlBuddy editor, it is not necessary for the user to concern about such possible scenarios or understand/know how the UIML document is written. Overall, the idea of having different ways of mapping the style, content and behavior is very useful in the context of using UIML documents at runtime. If another UIML rendering engine were to be created outside of XSLT, it would make perfect sense to support it with or without an editor.

3. The Concept of the Editor uimlBuddy

The main concept of the editor uimlBuddy is to allow non-professionals in XML related languages and users who have no technical background to create valid UIML and XForms documents. We refer to these users as domain experts. These domain experts are not required to write the UIML code in order to model the user interface, but rather use the editor and only to fill the mandatory and necessary properties of the UIML elements in a user-friendly manner, e.g. via popups windows and dialog boxes. Besides, this category of users can access the generated UIML source code as well as the final resulting XHTML document containing the XForms output.

Additionally, the editor allows to manually editing the XSLT files which perform the transformations to XForms and the CSS files responsible for styling the final UI. We call users who access these functionalities the developers. The developer can dive into the XSLT files into a window separate from the typical graphical editor. They can modify the actual transformations for both the developer and domain experts later on and perform modifications to the CSS files, which can be altered or completely substituted by different ones.

The final XForms result is obtained after matching the UIML document with specifically created XSL transformation scenarios. These scenarios link the UIML document to an XSL file, which does the actual transformation and outputs a document of XHTML type.

3.1. The Editor uimlBuddy

The main goal of the editor is to retain simplicity at a minimum, where it is sufficient enough to assist the user in creating user interfaces written in UIML. Thus, we decided to exclude features such as XML validation, XML tree representations and etc., in order to focus on the key feature – modeling UI. Furthermore, in order to facilitate this End-User Development, the application uses common UI look and feel. Next, were going briefly through some of the main features of the editor.

3.1.1. Main Application Window

The user interacts via several panels, namely a panel for inserting new elements (left expandable control), a panel for the graphical representation of the document and last but not least, a panel for reviewing the generated UIML code. There are also a menu bar and a toolbar which consist of a buttons set with common application functionalities.

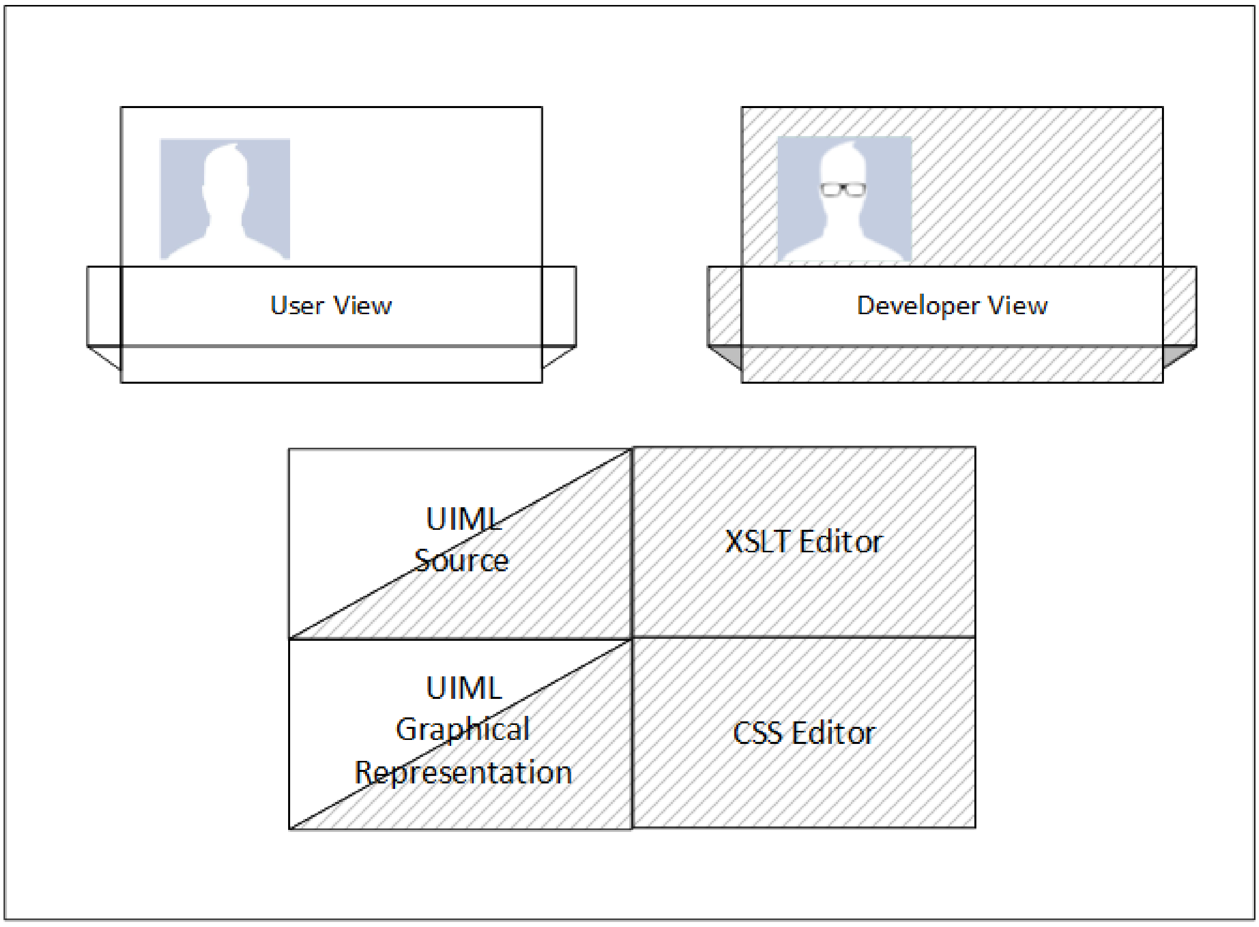


Figure 7. uimlBuddy User Access

3.1.2. GameX Main Application Window

To illustrate the functionalities of the editor, we create a portion of the GameX [BMS14-3] user interface using the editor as an example. We start with the top bar menu as shown in the screenshot of the game:

3.1.3. Working with the Editor

In order to insert an element, the user selects an item from the left expandable panel, which is divided into three subcategories – Layouts, Controls and Miscellaneous. Immediately upon selection, the user is presented with a dialog where the required input per element needs to be inserted. Due to the WYSIWYM nature of the editor, the user does not need to input properties such as size and colors (they are predefined). The top menu bar in the GameX has a horizontal orientation, thus the first thing we are going to insert in our UIML document is a horizontal layout that will contain all other elements inside it:

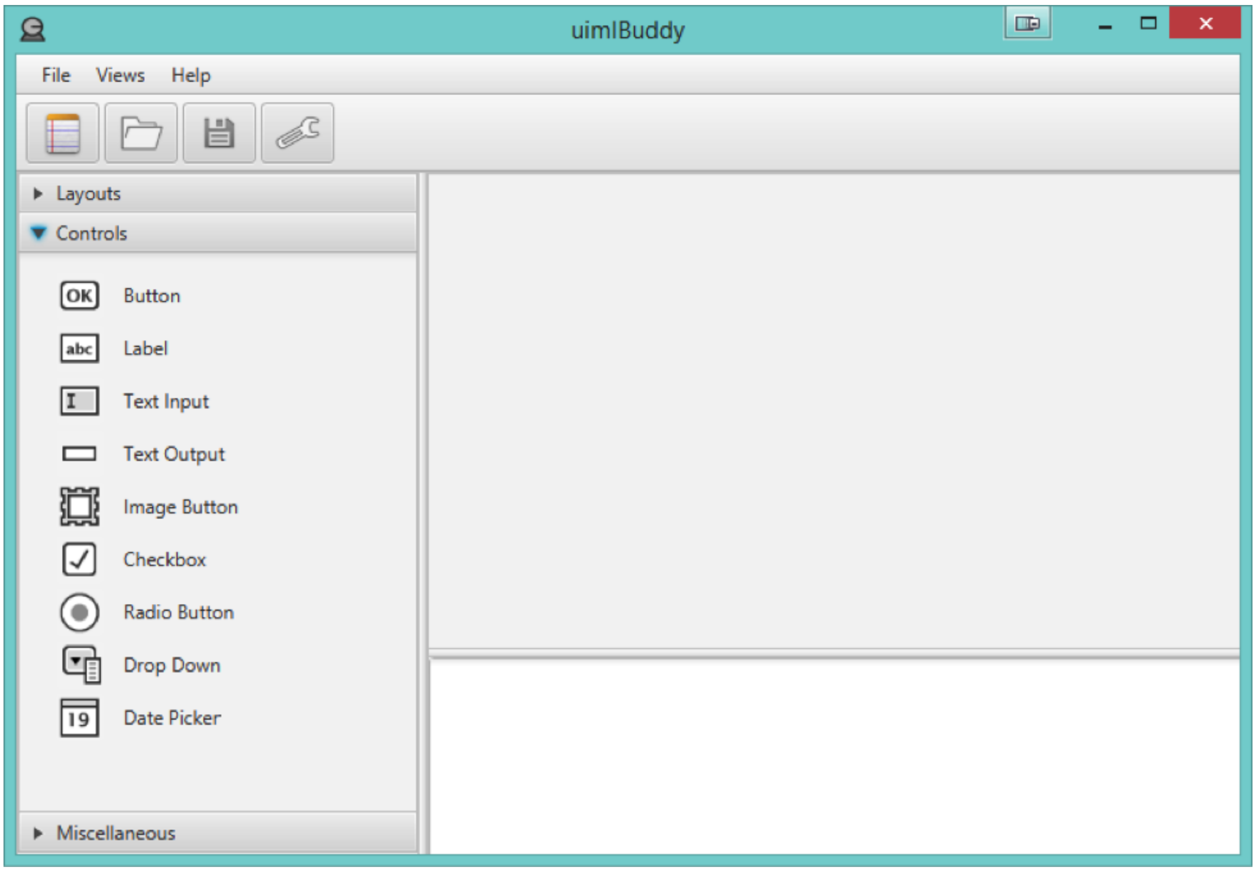


Figure 8. Main Application Window

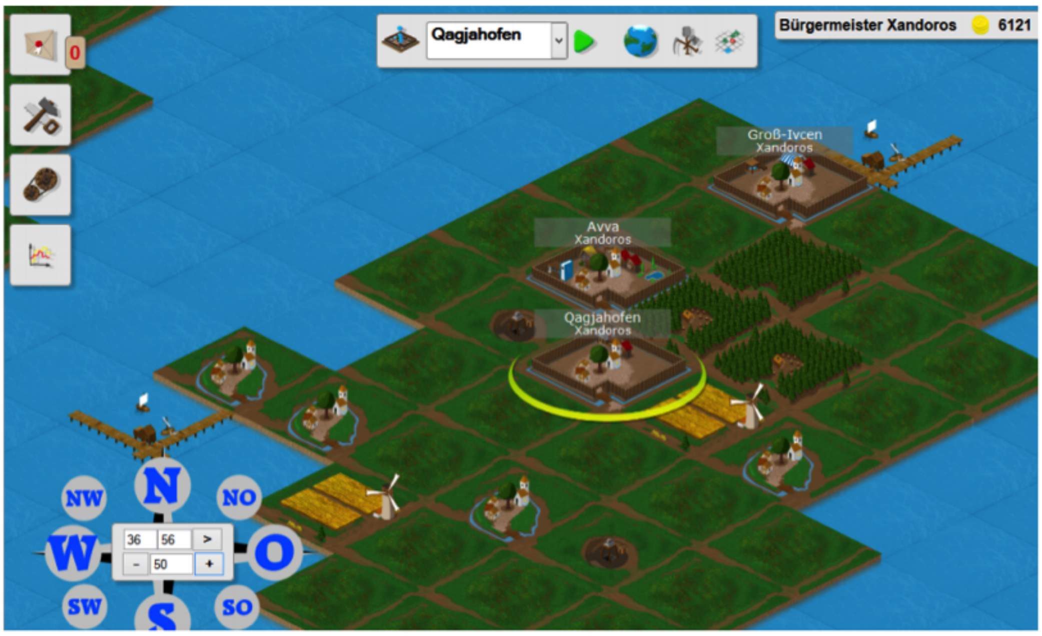
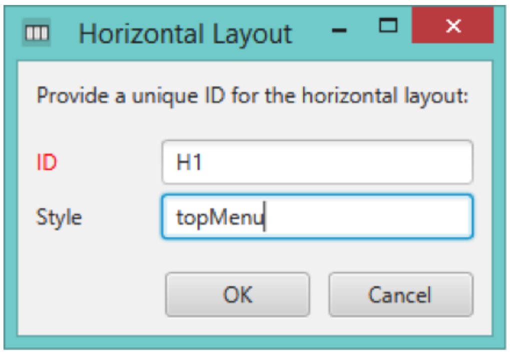


Figure 9. GameX Main Application Window



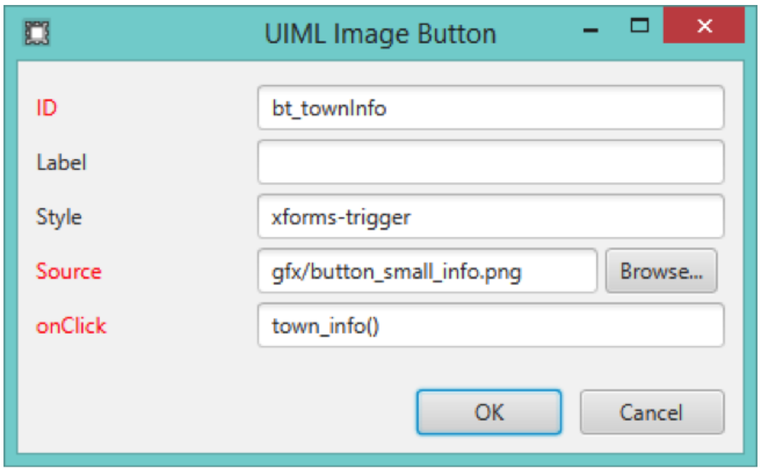
Provide a unique ID for the horizontal layout:

ID	H1
Style	topMenu

Figure 10. Inserting a Horizontal Layout

Once the layout is inserted, everything contained inside will be laid out in a horizontal fashion. The users can also nest layouts inside one another in order to create more complex UI.

Next, we would like to insert an image button, as part of the menu.



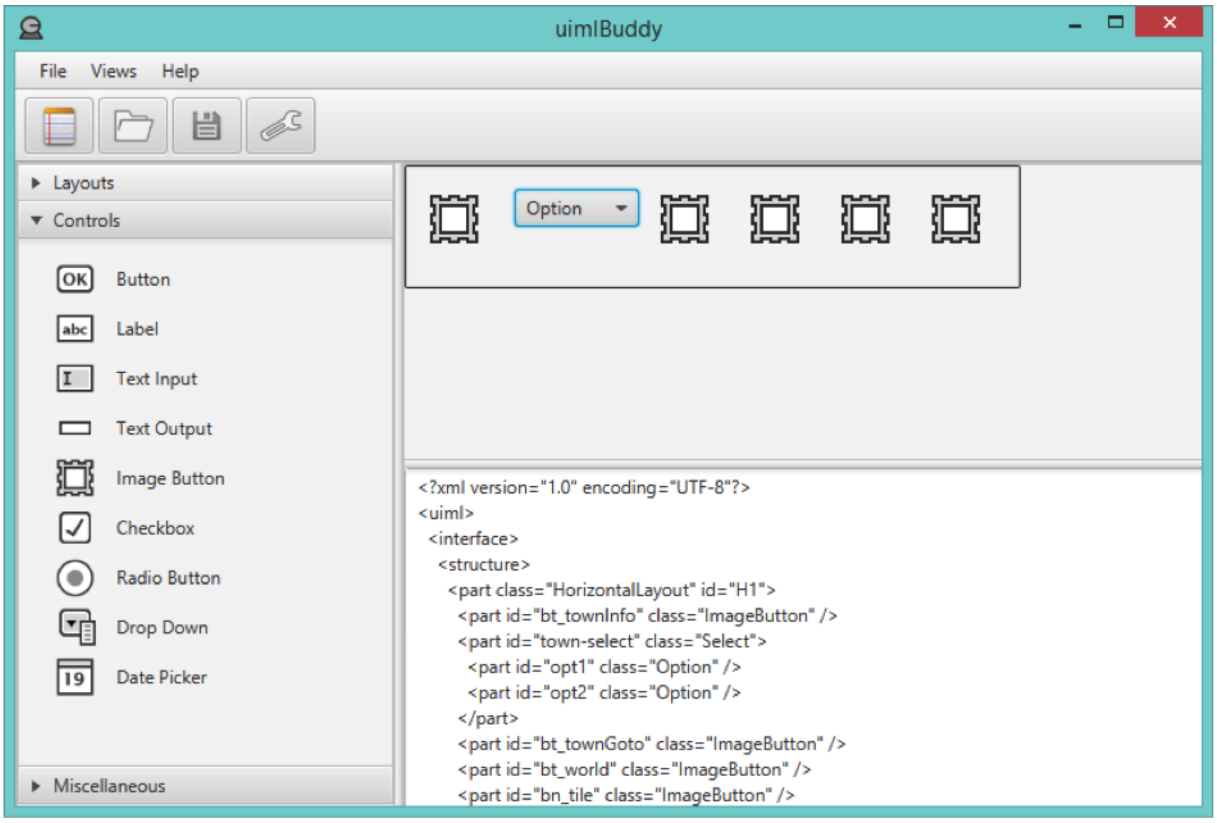
ID	bt_townInfo
Label	
Style	xforms-trigger
Source	gfx/button_small_info.png
onClick	town_info()

Figure 11. Inserting an Image Button

The above dialog is presented when user selects to insert an Image Button. The ID field is used to map the style, content and behavior for this particular image button. The source field contains the path where the image file is located and last but not least, the onClick field denotes the method that will be executed when an onClick event on this button occurs. The domain expert does not provide the actual logic in the onClick event to support that user interaction, but rather specifies only the name of the method or function that will be implemented by a developer in a concrete implementation language depending on the platform and device.

We need total of five such buttons and one drop down menu for selecting towns for our example from Figure 9.

The final result looks like this in the editor:



```
<?xml version="1.0" encoding="UTF-8"?>
<uiml>
 <interface>
  <structure>
   <part class="HorizontalLayout" id="H1">
    <part id="bt_townInfo" class="ImageButton" />
    <part id="town-select" class="Select">
     <part id="opt1" class="Option" />
     <part id="opt2" class="Option" />
    </part>
    <part id="bt_townGoto" class="ImageButton" />
    <part id="bt_world" class="ImageButton" />
    <part id="bn_tile" class="ImageButton" />
```

Figure 12. Final Result for TOP Menu bar Example

Now we are ready to transform the UIML document into an XForms document. For that purpose we use the "wrench" icon button on the toolbar.

3.1.4. Final Result

The final XForms result, obtained via the XSL files contained within the editor, is then rendered in the browser:

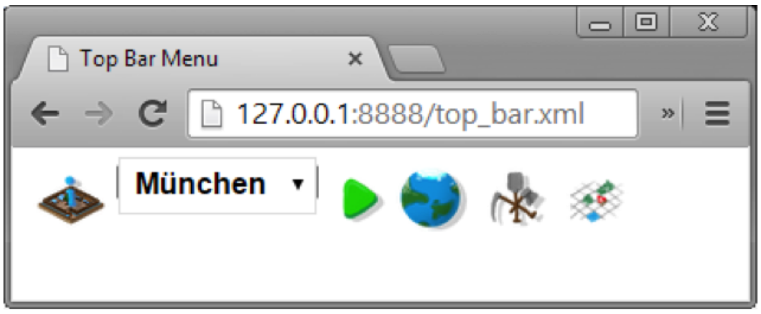
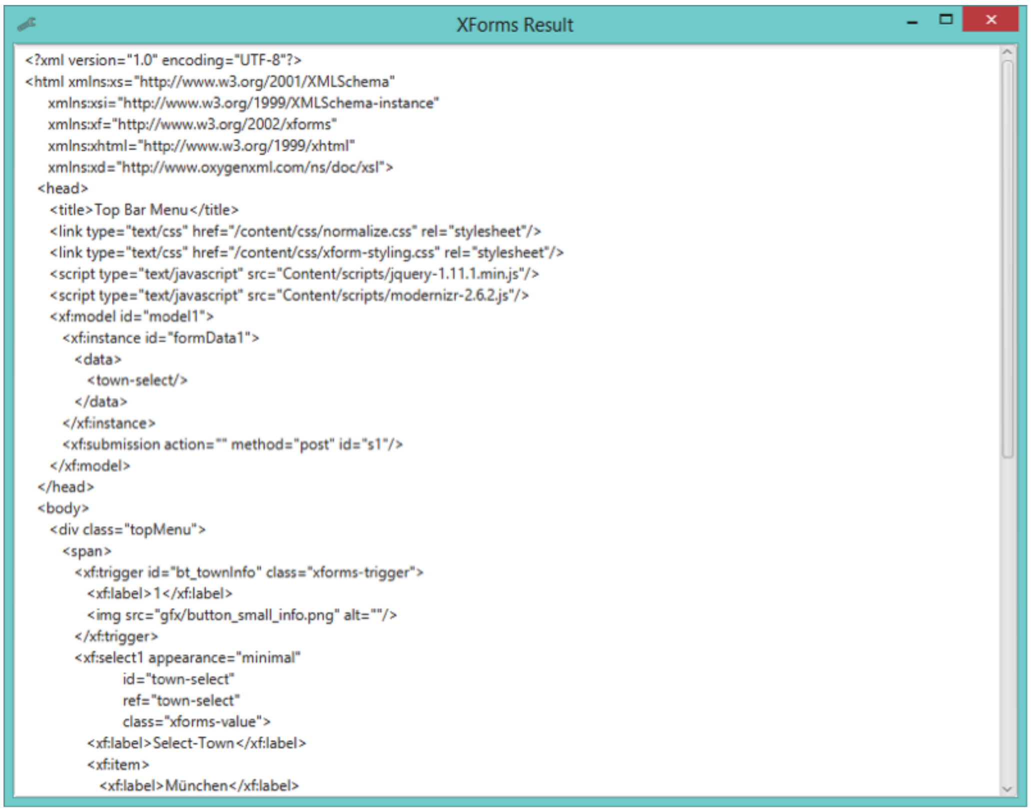


Figure 14. Final Result Rendered in the Browser via XSLTForms

This last step is manual and requires the user to save the result of the transformation to a new file that is then opened with an XForms implementation of choice.



```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xf="http://www.w3.org/2002/xforms"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl">
  <head>
    <title>Top Bar Menu</title>
    <link type="text/css" href="/content/css/normalize.css" rel="stylesheet"/>
    <link type="text/css" href="/content/css/xform-styling.css" rel="stylesheet"/>
    <script type="text/javascript" src="Content/scripts/jquery-1.11.1.min.js"/>
    <script type="text/javascript" src="Content/scripts/modernizr-2.6.2.js"/>
    <xf:model id="model1">
      <xf:instance id="formData1">
        <data>
          <town-select/>
        </data>
      </xf:instance>
      <xf:submission action="" method="post" id="s1"/>
    </xf:model>
  </head>
  <body>
    <div class="topMenu">
      <span>
        <xf:trigger id="bt_townInfo" class="xforms-trigger">
          <xf:label>1</xf:label>
          <img src="gfx/button_small_info.png" alt=""/>
        </xf:trigger>
        <xf:select1 appearance="minimal"
                id="town-select"
                ref="town-select"
                class="xforms-value">
          <xf:label>Select-Town</xf:label>
          <xf:item>
            <xf:label>München</xf:label>
```

Figure 13. Transformed UIML Document into XForms Document

4. Conclusion and Future Work

The main challenge was to develop the environment that allows people to create their own UIs without any XML knowledge [EUD06]. End-User Development is a set of techniques and methods that allow users without professional background to create or modify a software artifact. At the same time, the editor allows professional users to modify the XSL transformations performed in background and the Cascading Style Sheets governing the appearance of the final UIs.

The editor allows the users to create both UIML and XForms documents. In other words, the end user might only create and save the UIML documents without transforming them to XForms (or transforming them at a later stage). The actual transformation is concealed from the end user, hiding away the additional complexity.

The uimlBuddy editor is still a proof of concept, meaning that its functionality is limited and not all types of UI controls and interactions are supported yet. Moreover, additional work is required to further facilitate the End-User Development, more precisely; additional user-aiding concepts such as code highlighting, tips and hints.

Our approach and editor can theoretically support all kinds and forms of user interfaces. UIML was also created with that intention in mind. At the same time, the current state of the XSL transformations and the uimlBuddy editor tend to be more suited towards form-based and grid-based user interfaces. In oder to support a diversity of graphical user interfaces, we need to support additional types of layouts that are more flexible.

Another step forward is the ability of the editor to support multi-modal development in the future. For that purpose additional functionality has to be created that will support the linking towards different interface models, or task models. Additionally, support for adaptive and plastic UI, as concepts related to adapting to varying context, could be considered as a future path for extending out approach.

Furthermore, in this paper we demonstrated the use of UIML without the <peers> tag – a significant part of the Meta-Information Model. That being said, the <peers> tag and its subelements can be reintroduced at a later stage and thus allow modeling of user interfaces for more concrete platforms other than XForms and XHTML.

In this paper, we presented a WYSIWYM editor for the User Interface Markup Language (UIML). The editor is able to create UIs and transform them via XSLT to XForms, which is our concrete target platform. Additionally, we presented modifications to the latest specifications of UIML and the underlying MIM model to facilitate the proof of concept of end user development with UIML.

Bibliography

[1] Further references: UIML, GameX, Event-Based Programming, Cameleon Reference Framework, End-User Development, User Interface Software Tools, Graphical User Interface Programming.

[2] Anne Brüggemann-Klein, Mustapha Maalej, Marouane Sayih. *XML Schema Identity Constraints Revisited*. XMLPrague 2014, 2014. *available from http://www.xmlprague.cz/sessions2014/#xsd*.

[3] Anne Brüggemann-Klein, Mustapha Maalej, Marouane Sayih. *Identity Constraints for XML* . Balisage 2014, 2014. *available from http://www.balisage.net/Proceedings/vol13/html/Maalej01/BalisageVol13-Maalej01.html*. *doi:10.4242/BalisageVol13.Maalej01.*

[4] Marouane Sayih, Martin Kuhn, Anne Brüggemann-Klein. *GameX - Event-Based Programming with XML Technology* . Balisage 2014, 2014. *available from http://www.balisage.net/Proceedings/vol13/html/Bruggemann-Klein01/BalisageVol13-Bruggemann-Klein01.html*. *doi:10.4242/BalisageVol13.Bruggemann-Klein01.*

[5] Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J. *The CAMELEON Reference Framework*. *available from http://giove.isti.cnr.it/projects/cameleon.html*.

[6] Michael Kay. *XSL Transformations (XSLT) Version 2.0, W3C, 2007, Conflict Resolution for Template Rules* . available from http://www.w3.org/TR/xslt20/#conflict.

[7] Martin Kuhn. *Lerning Systemic Thinking: Design and Implementation of a Browser Game based on XML Technology*. Master Thesis, TU München, 2014.

[8] Henry Lieberman, Fabio Paterno, Markus Klann, Volker Wulf. *End-User Development: An Emerging Paradigm*. Human-Computer Interaction Series, Volume 9. Springer 2006. *pp. 1-8*.

[9] Mustapha Maalej, Anne Brüggemann-Klein. *Generating Schema-Aware XML Editors in XForms* . Balisage 2013, 2013. *available from http://www.balisage.net/Proceedings/vol11/html/Bruggemann-Klein01/BalisageVol11-Bruggemann-Klein01.html. doi:10.4242/BalisageVol11.Bruggemann-Klein01.*

[10] Brad A. Myers. *Graphical User Interface Programming*. chapter 48 of Computer Science Handbook -- Second Edition. Allen B. Tucker, editor in chief. Boca Raton, FL: Chapman & Hall/CRC Press, Inc., 2004. *pp. 48-1 - 48-29*.

[11] Brad A. Myers. *User Interface Software Tools*. ACM Transactions on Computer-human Interaction, Vol. 2, No. 1, March, 1995. *pp 64-103*.

[12] Brad A. Myers, Mary Beth Rosson. *Survey on User Interface Programming*. Proc. of the 10th Annual CHI Conference on Human Factors in Computing Systems, 2000. *pp. 195-202*.

[13] Constantinus Phanouriou. *UIML: A Device-Independent User Interface Markup Language*. PhD Thesis, Virginia Polytechnic Institute and State University, 2000.

[14] UIML4.0 specification. *available from http://docs.oasis-open.org/uiml/v4.0/uiml-4.0.pdf, 2009.*

Survey State Model (SSM)

XML Authoring of electronic questionnaires

Jose Lloret
The Robert Gordon University
<j.m.lloret-perez@rgu.ac.uk>
Nirmalie Wiratunga
The Robert Gordon University
<n.wiratunga@rgu.ac.uk>

Abstract

Computer Assisted Interviewing (CAI) systems use questionnaires as the instruments to conduct survey research. XML constitutes a formal way to represent the features of questionnaires which include content coverage, personalisation aspects and importantly routing functionalities. In this paper we conduct a comparative analysis on different XML approaches to questionnaire modelling. Our findings suggest that existing language formalism are more likely to cover content coverage but often fail to model routing aspects.

In particular the popular hierarchical approach to modelling routing functionality has one or more draw backs along the lines of ability to facilitate questionnaire logic validation, ease of understanding by domain experts and flexibility to enable refinements to questionnaires.

Accordingly we introduce the Survey State Model (SSM) XML language based on a state-transition model to address these shortcomings. We present our results from testing SSM on a sample of real-world surveys from Pexel Research Services in the UK. We use the distribution of SSM's vocabulary on this sample to demonstrate SSM's applicability and its coverage of questionnaire constructs and effective routing support.

Keywords: XML, XSD, SCH, authoring, survey, questionnaire, CAI, hierarchical model, state-transition model

1. Introduction

Surveys are the systematic collection of information from individuals or organizations to address research and business objectives [1]. Questionnaires are one of the instruments of surveys utilised to collect data considered as structured interviews.

The Computer Assisted Interviewing (CAI) systems allow the design, collection, management, analysis and reporting of surveys through computers and they should have a specification language addressed to describe every feature presented in questionnaires.

XML was designed to represent structured documents and as such may be used to formally represent every requirement from questionnaires. These requirements are imposed by the designers of surveys who decide the order in which the questions are shown to the respondent as well as the possibility to be asked or not based on respondent previous answers. The use of XML to describe questionnaire's specification may reduce the need of programmers to implement questions order as well as complicated logical decisions since the creation of intuitive interfaces can generate questionnaire's requirements easily in XML. Also, the exchangeability inherent in XML allows the design of questionnaires be circulated among different Computer Assisted Interviewing (CAI) systems without hardware or software restrictions.

Pexel Research Services carries out a large number of surveys through telephone every year based on client's specification. As such they use a Computer Assisted Interviewing (CAI) solution to conduct surveys but they are keen to consider alternative solutions which may offer commercial advantages over existing systems.

The rest of this paper is structured as follows: Section 2 explains what questionnaires are and the features that may appear on them. Section 3 reviews the different approaches based on XML aimed to represent questionnaires. Section 4 present the desired criteria when modelling routing of questionnaires and explains the most popular approach used to model the routing of questionnaires. Section 5 presents our alternative solution to describe surveys based on state-transition model in which XML examples are provided. Finally, Section 6 summarizes the results obtained after testing our approach based on real surveys provided by Pexel Research services.

2. Electronic questionnaires

A questionnaire is one of the instruments of surveys to collect information from people or organizations. A paper questionnaire contains a set of *questions* addressed to the interviewees and *instructions* for interviewers which allow skipping over questions or even directly jumping to the end of the questionnaire [4] based on interviewee responses.

Table 1. Paper questionnaire

INF1. This is an example survey to demonstrate the features that can appear in electronic questionnaires.
Q1. How often do you use your car?

01. Never **GOTO END**

02. Almost never **GOTO END**

03. Occasionally/Sometimes **GOTO END**

04. Amost every time

05. Every time

Q2. Which brands are you aware of? **[FIRST SPONTANEOUS MENTION]**

01. A

02. B

03. C

04. D

05. E

06. F

07. G

08. H

99. Don't know **GOTO END**

Q3. Which brands are you aware of? **[OTHER SPONTANEOUS MENTIONS Q2]**

Q4. Using a scale 1 to 5 where; 5=essential, 4=very important, 3=quite important, 2=relatively unimportant and 1=not at all important. How important are the following safety features when you want to buy a car?

01. Stable body shell

02. Pre-tensioned and load-limited seatbelts

03. Good head restraints

04. Seat-mounted side airbags

05. Side curtain airbags

06. Knee airbags

[IF 'F' IS SELECTED IN Q2 OR Q3 OTHERWISE GOTO END]

Q5. How many cars have you had or have of F brand?

[REPEAT Q6a FOR EACH CAR]

Q6a. Were you satisfied of the safety features?

01. Yes

02. No

03. Don't remember

[IF SATISFIED FOR EVERY CAR OF 'F']

INF2. We are happy that you always like our safety features and we hope you consider us again for future purchases.

END. THANKS AND CLOSE

Generally a questionnaire is divided into *sections* where the presence of *intro* sentences to introduce or end a section becomes important to locate the respondent. This example includes an outer section for INF1, Q1, Q2, Q3, Q4, Q5, INF2 and an inner section for Q6a.

Table 1 presents a small questionnaire that nevertheless demonstrates a variety of common constructs ranging from simple to complex semantics. The most common types of questions are *single-response*, *multiple-response* and *open-ended* (e.g. Q1, Q3, Q5 respectively). Whilst grid (e.g. Q4), unlike the common constructs differs in the manner in which a respondent chooses the responses and remains more cognitively demanding on the interviewee [6].

The instructions, in bold font, are normally needed to manage questionnaire routing according to responses from the interviewee. There are three such routing constructs in this example:

- *skip* feature attached over responses in Q1 or Q2, known as unconditional skips, as well as conditional skips linked in the Q5.
- *filter* constructs, based on a logical expression involving the responses to one or more questions. They are represented using if-then-else statement, for instance the instructions attached over Q5, INF2.
- *loop* feature, allowing the execution of a part of the questionnaire a number of times. The instruction over Q6a permits the execution of that question as many times as the respondent had/has cars of F brand.

In addition to the constructs discussed above there are several additional features in *electronic questionnaires* that are not present in paper questionnaires:

- *Piping* which allows the retrieval of an answer from a previous question as part of the text for another or the automatic generation of responses based on a expression (e.g. Q3 responses are generated automatically according to the responses non-selected in Q2).
- *Computation* that constitutes the execution of an arithmetical expression and its assignment over a variable referenced. Usually it is used to communicate data among sections. For example, after Q6a, if the respondent was satisfied with the safety features, the addition and assignament over a variable could be used in the logical expression preceding INF2.
- *Check* which involves the satisfaction of a logical expression notifying the respondent to solve the inconsistency if it is not fulfilled. For instance, imagine a

scenario in which it was asked: "Qx. do you use a car to go working?" and the respondent replied yes, after it was asked "Qy. how much do spend on petrol?" and the respondent answered zero. This construct might create a logical expression "check Qy is greather that cero if Qx is yes" *warning* or *stopping* the flow of the questionnaire until the inconsistency was solved.

3. Related work

A Computer Assisted Interviewing (CAI) system should have a specification language addressed to describe the features of questionnaires at a much higher level than a programming language in which these constructs will eventually be manipulated. As such, the authoring languages Computer-Assisted Survey Execution System (CASES) [7] and BLAISE [8] provide this level of abstract language specification covering a substantial number of questionnaire constructs. However they each use a proprietary representation language which constrains its adoption and wide scale usage.

In more recent work XML based formal representation of questionnaires has increased in popularity. Table 2 summarizes for each language in the literature the set of questionnaire features that are being addressed. First, *content* which contains the possible types of questions that may appear over questionnaires as well as how these may be grouped. Secondly, *routing* which eliminates the need to follow questionnaire intructions manually. Usually, the routing is described through boolean expressions (e.g. skip, filter, loop and check) but may use arithmetical expressions (e.g. computation) to guide the respondent through the questionnaire and finally *personalise* to adapt the survey to the respondent and to create dynamic adaptation of content at run-time. In this category, protecting a respondent's privacy through the *randomising* of responses to a question to reduce bias evasive responses [15], or *rotating* constructs are examples widely used in surveys.

The content category is well represented in each language explored because they cover the three type of questions most used (single, multiple and open) as well as section. However, Survey Interchange Standard (Triple-S) [2] and Questionnaire Definition Language (QDL) [9] do not contain intro questions and none of them provide the possibility to represent grid questions. Despite the fact that grid could be replaced through several single or multiple questions, we have decided to include this construct as a new question type because it is widely used and covered by Azzara in the design of questionnaires [1].

The routing is partially covered in Survey Interchange Standard (Triple-S) in the form of simple filters based on logical question (e.g. question with Yes/No answers). Simple Survey System (SSS) and Structured Questionnaire Building Language (SQBL) offer filter and loop constructs but they do not implement the skip feature. This could be because this feature can be reversed and use filters instead [3] or because a questionnaire designed without skip constructs is easier to modify, share

Table 2. Features of electronic surveys

Category	Feature	Triple-S	SSS	SQBL	QDL
Content	Section	✓	✓	✓	✓
	Intro	✗	✓	✓	✗
	Single-response	✓	✓	✓	✓
	Multiple-response	✓	✓	✓	✓
	Open-ended	✓	✓	✓	✓
	Grid	✗	✗	✗	✗
Routing	Skip	✗	✗	✗	✓
	Filter	✗	✓	✓	✓
	Loop	✗	✓	✓	✓
	Check	✗	✗	✗	✓
	Computation	✗	✓	✗	✓
Personalisation	Piping	✗	✗	Partial	✗
	Randomising	✗	✗	Partial	✗
	Rotating	✗	✗	✗	✗

and understand [10]. In regard to Questionnaire Definition Language (QDL), it is the only candidate able to represent every routing feature, however its expressions are typed in infix mode involving the use of parenthesis that result in complicated expressions that are prone to error and harder to process.

The personalisation features are only partially covered by Structured Questionnaire Building Language (SQBL) although it is less able to generate automatic responses from previous answers to single/ multiple questions and does not offer random order or rotating of responses to a question.

There are different ways to model the routing that questionnaires follow like flowcharts or the use of graph theory principles but the hierarchical model is the most popular among the languages that we have explored.

The hierarchical model is the approach used to describe the flow of questionnaires in Simple Survey System (SSS) and Structured Questionnaire Building Language (SQBL) and we have studied them in order to address the following questions:

- Does the hierarchical model lend itself to questionnaire design?

- How can features in a questionnaire be represented in an XML authoring language?

Other language approaches such as Survey Interchange Standard (Triple-S) is not considered in our comparative analysis because it does not offer a modelling approach or in the case of Questionnaire Definition Language (QDL) which is not supported any longer.

4. Hierarchical routing model

A language for questionnaire should provide a means to express both content features as well as vocabulary to express the routing of questionnaires. The latter is particularly challenging as it requires the modelling of sophisticated logic which dynamically impacts the relevance of questions given one or more responses to previous questions.

So what are the criteria that must be considered when creating a representation to model question routing?

- *pre-test*: A pre-test is a formal review of a questionnaire [5] aimed at discovering problematic questions and rewriting them to improve understanding. This is expected to lead to improved collection of responses. For instance in a pre-test one would expect to discover any logical inconsistencies in the routing. Thus, a model in which testing does not become cumbersome is beneficial.
- *matching design-model*: The designers of questionnaires specify routing through skip constructs or filters and having a model to support both features should be useful in many questionnaire design projects [13].
- *adaptability*: It is frequent to introduce changes after the questionnaire implementation in the Computer Assisted Interviewing (CAI) system, so a model in which the changes are easy to make would be desirable.

There are variety of modelling approaches that have been proposed for questionnaire routing management. Flow-charts which are used in programming languages not surprisingly has also been used to develop and understand questionnaires [11]. Accordingly the applicability of graph theoretic relationships has been studied by Fagan and Greenberg [14] in the context of questionnaires with particular focus on modelling skip logic that allows skipping over questions based on responses to previous questions. Generally questions, can be modelled as vertices, and skip constructs, modelled as edges that direct the source and destination questions. However, to the best of our knowledge such approaches to modelling routing behaviour have not been developed in to a formal XML representation.

A more promising approach to routing behaviour modelling can be seen in the hierarchical model, used in Simple Survey System (SSS) and Structured Questionnaire Building Language (SQBL) (see Figure 1). Here boxes represent questions and filters are diamonds. The logic behind this approach is a tree that presents advantages like: each question has one and only one path to determine its reachability. The tracing through directed edges between parent-child relationships also allows to determine all circumstances under which a question can be executed [10].

In particular the hierarchical model enables easy pre-testing of a questionnaire as it allows exploration of one or more paths from the start node to any selected end node. For instance, in order to test Q4 isolated, four previous questions must be reached (INF1, Q1, Q2, Q3) and three filters have to be true.

In regard to matching design-model, this alternative does not offer skip features which requires the negation of the logic when questionnaires are described using these constructs. Despite the fact that questionnaires could be described using filters only, the designers of questionnaires are usually non-programmers and they continue to use skips. This means that although the hierarchical model can be used to model routing behaviour; expecting designers to interpret any skip logic as filters is likely to be demanding. For example, the unconditional skips attached over the responses 01, 02, 03 of Q1 represented with a filter would be NOT(Q1 EQ '01' OR Q1 EQ '02' OR Q1 EQ '03) then Q2 else END.

Concerning the adaptability to changes, imagine a simple modification of the survey example in the Table 1 to introduce a new skip over the response 04 (Almost every time) in Q1. This scenario would require another filter to determine whether this response was selected or not as well as the duplication of the question Q4 since the other Q4 modelled is not directly reachable due to the logic inherited from a tree.

5. Survey State Model (SSM) solution

We propose a state-transition based modelling approach to better address the routing requirements involving pre-testing, designing and adaptability criteria.

The state-transition model, depicted in Figure 2, describes the questionnaire presented in Table 1. This model contains various types of states that are linked through transitions to form state-models. This approach has a set of initial states addressed to decide what state is executed first (e.g. filled circle pointing to INF1 or Q6a) as well as a group of states indicating the ending and represented by filled circles with a white outline. Each state-model includes a set of variables representing different types of question (e.g. Q1, Q2, Q3, Q4, Q5, Q6a are variables which describe questions and their values stored) or fields used for computations constructs (e.g. SATISFIED describing an integer number of cars in which the respondent is happy with the safety features). Every transition connects a source with a target state and allows the description of every possible route through the questionnaire. Transtions with decision states involving a source is depicted as diamond. Such nodes are selected when a boolean expression is satisfied. We propose the use of Reverse Polish Notation (RPN) [16] for all expressions involving variables and any constants that appear in our state model. This approach is considered faster than infix or prefix

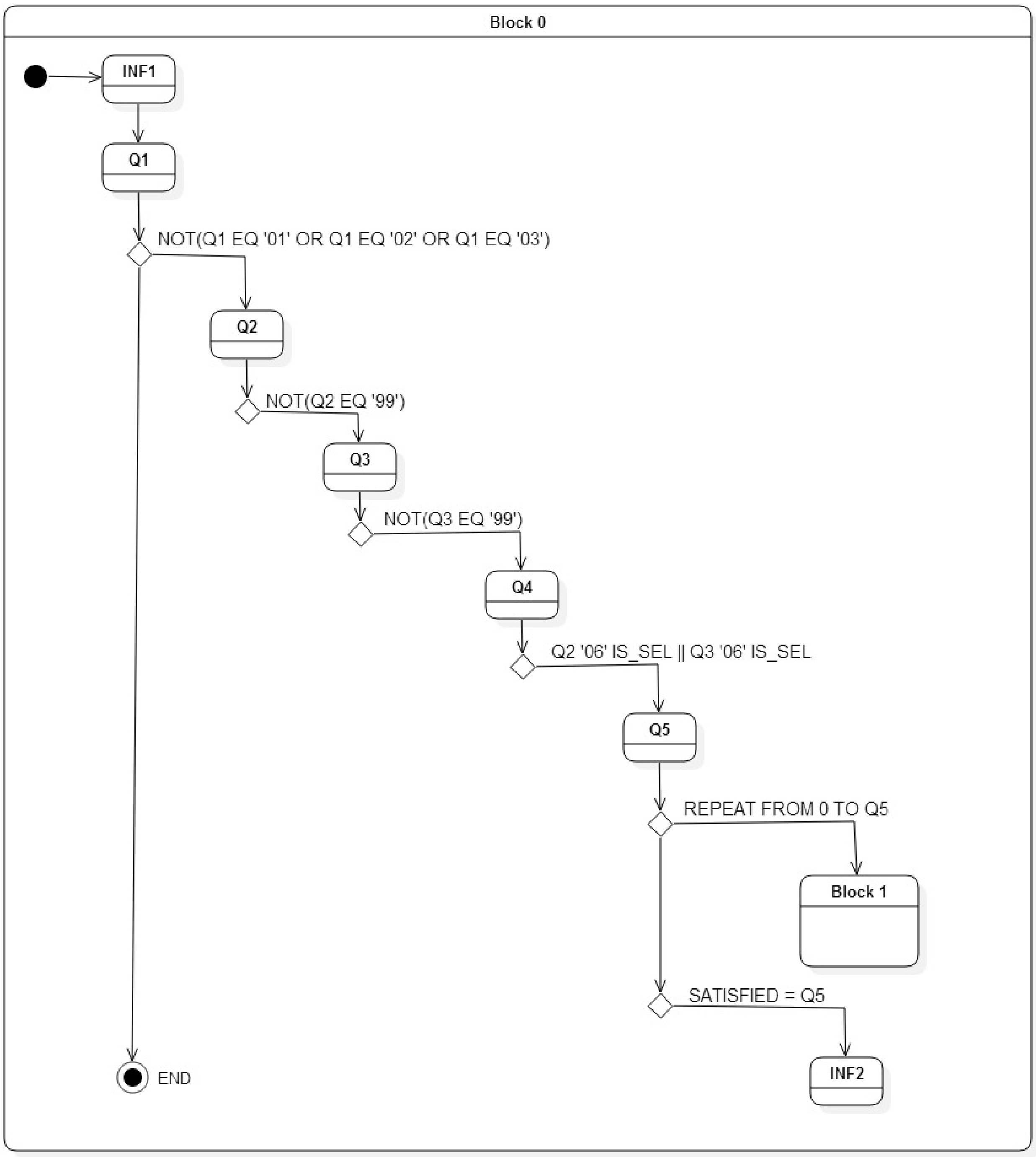


Figure 1. Hierarchical model

notations because the expressions do not need parenthesis and fewer operations are required to solve the expressions. There are several expressions implemented in the example like [Q1, '01', IS_SEL, Q1, '02', IS_SEL, OR, Q1, '03', IS_SEL, OR] addressed to decide whether ending the questionnaire or continuining with Q2.

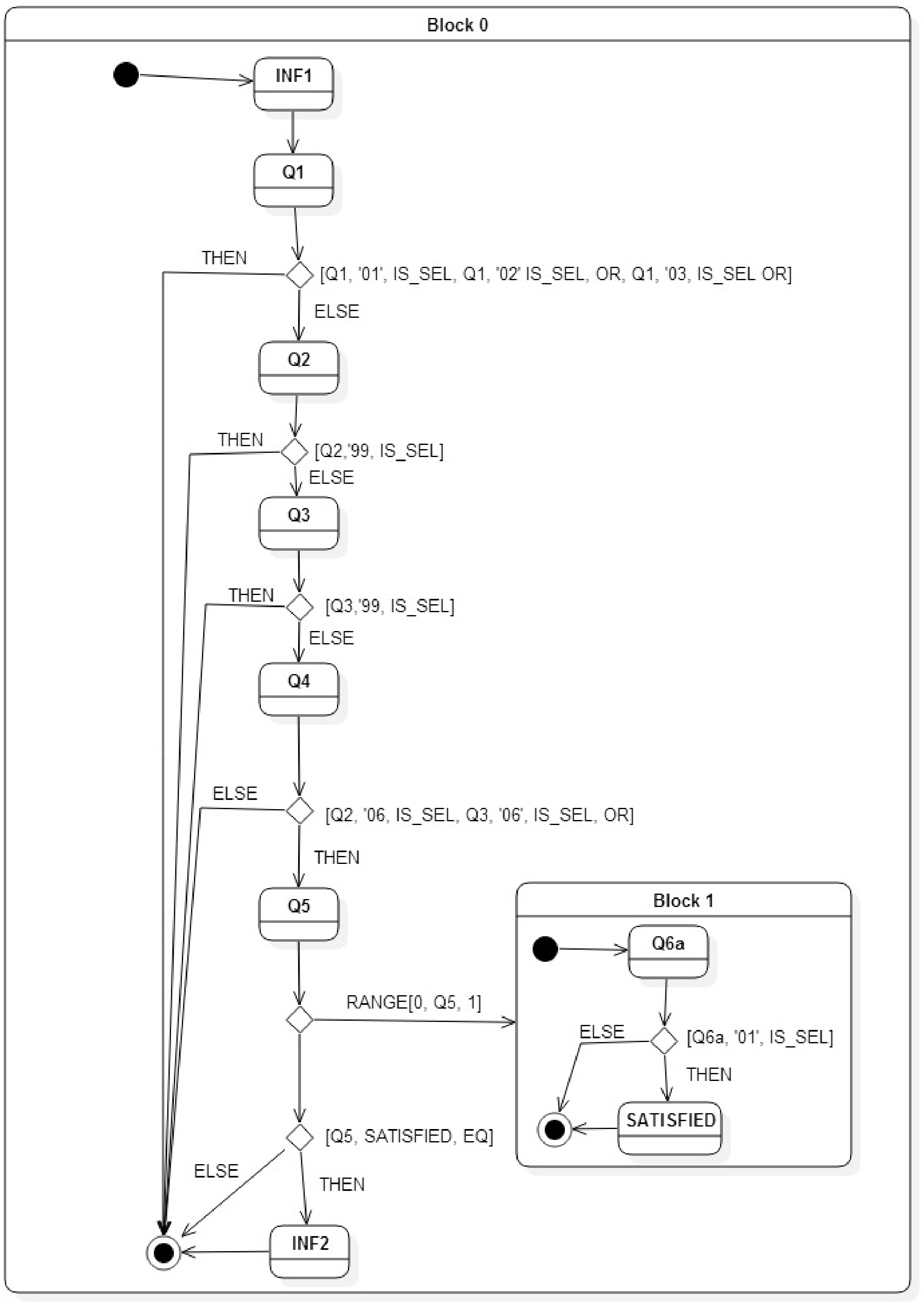


Figure 2. State-transition model

Accordingly we formalise the state-transtion model that is applied to questionnaire routing as follows:

$$M = \langle Q, V, T, I, E \rangle \tag{1}$$

where,

1. $Q \neq \varnothing$ is a finite set of states. These states are: simple, composite, if-then-else, for, check, computation and sink.
2. V is the set of variables. Every variable V may be accessed at every non-sink state $q \in Q$.
3. T is a finite set of transitions.

   A transition $t \in T$ is represented as $q \xrightarrow{[c]} q'$, where $\{q, q'\} \in Q$ and c is a boolean expression involving variables of V and/or constants. The absence of c is interpreted to *true*.
4. $I \subset Q$ is the set of initial or source states. These states determine which state is the first to be reached for a state-model defined.
5. $E \subset Q$ is the set of end states. These states determine which state is the last to be reached for a state-model defined.

5.1. States of Survey State Model (SSM)

In addition to meeting the routing criteria discussed in the previous section, a questionnaire modelling language should also have a good coverage of the different types of questionnaire routing logic. In the proposed Survey State Model (SSM) this translates to the different types of states that can be modelled. These are discussed next with reference to the question types illustrated in Figure 2. Additionally, each state explained ends with an XML code according to SSM XML authoring language.

- *Simple* state is responsible for retrieving the content of the variable (i.e. the definition of the question as well as the response stored, if any). This state allows retrieving one or more variables simultaneously which means the Computer Assisted Interviewing (CAI) system should present on the screen every variable referenced. For instance, Q1 state references the single-response variable Q1.

```
<state id="Q1">
    <variable ref="Q1"/>
    <transition target="p0"/> <!-- p0 is the decision state
after Q1 -->
</state>
```

- *Composite* state permits to switch and reuse the state-model referenced. It could be seen as functions in structured programming in which the code under that function is executed. For instance, the block 1 is called as many times as the range defined after Q5 is true.

```
<state id="c1"> <!-- c1 is the state pointed by the
                transition RANGE[0, Q5, 1]-->
    <include statemodel="block1"/>
</state>
```

- *If-then-else* state represents the filter and skip features of questionnaires. It is composed of a boolean expression in Reverse Polish Notation (RPN) and permits describing two transitions *then* and *else* for true and false result of the expression respectively. For instance, the diamond between Q1 and Q2 implements the logic for the unconditional skips attached over the responses of Q1. Due to the transition principle of this approach there is no difference between skip or filter. The next XML example code decides whether skipping to the END of the questionnaire or continuing with the Q2. The expression in Reverse Polish Notation (RPN) is [Q1, '01', IS_SEL, Q1, '02', IS_SEL, OR, Q1, '03', IS_SEL, OR] witch is translated to five binary expressions because the operators IS_SEL and OR expect two operands.

```
<state id="p0">
    <if>
        <condition>
            <binary>
                <binary>
                    <binary>
                        <variable ref="Q1"/>
                        <constant type="string" value="01"/>
                        <operator name="IS_SEL"/>
                    </binary>
                <binary>
                     <variable ref="Q1"/>
                     <constant type="string" value="02"/>
                     <operator name="IS_SEL"/>
                </binary>
                <operator name="OR"/>
            </binary>
            <binary>
                <variable ref="Q1"/>
                <constant type="string" value="03"/>
                <operator name="IS_SEL"/>
            </binary>
            <operator name="OR"/>
        </binary>
```

```
         </condition>
         <then>
            <transition target="sink0"/>
         </then>
         <else>
            <transition target="Q2"/>
         </else>
      </if>
   </state>
```

- *For* state matches with the loop feature of surveys and has two transitions, one for the true and another for the false result of the boolean expression. This state has start, end and step elements to define the boundaries of the range expression. For example, the diamond after Q5 starts at 0, ends at the number stored in Q5 and increments in steps of 1. A true result switches to the second state-model (e.g. block 1) whereas the false jumps to other state (e.g. the if-then-else state pointing to INF2).

```
<state id="p4">
    <for>
        <field ref="p4_iterator"/>
        <in>
            <range>
                <start><constant type="string" value="0"/></start>
                <end><variable ref="Q5"/></end>
                <step><constant type="string" value="1"/></step>
            </range>
        </in>
        <transition target="c1"/><!-- c1 is the composite
                                      state which includes
                                      block1 -->
    </for>
    <transition target="p5"/><!-- p5 is the decision state
                                  [Q5, SATISFIED, EQ] -->
</state>
```

- *Check* state defines a boolean expression in which the true result shows a message (warning or error) notifying to the respondent the trigger of an inconsistency. The error message is aimed to stop the execution of the state-model until the conflict is solved. The next XML code describes a expression to test whether Qy is greather than zero or not. A true result should show the label *"A car cannot run without fuel"* in warning mode.

```
<state id="py">
    <check type="warning">
        <condition>
            <binary>
```

```
                <variable ref="Qy"/>
                <constant type="integer" value="0"/>
                <operator name="GT"/>
            </binary>
        </condition>
        <label lang="en">A car cannot run without fuel</label>
    </check>
    <transition target="pz"/>
</state>
```

- *Computation* state permits defining an arithmetical expression. For instance, the state after Q6a, named as SATISFIED, describes an arithmetical operation of incrementing the variable by one. The use of this state allows referencing variables out of the scope of an state-model like SATISFIED variable referenced in the if-then-else state pointing to INF2 in the block 0.

```
<state id="p1">
    <computation ref="SATISFIED">
        <assignment>
            <unary>
                <variable ref="SATISFIED"/>
                <operator name="INC"/>
            </unary>
        </assignment>
    </computation>
    <transition target="sink0"/> <!-- sink0 is the sink state
                                  (filled circle with a white
                                   outline)
</state>
```

- *Sink* state is aimed to describe the ending of the state-model. For example, the two state-models, Block 0 and Block 1, contain sink states indicating that no more states are reached after them. As the reader may appreciate, there are no transitions going out from them.

```
<state id="sink0">
    <sink/>
</state>
```

Finally, the *start* state is a property which determines the first state to execute in the set of states from a state-model. As such, the following XML code describes the source state for block 0 and block 1 respectively.

```
<statemodel id="block0">
    <start id="INF1"/>
    <state id="INF">...</state>
    ...
    ...
```

```
        ...
    </statemodel>
    <statemodel id="block1">
        <start id="Q6a"/>
        <state id="Q6a">...</state>
        ...
        ...
        ...
    </statemodel>
```

5.2. Validation of questionnaires

Survey State Model (SSM) XML language implements a two-steps validation process to determine whether an XML document describing a questionnaire is correct or not (see Figure 3). Both steps take as input the *XML document* to be validated against *a set of rules* defined in a formal way to express syntax and/or semantics.

The first step checks the structure, form and syntax in XML Schema Definition (XSD). For instance, *"A section contains a set of questions and these may be intro, single-response, multiple-response, open or grid."*. This process may finish with "yes" involving that the document should be passed to the second step or "no" which reports a JavaScript Object Notation (JSON) document with the errors. If the errors are not fatal the process carries on until the end of the file is detected, otherwise the process stops checking at the line where the error was raised.

The second step examines the relationships among elements through Schematron (SCH). For example, *"Every transition's target must be a defined state in the statemodel"*. Similarly, when a non valid XML file is encountered the process ends by communicating the errors through a JavaScript Object Notation (JSON) file. This process is not able to differentiate between fatal and non-fatal so every error raised is reported.

6. Results

In order to test Survey State Model (SSM) XML authoring language coverage and to know the relevance of every feature of questionnaires we used it to model a set of 15 questionnaire that were sampled by Pexel Research Services Ltd. In particular we studied the frequency distribution of SSM vocabulary over this sample of questionnaires. Figure 4 shows the distribution of SSM vocabulary applied to a sample of 14 test questionnaires sorted by decreasing order of frequency. The graph includes SSM constructs relating to content constructs (section, intro, single, multiple, open, grid), routing behaviour (skip, filter, loop, check, computation) and personalisation constructs (piping, randomise, rotate). It was encouraging to us that all questionnaires in the sample were represented using SSM. Apart from the check and computation constructs, every other feature was included in them.

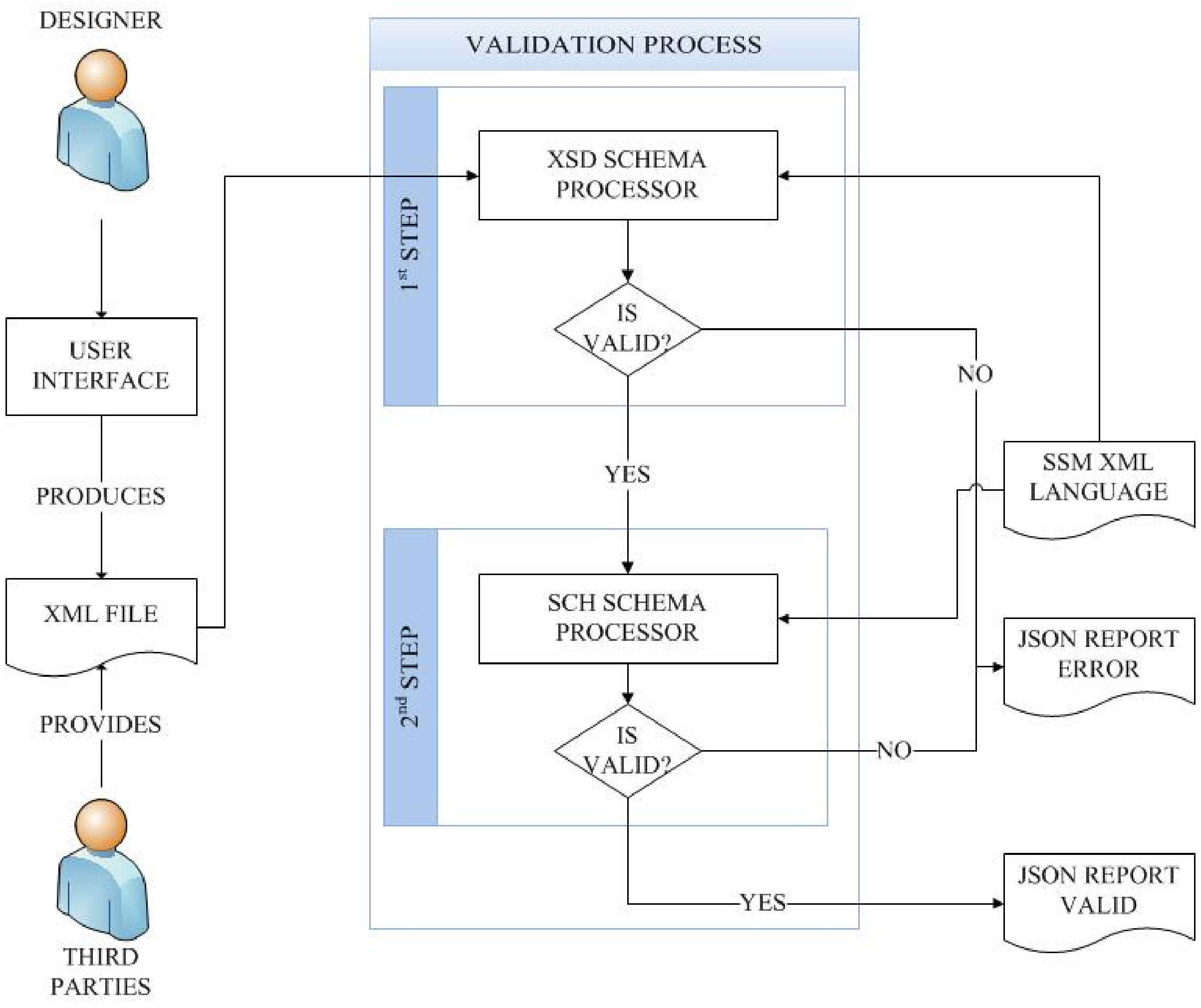


Figure 3. CAI validation process for SSM

Firstly, for the content category, may be affirmed that single-response has the most significant frequency whereas grid question, which unexpectedly has resulted in the lowest important in terms of question types.

Secondly, for the routing classification, the skip feature is frequently most used in our sample of questionnaires even though it is normally best avoided in conventional programming languages [12]. Whilst it is true that programmatically it can lead to non-elegant code constructs, here we conclude that it remains an important feature of questionnaires and so should ideally be facilitated by the underlying language of questionnaires. Computation feature, without any frequency at all, was expected some significance since it becomes essential when a section requires data from other section, however the absence of check does not constitute any surprise since it is a rare feature over questionnaires.

Finally, for the personalising grouping, the three features are presented in the sample tested and piping constitutes the most used specially for generating responses whereas randomising or rotating were less popular.

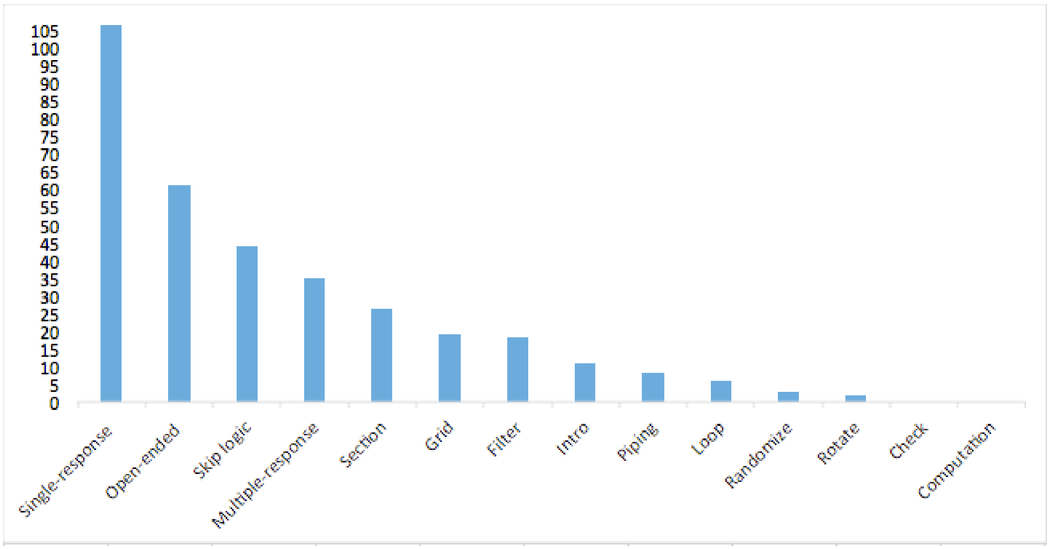


Figure 4. Real surveys frequencies

7. Conclusions

The use of XML to manage the representation of questionnaires for Computer Assisted Interviewing (CAI) systems is not new. We have conducted a detailed comparison of commonly used XML languages in terms of coverage of questionnaire constructs to facilitate representation of content and routing functionality.

Our findings suggest that all existing formalisms to modelling the routing task falls short in terms of one or more of the three key criteria: pre-test, matching design-model and adaptability to changes; similarly none of them covers all constructs that are to be expected in questionnaires.

Accordingly to address this problem we have introduced Survey State Model (SSM) XML language which uses a state-transition model to represent routing. Finally, our results from testing SSM on a set of real-world large-scale surveys suggest that the state-transition model is not only able to represent a wider range of questionnaire constructs but also lends itself to addressing the challenges of the routing task.

The authors would like to thank Bruce Leslie, technical Director of Pexel Research Services, for giving his expertise at every stage of this survey research as well as for selecting the variety of real-world survey samples. Also, we wish to thank to Pexel Research Services because without the funding provided by the company, the research paper would not have been possible.

Bibliography

[1] Carey V. Azzara. *Questionnaire design for business research*. Tate Publishing & Enterprises, LLC.

[2] Laurance Gerrard, Keith Hughes, Steve Jenkins, Ed Ross, and Geoff Wright. *Triple-S XML, The Survey Interchange Standard*. ASC.

[3] Albert D. Bethke. *Representing Procedural Logic in XML*. Journal of Software, Vol. 3, No. 2. February 2008.

[4] Jelke Bethlehem. *The routing structure of questionnaires*. Internation Journal of Market Research, Vol. 42, No. 1. 2000.

[5] IBM Corporation Software Group. *The hows and whys of survey research*. October 2012.

[6] Tim Bock. *Market Research*. http://mktresearch.org/wiki/Main_Page .

[7] University of California at Berkeley. *Computer-Assisted Survey Execution System*. http://cases.berkeley.edu/ .

[8] Statistics Netherlands. *Blaise, Survey software for professionals*. http://www.blaise.com/ .

[9] Jelke Bethlehem and Anco Hundepool. *On the Documentation and Analysis of Electronic Questionnaires*. Statistics Netherlands. 2002.

[10] Samuel Spencer. *A case against the Skip Statement*. 2012.

[11] Thomas B. Jabine. *Flow Charts: A Tool for Developing and Understanding Survey Questionnaires*. Journal of Official Statistics. Vol. 1, No. 2. 1985.

[12] Edsger W. Dijkstra. *A Case against the GO TO Statement*. ACM 11 (1968), 3: 147-148.. 1968.

[13] Irvin Katz, George Mason, Linda Stinson, and Frederick Conrad. *Questionnaire designers versus instrument authors: Bottlenecks in the development of computer-administered questionnaires*.

[14] Jim Fagan and Brian V. Greenberg. *Using graph theory to analyze skip patterns in questionnaires*. Statistical Research Division Bureau of the Census Washington, D.C.. 1988.

[15] Stanley L. Warner. *Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias*. Journal of the American Statistical Association. Vol. 60, No. 309. 1965.

[16] Bob Brown. *Postfix Notation Mini-Lecture*. http://bbrown.spsu.edu/web_lectures/postfix/ .

Glossary

Computer Assisted Interviewing (CAI)

Survey Interchange Standard (Triple-S)

Questionnaire Definition Language (QDL)

Simple Survey System (SSS)

Structured Questionnaire Building Language (SQBL)

Survey State Model (SSM)

Computer-Assisted Survey Execution System (CASES)

XML Schema Definition (XSD)

Schematron (SCH)

JavaScript Object Notation (JSON)

Reverse Polish Notation (RPN)

Schematron for Information Architects

George Bina
Syncro Soft
`<george@oxygenxml.com>`

Abstract

Schematron is a different kind of XML schema language, it focuses not on the grammar of the document but on different rules the structure and the content should follow. It is used successfully in the industry to enforce business rules on XML documents. Although it contains only 21 elements and a few attributes many people that do not have a technical background will be intimidated by the thought of learning Schematron.

"Information architect" is an emerging profession in the XML information domain that defines a role responsible for the overall structure of the information models. Such a person should try to achieve consistent writing styles, structures, and reuse decisions and communicate and enforce the information model to information developers. While information architects will be able understand the business needs they will not necessarily be experts on XSLT, XPath, Schematron and other XML technologies.

As one of the missions of an information architect is to enforce an information model and the use of a consistent structure and style, we can immediately infer that Schematron will be a great tool to master, but we cannot expect these people to become experts in Schematron. During many years of experience with Schematron I discovered that if we follow a set of best practice rules we can make Schematron accessible to anyone, thus enabling information architects to express business rules that will govern the XML information created by XML authors.

We can structure Schematron rules to enable people that are not Schematron experts to create business rules in an easy way and we can take this idea further and build the business rules as part of a style guide, thus single sourcing the prose and the rules to automatically enforce the prose of the style guide. These ideas are materialized as an open source project on GitHub.

Keywords: XML, Schematron, business rules, validation

1. Introduction

There are so many technologies these days around XML - there are at least 5 schema languages (DTD, XML Schema, Relax NG, Schematron, NVDL), at least 3 main processing languages (XSLT, XQuery, XProc), most of these with multiple versions, etc. and it is very difficult to keep up with everything.

Many times I find myself planning to learn a new technology but never getting the time to actually look into that and learn it, so I remain familiar with that technology, have some idea of what it can do but that is all. Usually, once I get the time and start to learn a new technology, after a few a-ha moments I am wondering why I did not lean that earlier. I think I am not alone and this happens to all of us.

In our busy world it becomes important to lower the entry barrier for any technology, so people can be able to use it sooner, without investing much of their time initially and then, as they discover how that technology can help them, they will be willing to invest more time to learn and master it. We try to find this low entry point for Schematron, to get people to use Schematron right away.

2. Business rules

Business Rules are constraints that are not generic enough to be part of a standard but they are relevant for a project or an organization. Here there are some examples:

- do not scale images dynamically in the XML source
- make sure list items do not end with a ";" character
- make sure the number of words in a short description stays within some specific limits

The business rules are usually enforced before the content goes to the application so they sit between validation and application processing. The border is not very clearly defined, some rules can be implemented in the validation layer, especially if we use a customized schema for our project or organization or moved into the application layer if there is no business rules layer available.

One of the main roles for Schematron is to specify this type of checks, so Schematron schemas usually act as the business rules layer between document validation and application.

3. Information architects

One of the information architect role is to identify and specify the business rules that the information should follow. Usually this is documented in some prose form but the question is, how can we make sure that what is written in the prose is actually enforced? And one answer to this is to implement what the prose says as business rules in Schematron. Thus we should be able to automatically apply the Schematron checks against a document and determine if there is any check that fails.

If the information architect identifies and documents the business rules it will be great if he can also write the corresponding Schematron checks that implement those business rules. Thus when he modifies a rule he can also update the Schematron check that implements that rule. We cannot expect however, to have all information architects become Schematron experts over night so we need to think of a way that will allow them to define Schematron rules with a minimal level of Schematron knowledge.

4. Schematron

Schematron is one of the 5 standards that specify schema languages for XML. It is not focused on defining the structure of the documents, like the grammar-based schema languages (DTD, XML Schema and Relax NG) but rather to check different patterns in the documents.

Schematron was invented by Rick Jelliffe in 1999 and since 2006 it is an ISO/IEC standard being part 3 of ISO/IEC 19757 – DSDL (Document Schema Definition Language) Part 3, called "Rule-based validation".

The reference implementation for Schematron is called Skeleton and it is an XSLT based implementation, the Schematron schema being translated to an XSLT script that when applied on the XML document will provide the Schematron validation result.

Along with being tuned for expressing business rules, one of the main advantage of using Schematron is that the error messages are defined by the schema author so they can be written in a language that the writer of the document will easily understand, in contrast with the more generic validation messages provided by a validation against a grammar based schema language.

Usual use cases for Schematron include:

- provide error messages that writers/authors can easily understand
- check rules that cannot be specified in the DTD or XML schema
- check for common mistakes
- controlled values - check against an external data source
- integrity checks across multiple files
- style guide integration
- checks for targeting a specific deliverable, for example if you want to publish for mobile devices

5. Simple use of Schematron

During the last years I presented Schematron at many events, introducing this technology to different users, usually not very technical users. Doing this I received interesting feedback from participants and that feedback pointed positively to the use of abstract patterns.

A specific check is implemented in Schematron as a pattern that contains a set of rules, from which only one will be activated, depending on the context. Each rule contains a number of assertions and reports that are verified in the rule context. To be able to reuse pattern definitions for similar checks, Schematron introduced the notion of abstract patterns, where the pattern definition can contain some parameters that will be provided with specific values whenever the abstract pattern will be instantiated for an actual use. Now, the complexity of the check remains in the abstract pattern definition, while the instantiation of the pattern is very simple, it just points to the abstract pattern name and provides values for its parameters.

Example 1. A Schematron abstract pattern and its instantiations

```
<pattern id="restrictWords" abstract="true">
  <title>
    Check the number of words to be within certain limits
  </title>
  <p>This pattern allows to check that the number of words in an
    element fits between a lower and an upper limit and instructs
    the user to stay within thse limits.</p>
  <p>As parameters we have <emph>parentElement</emph> that
    specifies the element containing the text to be checked,
    <emph>minWords</emph> and <emph>maxWords</emph> that specify
    the minimum and maximum number of words, respectively.</p>
  <rule context="$parentElement">
    <let name="words"
          value="count(tokenize(normalize-space(.), ' '))"/>
    <assert test="$words &lt;= $maxWords" role="warn"
            sqf:fix="restrictWords_setNew"> It is recommended to
      not exceed <value-of select="'$maxWords '"/> words! You
      have <value-of select="$words"/> <value-of
      select="if ($words=1) then ' word' else ' words'"/>.
    </assert>
    <assert test="$words &gt;= $minWords" role="warn"
            sqf:fix="restrictWords_setNew"> It is recommended to
      have at least <value-of select="'$minWords '"/> words! You
      have <value-of select="$words"/> <value-of
      select="if ($words=1) then ' word' else ' words'"/>.
    </assert>
  </rule>
</pattern>

<pattern is-a="restrictWords" id="restrictDesctription">
  <param name="parentElement" value="shortdesc"/>
  <param name="minWords" value="3"/>
  <param name="maxWords" value="50"/>
</pattern>

<pattern is-a="restrictWords" id="restrictAbstract">
  <param name="parentElement" value="abstract"/>
  <param name="minWords" value="10"/>
  <param name="maxWords" value="100"/>
</pattern>
```

As can be seen in the example, the instantiations done by the `restrictDesctription` and `restrictAbstract` patterns are very simple, their complexity derives only from the semantics of the expected parameters, the user only needs to provide an element name and two numbers that will represent the minimum and maximum number of words to be present in that element. Anyone can learn the syntax for this in a few minutes and then they can create specific patterns by instantiating the available abstract patterns.

If we structure the use of Schematron by defining a library containing a set of abstract patterns that implement generic rules, then we can create actual Schematron schemas just by instantiating those generic patterns by referring to a pattern name and specifying values for its parameters. We can define this as a best practice and that will split the roles of using Schematron in 3 parts:

1. schematron developer
   - defines a library of abstract patterns
   - needs to know Schematron
2. information architect
   - instantiates abstract patterns to define business rules
   - needs to know only what rules are available and their parameters
3. document writer/author
   - will be notified as he/she changes the document, as soon as an check fails

The information architect will identify business rules and will try to enforce them automatically with Schematron. For that he will look into the library of available generic rules to see if there is an abstract pattern that can be used to implement that business rule. If such a pattern exists then he will just use it, otherwise he will make a request for a new abstract pattern and define its generic processing and what parameters should be provided. The Schematron developer will implement the new abstract pattern and when that is ready the information architect will be able to use it to implement the business rules. The document writer role remains the same, he will just observe different notification messages that will identify potential issues in the created information content.

The abstract patterns can be self documenting, we can use annotations with enough information to describe the use cases for each generic rule and the meaning of the expected parameters.

6. Integrating Schematron rules within a style guide

As an information architect identifies different rules and documents them in prose as part of a style guide (guidelines, best practices, etc.) he will be able to instantiate also generic patterns (assuming we have a library of abstract patterns as described in the previous section) to implement an automatic enforcement of the prose of the style guide. This, however, keeps the prose and the rules separated and they may easily get out of sync.

If we look again at the instantiations of abstract patterns, we have in fact very little information there and we should be able to easily encode that directly in the style guide, if that uses a structured format like DITA, DocBook, etc. This will allow us to single source the prose and the automatic checks in the same document, thus reducing the cases when they get out of sync.

Because the style guide encodes all the information for a pattern instantiation it means that we can process the style guide source and create the Schematron schema containing the pattern instantiations.

To enable a better authoring experience, we can customize the editing environment to be aware of this specific encoding and make it easier to insert new business rules and provide a specific rendering for them.

7. Implementation

These ideas are implemented in an open source project called Dynamic Information Model (DIM). This project provides one possible implementation of these ideas based on a DITA encoded style guide, but other implementation may provide similar support for DocBook or any other structured markup that can be used to define the style guide.

The DIM project was initiated by Syncro Soft and Comtech Services and it is made available under Apache 2.0 license on GitHub at http://www.githug.com/oxygenxml/dim.

It contains a starter DITA style guide donated by Comqtech Services, defined as a DITA map and a set of topics. Schematron patterns that instantiate abstract patterns from a library are encoded using data lists (`dl` elements) placed in a section marked for a specific audience attribute. An XSLT script is used to extract the actual Schematron schema from the DITA source.

In addition to these, a customization of the oXygen XML Editor editing environment is provided to offer a visual editor to easily create new patterns and discover what each pattern does and what each parameter represents.

7.1. Encoding Schematron patterns in DITA

There are multiple possibilities to encode an instantiation of an abstract pattern in DITA. They fall mainly in two parts - you either define specialized DITA markup or define a specific pattern of existing markup to encode the information. We decided to use the second approach to make it easy to specify Schematron patterns in any DITA document, thus we do not have a requirement for the style guide topics to be only from a specific set of specialized topic types.

We use a data list to specify the abstract pattern instantiations, we specify the parameters and their values using the data term and data definition, respectively and we specify the name of the abstract pattern using the data definition head. To identify the data lists that hold Schematron pattern instantiations we require them to be placed in a section marked with an `audience` attribute set to the value "`styleguide`". Thus, the pattern instantiation:

```
<pattern is-a="restrictWords">
  <param name="parentElement" value="shortdesc"/>
  <param name="minWords" value="3"/>
  <param name="maxWords" value="50"/>
</pattern>
```

will be encoded in DITA as:

```
<section audience="rules">
  <dl>
    <dlhead>
      <dthd>Rule</dthd><ddhd>restrictWords</ddhd>
    </dlhead>
    <dlentry><dt>parentElement</dt><dd>shortdesc</dd></dlentry>
    <dlentry><dt>minWords</dt><dd>3</dd></dlentry>
    <dlentry><dt>maxWords</dt><dd>50</dd></dlentry>
  </dl>
</section>
```

Having the patterns defined in sections marked with an `audience` profiling attribute allows us to easily exclude or include these sections when we publish the style guide.

7.2. Extracting Schematron from DITA sources

An XSLT script is provided that when applied on a DITA map that represents the style guide will follow all maps and topics referred in the style guide and will identify all the data lists that represent encoded abstract pattern instantiations converting them to the schematron patterns.

The XSLT script will also read the Schematron library and it will generate include statements for all the abstract patterns defined in the library.

7.3. Schematron library of abstract patterns

The Schematron library can be organized either as one file per abstract pattern or we can group multiple patterns in the same schema file. The later approach however will require using an experimental functionality to allow inclusion of a specific element from a Schematron schema using a fragment identifier in the URL that will point to the ID of the to be included element. We took the second approach taking into account that this functionality is already supported in the Skeleton implementation and that it makes it easier for processing to access the patterns from the same file rather than looking for multiple files in a folder.

Example 2. Including an abstract pattern

```
<include href="library.sch#restrictWords"/>
```

`restrictWord` is the ID of an abstract pattern defined in the `library.sch` file.

In order to make the abstract patterns self documenting, they can be annotated with information to contain a title and a description that will help a user understand which use cases that pattern can solve and what parameters are available. To enable automatic processing of the abstract patterns we annotated each parameter using elements in a specific namespace. These annotations allow for easy detection of what parameters are available for each abstract pattern. Such a detection is also possible without these annotations but it may be more difficult if we take into account that abstract parameters and variables have the same representation (`$parameterName`) and that may appear also in a string literal, etc. As these abstract patterns are added exactly with the scope of being used also by people without a lot of technical knowledge, it seems reasonable to explicitly define the expected parameters and document their meaning to enable creating a better user interface for users that will instantiate them.

Example 3. Parameters annotation

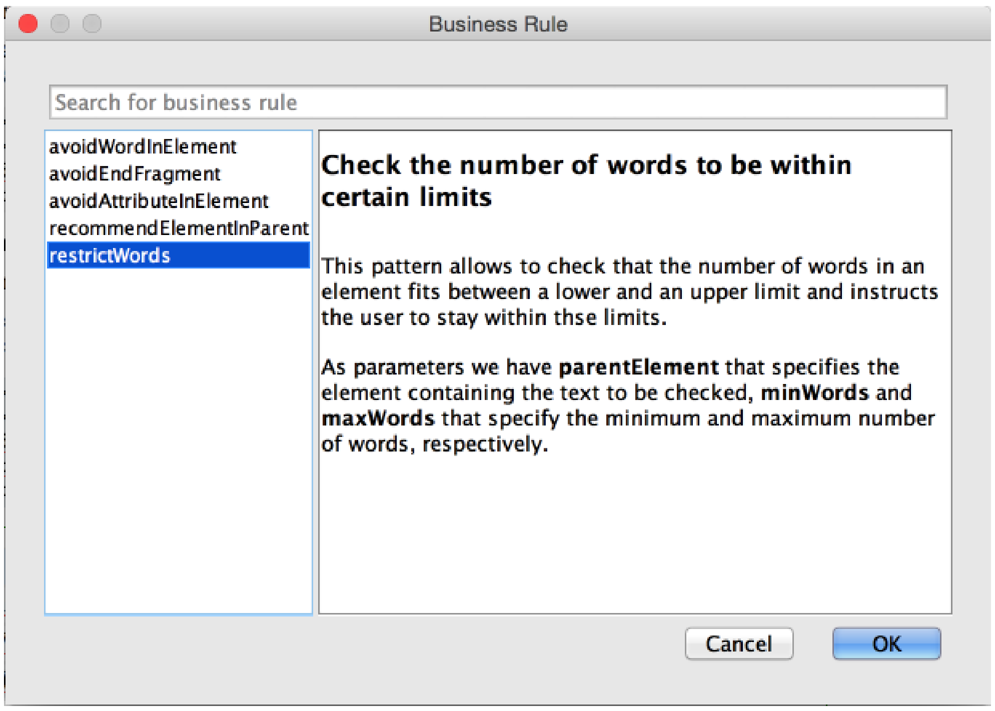
For the `restrictWords` pattern we have the following annotation to describe the abstract pattern parameters and their semantics:

```
<parameters xmlns="http://oxygenxml.com/ns/schematron/params">
  <parameter>
    <name>parentElement</name>
    <desc>
```

```
        Specifies the element who's word number should be counted.
      </desc>
    </parameter>
    <parameter>
      <name>minWords</name>
      <desc>
        Specifies the minimum number of words that is accepted.
      </desc>
    </parameter>
    <parameter>
      <name>maxWords</name>
      <desc>
        Specifies the maximum number of words that is accepted.
      </desc>
    </parameter>
  </parameters>
```

7.4. Easy creation of rules

To allow encoding pattern instantiations easily in DITA, we provide also a visual interface that will lookup all the generic rules from the library (the set of available abstract patterns) and present them to the user to choose the one he wants to instantiate. Here we can take advantage of the annotation information to present the title and the description. A filter is provided to allow finding a generic rule when we have a large number of rules - the filter is applied on the name and also on the annotations provided for the abstract pattern.


Business Rule

Search for business rule

avoidWordInElement
avoidEndFragment
avoidAttributeInElement
recommendElementInParent
restrictWords

Check the number of words to be within certain limits

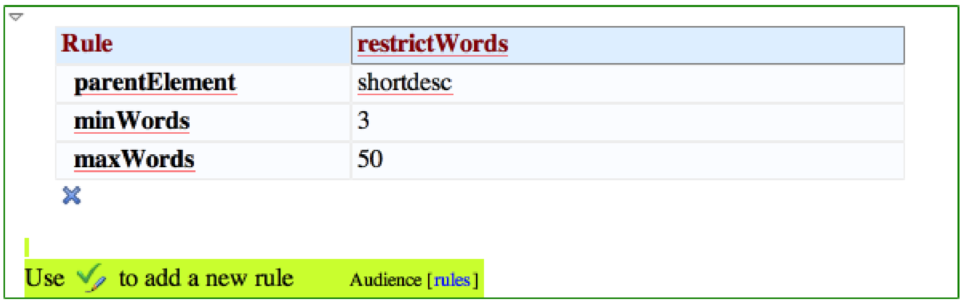
This pattern allows to check that the number of words in an element fits between a lower and an upper limit and instructs the user to stay within thse limits.

As parameters we have **parentElement** that specifies the element containing the text to be checked, **minWords** and **maxWords** that specify the minimum and maximum number of words, respectively.

Cancel OK

Figure 1. The dialog that helps selecting a business rule

The result of selecting a generic rule will be an XML fragment that represents the data list encoding of the pattern instantiation, pointing to the abstract pattern we want to instantiate and containing also the names of the possible parameters already filled in. The user can then just fill in values for each parameter to create a specific rule.



Rule	restrictWords
parentElement	shortdesc
minWords	3
maxWords	50

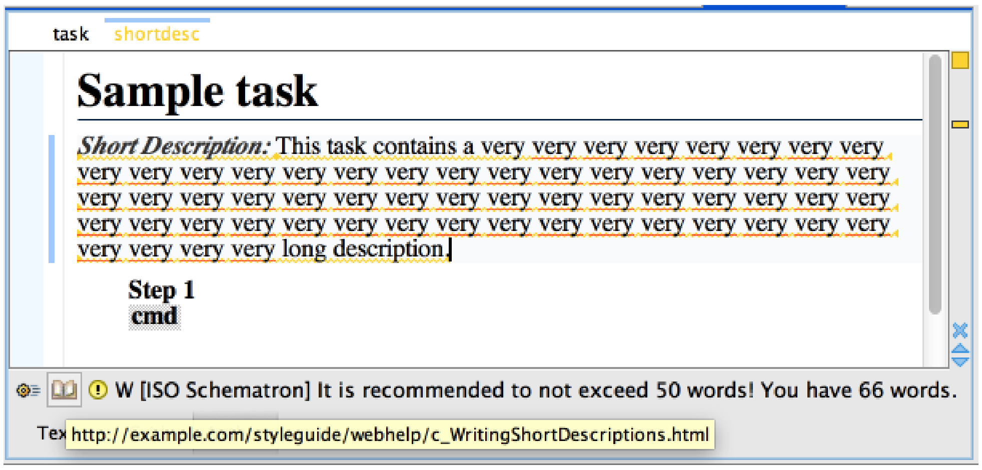
Use to add a new rule Audience [rules]

Figure 2. The visual rendering of a business rule (abstract pattern instantiation)

The UI actions use the oXygen API to insert the XML fragment that encodes the pattern in DITA but a similar API can be used also to create such actions for other XML tools.

8. Deliverables

The main result we are interested in is the Schematron schema that implements the rules defined in the style guide. However, the style guide itself is also a deliverable and it can be published in different formats. The Schematron rules and the style guide are dynamically linked, each reported issue will link to the style guide topic that contained the abstract pattern instantiation corresponding to the schematron pattern that generated that issue.



task shortdesc

Sample task

Short Description: This task contains a very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very long description.

Step 1
cmd

W [ISO Schematron] It is recommended to not exceed 50 words! You have 66 words.

http://example.com/styleguide/webhelp/c_WritingShortDescriptions.html

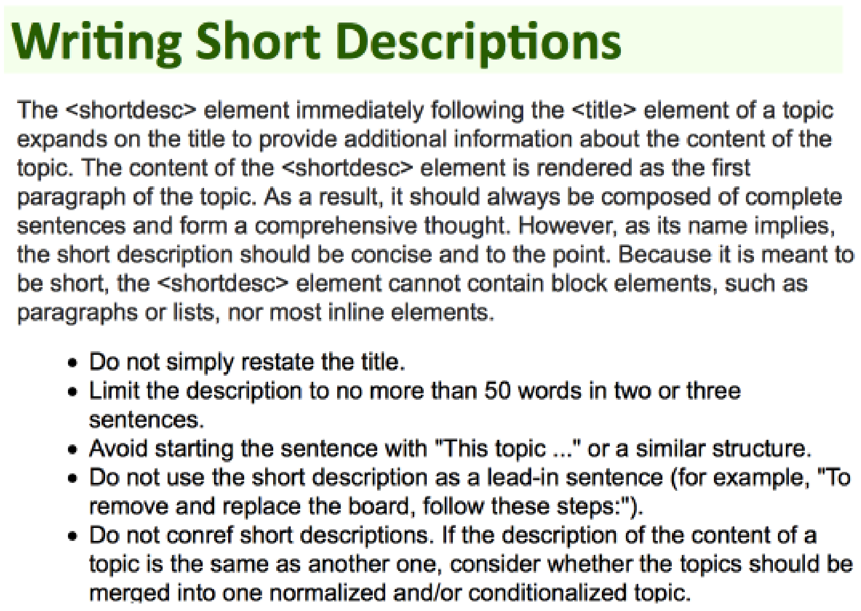
Figure 3. Schematron reported issue and the URL to the style guide it points to

This reported issue points to the style guide topic that contains the abstract pattern instantiations as well as the prose that triggered that business rule

9. Conclusions and future work

By defining a library of abstract patterns we enable people to use Schematron in a very simple way - just instantiate the available generic rules by providing parameters to them, thus lowering the entry barrier for non technical people to start using Schematron. Taking this one step further and defining the patterns inside a style guide, together with helper actions to easily identify available generic rules and to insert the corresponding encoding of the Schematron pattern in the style guide helps to lower the barrier even more and to single source the style guide and the rules that enforce it on documents.

An interesting approach to DITA encoded Schematron patterns will be to implement a custom URL scheme that will return the Schematron rules when the style guide map is accessed though that URL scheme, thus removing the need for conversion to Schematron. Another possibility will be to extend the Schematron implementation and extract the Schematron rules from DITA in a way similar to how the


Writing Short Descriptions

The <shortdesc> element immediately following the <title> element of a topic expands on the title to provide additional information about the content of the topic. The content of the <shortdesc> element is rendered as the first paragraph of the topic. As a result, it should always be composed of complete sentences and form a comprehensive thought. However, as its name implies, the short description should be concise and to the point. Because it is meant to be short, the <shortdesc> element cannot contain block elements, such as paragraphs or lists, nor most inline elements.

- Do not simply restate the title.
- Limit the description to no more than 50 words in two or three sentences.
- Avoid starting the sentence with "This topic ..." or a similar structure.
- Do not use the short description as a lead-in sentence (for example, "To remove and replace the board, follow these steps:").
- Do not conref short descriptions. If the description of the content of a topic is the same as another one, consider whether the topics should be merged into one normalized and/or conditionalized topic.

Figure 4. Style guide entry containing the prose the business rule implements

Skeleton implementation extracts embedded Schematron rules from XML Schema and Relax NG schemas.

We plan to create a comprehensive library of abstract patterns that will be generally available as part of the DIM project trying to decrease the cases when an information architect will need to ask the Schematron developer to create a new abstract pattern (generic rule). Those patterns will be used to instantiate rules in the style guide included in the DIM project.

In interesting development on top of Schematron is the quick fixes support initiated by Nico Kutscherauer. Quick fixes are actions that the user can select to solve a reported problem. We can extend the abstract patters to include also quick fixes actions and thus the authors will be able to select a possible fix to the reported problem.

Bibliography

[1] *Information technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-based validation -- Schematron*. http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip.

[2] *The Dynamic Information Model project*. Comtech Services. Syncro Soft. http://www.github.com/oxygenxml/dim.

[3] *Schematron Quick Fixes*. Nico Kutscherauer. http://www.schematron-quickfix.com/index.html.

TXSTEP – an integrated XML-based scripting language for scholarly text data processing

Wilhelm Ott
University of Tuebingen
<wilhelm.ott@uni-tuebingen.de>
Tobias Ott
pagina publication technologies ltd.
<tobias.ott@pagina-tuebingen.de>

1. Introduction

With TXSTEP, we present and put up to discussion a preliminary version of a new, powerful XML-based tool for scholarly research in the text-based humanities. Its architecture is based on more than 40 years of experience in supporting humanities projects at the University of Tübingen and beyond.

The purpose of TXSTEP is not to provide another toolbox containing ready-made solutions for pre-defined problems. Of course, tools like these are adequate for many purposes; but we see no urgency to add a further one to the existing packages of this kind.

In fact, TXSTEP has been designed as a high performing scripting environment for the serious humanities scholar and other professionals in text data processing who face problems not easily solvable by XSLT or other means. TXSTEP gives them complete control over every detail of the data processing part of their projects.

TXSTEPs architecture is based on the Tübingen System of Text Processing tools, TUSTEP, whose current version is the result of more than 40 years of experience in supporting humanities projects at the University of Tübingen and beyond.

2. Humanities software: basic requirements

Software for serious humanities research has to have certain basic qualities:

- it must be easy to handle, so that the scholar who is an expert in his field, but not in programming or computer science, can use it safely;
- it must be flexible enough to be adapted to the special requirements of each project, be it a philological analysis of a text or the preparation of a critical edition;
- it should support not only single phases of a project, but all its stages and steps, including (for an editorial project) a first transcription of the sources and the

collation of the transcribed texts, the evaluation of the variant readings, the constitution of the edition text and the critical apparatus, up to (and even beyond) the preparation of the final publication of text, apparatuses, and indexes.

TXSTEP tries to satisfy these somewhat contradicting requirements by taking into account the fundamental operations necessary for the scholarly processing of textual data, and by providing a separate program module for each of these basic functions.

3. The solution: 1. Modularity

These modules may be combined almost arbitrarily: each module reads from and writes to a single basic file structure. This allows to combine these modules like Unix filters in arbitrary ways, giving the system the flexibility which is required for scholarly research and which allows to perform work not explicitely foreseen by the developers.

Where necessary, the single modules can be adapted to special requirements by the user, who may change default parameters (e.g. for providing a sort key for a non-latin alphabet) or provide additional ones (e.g. for the omission of the definite article in the sort key for titles in bibliographic records).

How limited the scope of the single modules is, may be illustrated by the fact that, for example, there is no dedicated program for generating an alphabetical word list. For this purpose, the user who wants to prepare a script for this purpose, e.g. for an index to a greek text, has to combine the module for text decomposition (for which he has to provide the parameters defining the single elements and the sort keys), the SORT module, and the module which reduces identical or partially identical records contained in the sorted file to single index entries, and adds - when required - informations like frequency counts and/or references to the source text. For more complex tasks, like the preparation of a lemmatized concordance, other modules may intervene.

The modules provided by TXSTEP include:

- collation of different versions of a text; output of the differences in a synoptic list (for eye inspection) and for automatic processing in a file showing an appropriate structure;
- text correction and enhancement not only by an interactive editor, but also in batch mode, e.g. by means of correction instructions prepared beforehand (by manual transcription, or by program, e.g. the collation module);
- decomposing texts into elements (e.g. word forms) according to rules provided by the user;
- building logical entities (e.g. bibliographic records) consisting of more than one element or line of text;

- sorting such elements or entities according to the sort keys provided by the preparatory modules, accounting also for non-latin alphabetical rules and other sorting criteria;
- preparing indexes by generating entries from the sorted elements;
- transforming textual data by selecting records or elements, by replacing strings or text parts, by rearranging, complementing or abbreviating text parts, by making explicit numerical information implicitely contained in a text;
- integrating external information into a file by means of acronyms;
- updating crossreferences;
- converting textual data from TUSTEP files into file formats used by other systems (e.g. for statistical analysis or for electronic publication) and vice versa.

As the output of any one of these modules may serve as input to any other module, the range of research problems for which this system may be helpful is quite wide.

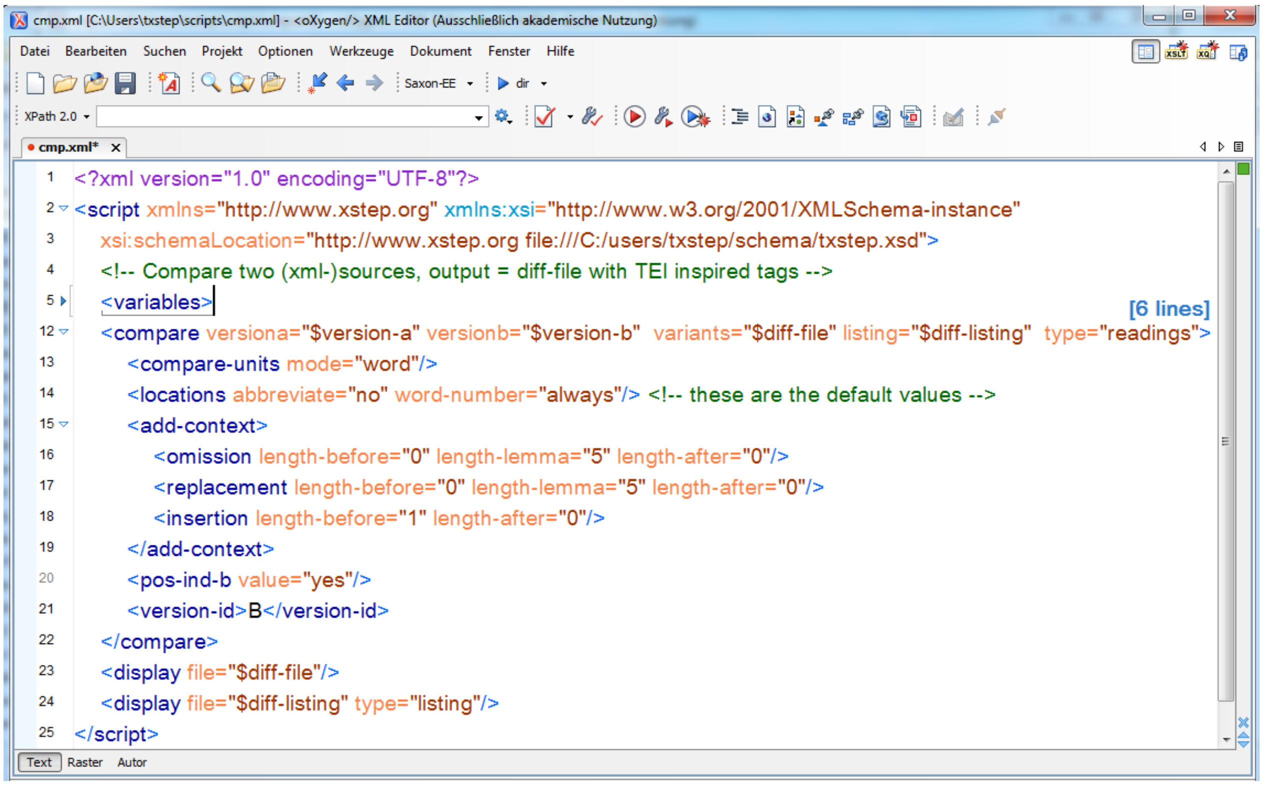
In fact, TUSTEP, the Tübingen System of Text Processig tools, has been developed along these lines. It has long been known as a very powerful set of tools for the processing, analysis and publication of texts, meeting the requirements of scholarly research. It has been and still is successfully used for many humanities projects in the German speaking part of the world, as may be checked by visiting www.tustep.org.

4. The solution: 2. XML interface to an established text processing and analysis suite

At the same time, TUSTEP is said to have a very steep learning curve due to an unfamiliar command-line based user interface, due to a proprietary syntax, and due to the fact that the documentation is avaliable in German only.

TXSTEP breaks down these barriers to the usability of these tools for the growing e-humanities community: it makes them available in a user-friendly XML-syntax, allowing beginners and advanced programmers to utilize the whole scope of TUSTEP services in a modern, established scripting environment.

The benefits are obvious: support of an open standard, widespread dissemination, programming in every XML-editor, syntax highlighting, code completion and intelligible APIs.


cmp.xml [C:\Users\txstep\scripts\cmp.xml] - <oXygen/> XML Editor (Ausschließlich akademische Nutzung)
Datei Bearbeiten Suchen Projekt Optionen Werkzeuge Dokument Fenster Hilfe
Saxon-EE
dir
XPath 2.0
cmp.xml*
Text Raster Autor

```
<?xml version="1.0" encoding="UTF-8"?>
<script xmlns="http://www.xstep.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xstep.org file:///C:/users/txstep/schema/txstep.xsd">
  <!-- Compare two (xml-)sources, output = diff-file with TEI inspired tags -->
  <variables>                                                                    [6 lines]
  <compare versiona="$version-a" versionb="$version-b"  variants="$diff-file" listing="$diff-listing"  type="readings">
    <compare-units mode="word"/>
    <locations abbreviate="no" word-number="always"/> <!-- these are the default values -->
    <add-context>
      <omission length-before="0" length-lemma="5" length-after="0"/>
      <replacement length-before="0" length-lemma="5" length-after="0"/>
      <insertion length-before="1" length-after="0"/>
    </add-context>
    <pos-ind-b value="yes"/>
    <version-id>B</version-id>
  </compare>
  <display file="$diff-file"/>
  <display file="$diff-listing" type="listing"/>
</script>
```

Figure 1. Comparing two (xml-)sources with TXSTEP, generating a diff-file as output

Compared to the original TUSTEP command language, TXSTEP

- offers an up-to-date and established syntax
- allows to draft respective scripts using the same XML-editor as when writing XSLT or other XML based scripts
- lets you enjoy the typical benefits of working with an XML editor, like content completion, highlighting, showing annotations, and, of course, verifying your code
- offers - to a certain degree - a self teaching environment by commenting on the scope of every step.
- helps to avoid many syntactical errors, even compared to the original TUSTEP scripting environment.

One of the features of TXSTEP is its capability to process almost all forms of textual data, whether this being XML-data or plain text files. Therefore, even if textual data have to be processed in the first place in order to gain, for example, TEI-data or to enhance the markup of insufficiently tagged XML data, TXSTEP is at its place.

In an early stage of its development, TXSTEP has been subjected to a closer examination by Michael Sperberg-McQueen regarding its overall goal and design, the syntax and structure of the XML command language, including details of naming and style, operating system dependencies, and its positioning within the XML software ecosystem. His critics and proposals - and his very encouraging final remarks - have been very helpful for the further work on the system in the past three years.

Since then, the work on the project resulted in an operational version of TXSTEP which already contains the most important features of all the modules of TUSTEP listed above.

One of the features of TXSTEP is its capability to process almost all forms of textual data, whether this being XML-data or plain text files. Therefore, even if textual data have to be processed in the first place in order to gain, for example, TEI-data or to enhance the markup of insufficiently tagged XML data, TXSTEP is at its place.

The proposed paper will include a short demo of TXSTEPs functionality, showing solutions also for tasks which can not easily be performed by existing XML tools.

TUSTEP and TXSTEP are open source software under the Revised BSD License and are available for download from www.tustep.org

In-Browser XML Document Streaming

Cyril Concolato
Institut Mines-Télécom; Telecom ParisTech; CNRS LTCI
<cyril.concolato@telecom-paristech.fr>

Emmanouil Potetsianakis
Institut Mines-Télécom; Telecom ParisTech; CNRS LTCI
<emmanouil.potetsianakis@telecom-paristech.fr>

Abstract

Through the past few years, in-browser streaming of audiovisual content has become a commodity. Due to the diverse nature of possible audiovisual applications, there is often a need for accompanying descriptors and other metadata, such as semantic annotations, captions, etc. The metadata need to be sent in a timely fashion along with the multimedia content. The use of XML in such cases is common, and the usual approach for in-browser transmission of such data is via AJAX. Even though AJAX can be sufficient for many services, there is consideration for few offline or live scenarios. With MP4Box and MP4Box.js we are able to synchronously stream and consume XML and multimedia data, packaged in MP4 containers, with a standard browser. Accompanying XML documents can be transmitted as a whole, or progressively (in fragments). In this paper, we define the use-cases for this technology, analyze the requirements and present the mechanisms of MP4Box and MP4Box.js for XML end-to-end transmission inside the browser.

Keywords: XML, Streaming, Multimedia

1. Introduction

With the bloom of HTML5 and its `<video>` and `<audio>` tags, audiovisual (A/V) content transmission and presentation has become an essential functionality of the modern browser. In 2013, IP video traffic accounted for the 66% of the total internet traffic, and this number is expected to reach 79% by 2018, according to CISCO [1]. This growth affects XML technologies, since an important use of XML is for storing media information, subtitles, lyrics and other audiovisual content enhancements. The aforementioned complementary XML documents have to be transmitted and processed in-browser - synchronously with the main A/V content. However, such a technology, with support for both online (live and on-demand) and offline content, is yet to be widely spread.

Using the `<video>` element allows easy integration of A/V content, with common web technologies such as HTML, CSS and Javascript. Specifically in the case of Javascript, HTML5 defines an API consisting of Methods, Properties and Events, aiming at A/V content control. However, there are several ways to deliver the actual media to the client. As a result, the transmission of the accompanying XML documents and their proper handling on the receiving end becomes challenging.

In this paper we propose an end-to-end approach for streaming XML documents in the browser. The multimedia content along with the accompanying XML document(s) are packaged in a MP4 container. Then, they are transmitted synchronously, and the received content is made available by the browser to the web application, either live, or for offline consumption.

Synchronized In-Browser XML streaming can be applied to a plethora of use cases. From common scenarios such as subtitling and vector graphic enhancements, to more exotic, such as distributed Kinect-based applications for multimedia control [2] or digital puppetry [3].

In Section 2, we present some past and current trends in XML streaming. Followed in Section 3 by a description of our proposal for packaging XML in MP4 files. Then in Section 4 and Section 5, we detail the implementation of the packaging process with MP4Box, and the extraction mechanism in MP4Box.js, respectively. Finally in Section 6, we conclude this paper and reveal our current work on the topic.

2. Related Work

The Remote Events for XML [4] W3C Draft was produced with the purpose of DOM events transmission. The syntax defined could have been used for document streaming, besides the fact that it was focused on DOM tree enhancements. However, it did not include any notions of media synchronization or packaging. In the end, the draft was dropped due to legal implications.

Niedermeier et. al. developed a technique for compression and streaming of XML Data [6]. This schema-aware method is part of the MPEG-7 standard. XML documents are run through a binary encoding algorithm and the resulting fragments are transmitted. It also includes a text-encoding method. MPEG-7 streams can be stored in MP4 containers. But neither of the stream formats is deployed in browsers yet.

Apple HTTP Live Streaming (HLS) uses MPEG-2 Transport Streams (TS) for media packaging and ID3 Tags for metadata [5]. However, the metadata is not in an XML format, nor easily extractable within a browser. The WebVTT format also used by HLS for subtitles could facilitate timed XML data delivery, but WebVTT cannot be packaged in a single MPEG-2 TS file with the A/V content, making offline processing and content distribution difficult.

A web development paradigm that can be used for in-browser metadata is Asynchronous JavaScript and XML (AJAX). More specifically, in the Comet model of AJAX, a persistent connection is established between the server and the client. By using the open request made for the connection, it is up to the server to set the event to commence the data transfer (Long-Polling technique). When the server has available data, the client receives a Push Notification. Even though AJAX has some advantages comparing to the other alternatives, it can only be used for online services (not for offline). Our implementation on the other hand, is suitable for any consumption mode - even offline.

3. The Mechanism

As mentioned in Section 1, the XML documents are packaged with the accompanying media file(s) in an MP4 container (the process is detailed in Section 4). Then, the client browser is responsible for requesting the MP4 file from the server. The playback can be achieved via *Simple Streaming*, in which we have a Progressive Download of the file, and it is the use-case of plain `<video>` tag. Alternatively, we can have *Adaptive Streaming* by utilizing the extra functionalities provided by the use of Media Source Extensions (MSE).

In respect to these media delivery methods, there are two ways XML data can be streamed:

- *One XML document per time (or time range).* In this scenario, an entire XML document is timely delivered, for consumption within a time range of the audiovisual content. This method can introduce some latency with sizable files, since it requires the browser to fetch the whole XML document in order to use it, or it may cause some overhead if the XML documents are repetitive. However, it is simple and applicable to many use cases. An example usage is the carriage of [9] subtitles in MP4 files.
- *One XML Section (of a document) per time (or time range).* In this scenario, fragments of an XML document are coupled with a time range of the audiovisual content. Fragments are delivered progressively, removing the latency of the previous approach, but requiring a progressive consumption of the XML data. In particular, since the reception of the XML document is continuous and in fragments, we cannot have a balanced XML at a given time. To remove the overhead of the previous approach, common data is delivered upfront in a document header. Finally, seeking into the document stream can be more complex than with the previous scenario. It is only possible at positions in the document that require only the header information and nothing between the header and the current position. Such position is called a Random Access Point (RAP). The storage in MP4 permits indications on where the RAPs are located in the stream, thus -in conjunction with the header stored specifically- allowing seeking.

Both of the aforementioned delivery methods can be realized with our platform, which is composed of two parts, and detailed below:

- a server side packaging solution (MP4Box)
- a client side browser tool (MP4Box.js)

3.1. MP4Box

In order to provide a complete end-to-end XML streaming solution, MP4Box can be used for the packaging of the A/V and XML content in MP4 containers (on the server side). MP4Box is part of the GPAC multimedia framework [8].

An example of XML document streaming would be Timed Text Markup Language (TTML) subtitling [9]. In live scenarios, with real-time subtitle editing, several XML documents can be packed in the A/V stream. For such cases, Timed Text support was added to the ISO Base Media File Format (ISOBMFF) MPEG-4 Part 30.

MP4Box achieves any XML stream integration, in a similar with TTML manner, by using NHML descriptor files as follows:

```
MP4Box -add test_file.nhml:lang=en media_file.mp4
```

The NHML file details the integration of the XML documents in the MP4 track. For the command mentioned before, we can use a file as the one shown in Figure 1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<NHNTStream version="1.0" timeScale="1000" trackID="1"
            mediaType="meta" mediaSubType="metx"
            xml_namespace="http://example.namespace.org">
   <NHNTSample DTS="0" isRAP="yes" mediaFile="first.xml"/>
   <NHNTSample DTS="10000" isRAP="yes" mediaFile="second.xml"/>
   <NHNTSample DTS="20000" isRAP="yes" mediaFile="last.xml" duration="10000"/>
</NHNTStream>
```

Figure 1. Sample NHML file for packaging of multiple documents

With the above specifics, the metadata is inserted in an MP4 track with ID "1". This track will contain media of type 'meta' (i.e. metadata) and subtype 'metx' (i.e. in XML format). With this 'metx' configuration, we have one XML document per sample, as indicated in Figure 2, with here 3 samples made of 3 different documents. Other supported mediaTypes for XML packaging are shown in Table 1. The 'xml_namespace', is the namespace of the packaged XML document. The start time (measured in timeScale units) of each sample is is given by the DTS attribute, set to 0 for the first one. If multiple samples are used, for each sample, the DTS difference between two consecutive samples is used to compute its duration, but the last sample should have the duration attribute set. All samples are marked as RAP since they contain a self-contained XML document.

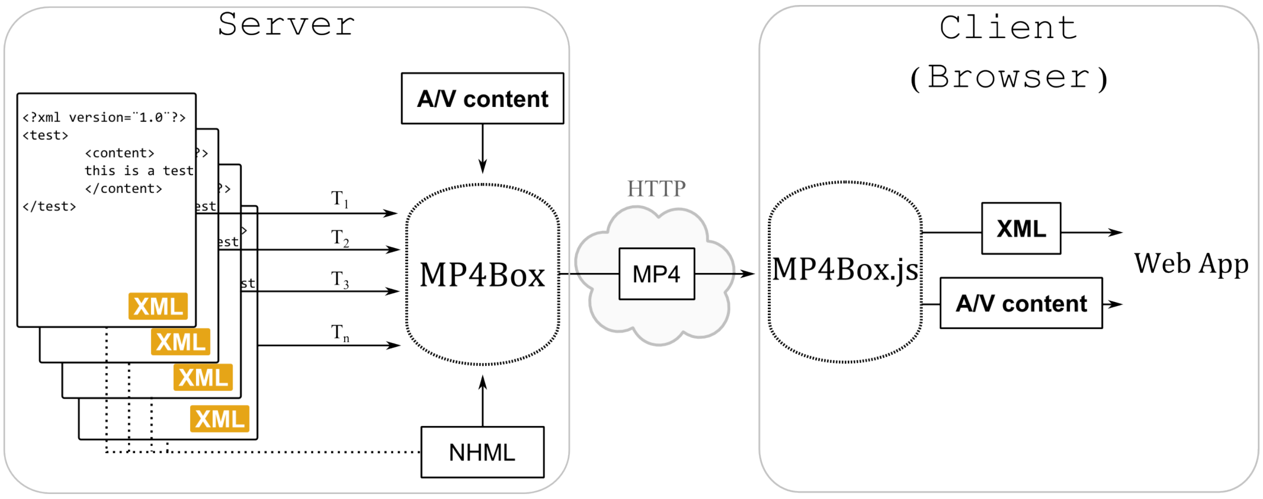


Figure 2. Streaming of multiple XML documents

Table 1. XML-suitable mediaTypes and mediaSubTypes

media-Type	mediaSub-Type	Definition	Usage
meta	metx	XML Metadata	One XML document per sample
	mett	Text (including XML) Metadata	XML documents and fragments
subt	stpp	XML Subtitle	One XML document per sample
	sbtt	Text (including XML) Subtitle	XML documents and fragments
text	stxt	Text (including XML) Stream	XML documents and fragments

NHML also considers the fragmented packaging of a XML document, for progressive consumption (Figure 3). The desired fragments are defined either in terms of document element tags (referenced by their "id" or "xml:id" attributes), or sample position and size. For the tag approach, the 'xmlFrom' field indicates the location of the first tag to copy from the document, while 'xmlTo' the last one. Alternatively, if we want to define the fragment in position and size, 'dataLength' defines its size, and 'mediaOffset' the position of the first byte. The 'isRAP' field is used to specify if a fragment is suitable for RAP. In order to achieve seeking with RAPs, the header of the document must be declared. Figure 4, shows an example use of NHML for progressive XML streaming, using XML elements (tags) selection.

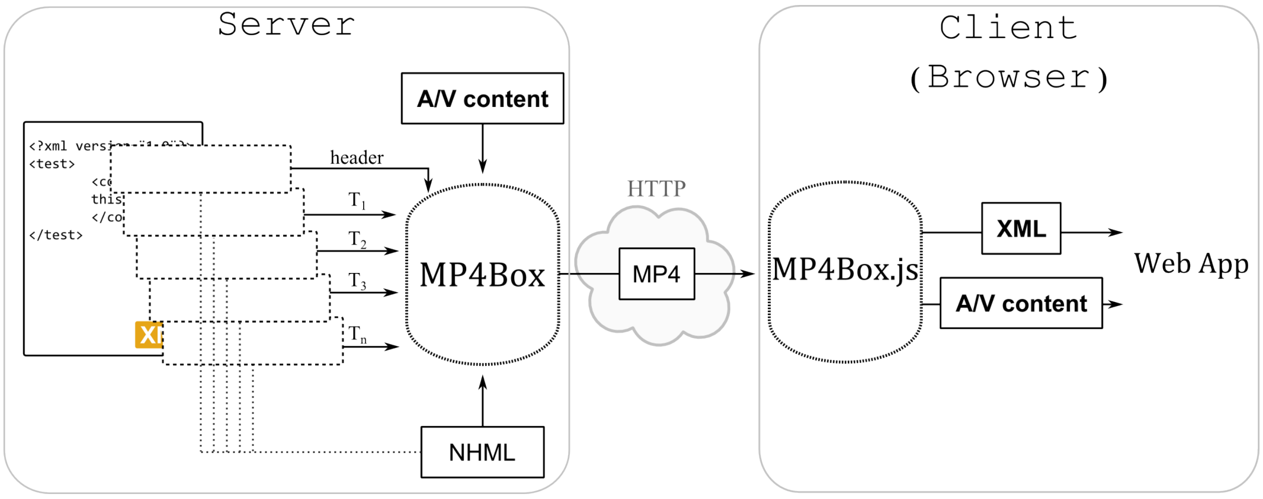


Figure 3. Progressive streaming of single XML document

```
<?xml version="1.0" encoding="UTF-8" ?>
<NHNTStream version="1.0" timeScale="1000" trackID="1"
            mediaType="meta" mediaSubType="mett"
            mediaFile="document.xml" headerEnd="elt1.start"/>
   <NHNTSample DTS="0"    isRAP="yes" xmlFrom="elt1.start" xmlTo="elt1.end"/>
   <NHNTSample DTS="1000" isRAP="no"  xmlFrom="elt1.end"   xmlTo="elt3.end"/>
   <NHNTSample DTS="2000" isRAP="yes" xmlFrom="elt3.end"   xmlTo="elt10.end"/>
</NHNTStream>
```

Figure 4. Sample NHML file for fragmented packaging

3.2. MP4Box.js

In order to achieve the proposed XML document transmission method, on the client side, we developed a tool that is able to extract the accompanying data from the MP4 container. Since the XML fetching happens inside the browser, *MP4Box.js* was developed in javascript. This way, it can be integrated in any web application.

MP4Box.js decouples the accompanying XML documents from the A/V stream and utilizes the `<track>` HTML element to synchronize (and possibly render) the data. The `<track>` element is used inside the `<video>` or `<audio>` tag, as a child element, and holds timed text data. The pre-defined data types that `<track>` can host are set in the "kind" attribute, which can take values of: subtitles, captions, descriptions, chapters or metadata. Metadata is a special case, since there is no predefined rendering and accompanying scripts can utilize the data as desired, by using cue events for synchronization.

An example usage of MP4Box.js setup in order to parse XML samples is shown in Figure 5. An MP4Box instance is created and with `setExtractionOptions`, its parameters are the track "id" and the "user" parameter for the `onSample` callback - typically a `TextTrack` object. The "options" parameter is used to define if the sample array should start with a RAP sample and the total number of samples to receive for the callback. In turn, `onSample` returns an Array of samples, for track with "id", when called by "user".

```
mp4box = new MP4Box();
mp4box.setExtractionOptions(id, aTextTrack, options);
mp4box.onSamples = function (id, user, samples) {
 var sample;
 var parser;
    for(var i in samples) {
      sample = samples[i];
      if(samples.description.type === "metx") {
        parser = new XMLSubtitlein4Parser();
        var sampleDocument = parser.parseSample(sample).document;
         user.addCue(transformDocToCue(sample, sampleDocument));
      }
      else if(sample. description.type == "mett") {
        parser = new Textin4Parser();
         var sampleText = parser.parseString(sample);
         user.addCue(transformTextToCue(sample, sampleText));
      }
    }
}
```

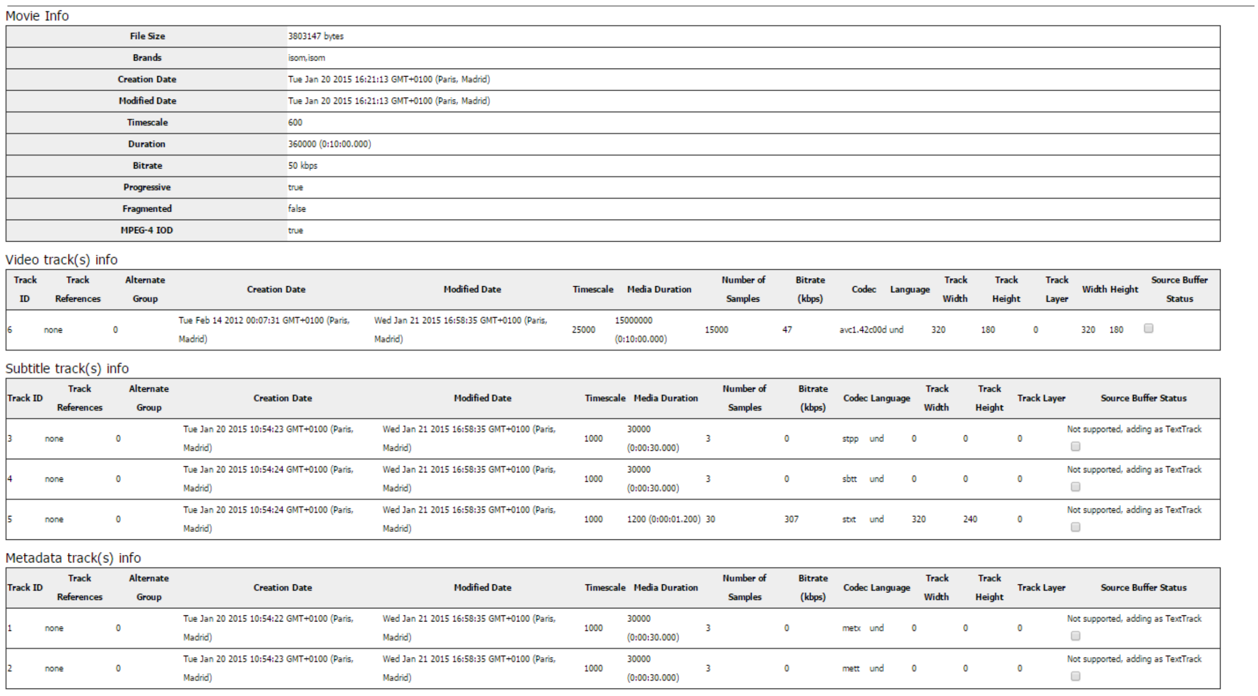
Figure 5. MP4Box.js code extract

Each sample that is extracted from the stream contains the XML data and the packaging information (timestamps, isRAP, etc) - set with MP4Box via the NHML descriptors. The actual XML data is stored in the field "data" of the sample, as an ArrayBuffer. Figure 6 shows the extracted first sample (XML document), as defined in Figure 1.

```
{
    "track_id":1,
    "description": [Box],
    "is_rap":true,
    "timescale":1000,
    "dts":0,
    "cts":0,
    "duration":1000,
    "size":41,
    "data": [ArrayBuffer]
}
```

Figure 6. JSON representation of a sample

A screenshot of a webpage containing information on the available tracks of a mp4 file is in Figure 7. MP4Box.js is used to extract the details of one video track and five tracks with XML data - one for each mediaSubType. More information on the usage and features of MP4Box.js can be found at the github repository of GPAC[1].



Movie Info

File Size	3803147 bytes
Brands	isom,isom
Creation Date	Tue Jan 20 2015 16:21:13 GMT+0100 (Paris, Madrid)
Modified Date	Tue Jan 20 2015 16:21:13 GMT+0100 (Paris, Madrid)
Timescale	600
Duration	360000 (0:10:00.000)
Bitrate	50 kbps
Progressive	true
Fragmented	false
MPEG-4 IOD	true

Video track(s) info

Track ID	Track References	Alternate Group	Creation Date	Modified Date	Timescale	Media Duration	Number of Samples	Bitrate (kbps)	Codec	Language	Track Width	Track Height	Track Layer	Width	Height	Source Buffer Status
6	none	0	Tue Feb 14 2012 00:07:31 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	25000	15000000 (0:10:00.000)	15000	47	avc1.42c00d	und	320	180	0	320	180	☐

Subtitle track(s) info

Track ID	Track References	Alternate Group	Creation Date	Modified Date	Timescale	Media Duration	Number of Samples	Bitrate (kbps)	Codec	Language	Track Width	Track Height	Track Layer	Source Buffer Status
3	none	0	Tue Jan 20 2015 10:54:23 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	1000	30000 (0:00:30.000)	3	0	stpp	und	0	0	0	Not supported, adding as TextTrack ☐
4	none	0	Tue Jan 20 2015 10:54:24 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	1000	30000 (0:00:30.000)	3	0	sbtt	und	0	0	0	Not supported, adding as TextTrack ☐
5	none	0	Tue Jan 20 2015 10:54:24 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	1000	1200 (0:00:01.200)	30	307	stxt	und	320	240	0	Not supported, adding as TextTrack ☐

Metadata track(s) info

Track ID	Track References	Alternate Group	Creation Date	Modified Date	Timescale	Media Duration	Number of Samples	Bitrate (kbps)	Codec	Language	Track Width	Track Height	Track Layer	Source Buffer Status
1	none	0	Tue Jan 20 2015 10:54:22 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	1000	30000 (0:00:30.000)	3	0	metx	und	0	0	0	Not supported, adding as TextTrack ☐
2	none	0	Tue Jan 20 2015 10:54:23 GMT+0100 (Paris, Madrid)	Wed Jan 21 2015 16:58:35 GMT+0100 (Paris, Madrid)	1000	30000 (0:00:30.000)	3	0	mett	und	0	0	0	Not supported, adding as TextTrack ☐

Figure 7. Stream info output of MP4Box.js

4. Conclusion and Future Work

In this paper we have presented the mechanisms of MP4Box and MP4Box.js. An end-to-end solution for XML document streaming inside the browser. We have explained the process of packaging in mp4 containers on the distribution side, as well as the XML data extraction mechanism for the client.

In the future we plan on adding support for specific XML senarios, such as SVG. SVG files can be currently transmitted as a whole inside an MP4 file. However, there is consideration for progressive streaming of SVG files [10]. In that case, a fragmented file can be progressively transported and rendered.

[1] https://github.com/gpac/mp4box.js/

Bibliography

[1] Cisco Visual Networking Index: Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018. 2014. http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

[2] Potetsianakis, E.; Ksylakis, E.; Triantafyllidis, G.: A Kinect-based framework for better user experience in real-time audiovisual content manipulation. Telecommunications and Multimedia (TEMU), 2014 International Conference on , vol., no., pp.238,242, 28-30 July 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6917767&isnumber=6917722

[3] Alberto Pacheco: Digital Puppet Ver. 0.4. 2014. http://podcast.itch.edu.mx/marioneta/index.v4.waves.html

[4] Berjon, Robin: Remote Events for XML (REX) 1.0. Working Draft, 2006, W3C. http://www.w3.org/TR/rex/

[5] R. Pantos: HTTP Live Streaming. Internet Draft, 2014, IETF. http://tools.ietf.org/html/draft-pantos-http-live-streaming-14/

[6] Niedermeier, U., Heuer, J., Hutter, A., Stechele, W., Kaup, A.: An MPEG-7 tool for compression and streaming of XML data. In Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on (Vol. 1, pp. 521-524). IEEE.

[7] Colwell Aaron, et. al.: Media Source Extensions. Candidate Recommendation, 17 July 2014, W3C http://www.w3.org/TR/2014/CR-media-source-20140717

[8] Le Feuvre, Jean and Concolato, Cyril and Moissinac, Jean-Claude: GPAC: Open Source Multimedia Framework. Proceedings of the 15th International Conference on Multimedia. MULTIMEDIA ’07. New York, NY, USA: ACM, 2007, pp. 1009–1012 http://doi.acm.org/10.1145/1291233.1291452

[9] Adams, Glenn: Timed Text Markup Language 1. Recommendation, 24 September 2013, W3C. http://www.w3.org/TR/ttaf1-dfxp/

[10] Concolato, Cyril: SVG Streaming. Editor's Draft, 04 June 2013, W3C. http://www.w3.org/SVG/modules/streaming/

Jiří Kosek (ed.)

XML Prague 2015
Conference Proceedings

Published by
Ing. Jiří Kosek
Filipka 326
463 23 Oldřichov v Hájích
Czech Republic

PDF was produced from DocBook XML sources
using XSL-FO and XEP.

1st edition

Prague 2015

ISBN 978-80-260-7667-4