STX: Scheme-enabled XSLT processor

[Position Paper]

Kirill Lisovsky
lisovsky@acm.org

ABSTRACT

STX is an XML transformation tool based on XSLT and Scheme which combines a processor for most common XSLT stylesheets and a framework for their extension in Scheme and provides an environment for a general-purpose transformation of XML data. It integrates two functional languages – Scheme and XSLT-like transformation language on the basis of the common data model – SXML.

1. STATUS OF THIS DOCUMENT

This document is a work in progress. It describes some principal ideas and concepts of STX design, and provides a decent coverage of STX features inherited from 1.xx series. Some preliminary description of latest innovations (custom tags, transformation of stylesheets) is already included, but it is NOT fully adjusted to upcoming 2.xx (or development 1.9xxx) versions of STX yet.

Current version of this document is considered as a draft.

2. INTRODUCTION

XSLT is one of the most powerful and actively used technologies of the XML family. It has earned wide recognition and is being actively used in industry, usually as a language for transforming XML documents into another format, often for presentation.

On the other hand, XSLT was never intended as a general-purpose XML transformation language, let alone XML application development language. As it is clearly stated in XSLT Requirements [5], turning XSLT into a general-purpose programming language is explicitly not a goal for its future versions.

XSLT is designed primarily for the kinds of transformations that are required when XSLT is used as part of the XSL stylesheet language [3]. For more sophisticated processing of XML data XSLT is usually used in conjunction with a general-purpose programming language, such as Java or Python, which leads to well-known problem of impedance mismatch due to the differences in programming paradigms and data models.

XSLT may be considered as a functional language, and XML data model is really close to Scheme's S-expressions [10]. Scheme is widely used as an extension language and as an XML-processing language, which makes it a natural candidate for an XSLT extension language.

3. RATIONALE AND OBJECTIVES

XSLT is a Turing complete programming language, and many impressive examples prove its applicability for a number of different tasks. Still the majority of the "real life" applications use a small subset of XSLT for a relatively simple transformation of XML documents. Some of the XSLT tutorials [7] even claims that just a few XSLT commands are sufficient for most practical needs.

STX (Scheme-enabled Transformation of XML data) is an XML transformation system which is intended for extension of typical trivial XSLT stylesheets with a program code in Scheme. In such a system, XSLT-compatible templates describe a presentation, while more sophisticated data transformation, data analysis, or even business logic are expressed in Scheme.

Such an approach makes it possible to reuse customer XSLT skills and protects the investment in presentational XSLT stylesheets and provides advanced data processing and application capabilities which make it possible to implement a complete business solution inside the proposed framework.

STX was originally designed as one of the development tools for an Extranet Customer Information System in the IT department of the Moscow branch of Cargill Enterprises Inc. Eventually it evolved into a general-purpose XML transformation tool that provides the full power of Scheme programming language for complex transformation and business logic implementation and at the same time allows to reuse existing XSLT skills and presentational stylesheets.

Major requirements taken into account for STX design were:

1. Compatibility with a specified subset of XSLT
2. Seamless XSLT/Scheme integration

3. Markup and code separation

4. Limited compatibility of STX stylesheets with regular XSLT processors

4. DATA MODEL

XSLT uses a slightly modified data model of XPath [2] which may be considered as a labeled nodes tree representation of XML Information Set [4].

SXML is an implementation of XML Information Set in the form of S-expressions. It was designed with the goal of effective evaluation of XPath queries in the first place. SXML has formal specification [10] and is accompanied by a stack of major XML-related technologies implemented in Scheme. Two of these technologies, namely an XML parser SSAX and an SXML query language SXPath constitute a core of STX.

Both SXML and XPath data models are based on the XML Information Set represented as a tree-like structure. The correspondence between the representations of XML Infoset Information Items in SXML, XPath and XML is shown in table 1. The numbers in this table are the numbers of corresponding productions in the XML Recommendation and SXML Specification, and section numbers in the XPath recommendation. Similarities between this data models are obvious, and their mutual mapping is straightforward, which simplifies their integration significantly.

SXML has a capability for extension with auxiliary information (an unique ID, a pointer to a parent node, and so on) in so-called “auxiliary lists”. Such information may be redundant, or it might not be a part of XML Information Set but it may be used for performance optimization or other like purposes. For example the availability of direct pointers to parent nodes is important for XSLT processor to achieve faster pattern matching. These pointers may be stored in SXML auxiliary lists, as will be discussed in section 5.2.

Due to these reasons STX is based on the SXML data model.

5. PATTERN MATCHING

Pattern matching in STX is performed in accordance with XSLT Recommendation. Template rules identify the nodes to which they apply by using a pattern, which is an XPath expression. A node matches a pattern if the node is a member of the resulting nodeset of the evaluation of the pattern as XPath expression in context of the node being matched or one of its ancestors. Current STX implementation uses a straightforward algorithm for pattern matching based on SXPath, and employs the capability of SXML to be extended with auxiliary information in so-called “auxiliary lists”.

5.1 SXPath

SXPath is a query language for SXML. It treats a location path as a composite query over an XPath tree or its branch. A single step is a combination of a projection, selection or a transitive closure. Multiple steps are combined via join and union operations [11].

SXPath consists of a set of lower-level predicates, filters, selectors and combinators, and higher-level abbreviated SXPath functions which are implemented in terms of lower-level functions.

STX includes the extended implementation of SXPath [1] which provides two kinds of higher-level SXPath functions. The first one is slightly extended version of original `sxpath` function which represents an XPath expression as a list of steps in the location path which may also include a lambda expression as a converter. This list is translated to a combination of SXPath primitives according to a set of rewriting rules. We will use the term "native" SXPath processor when referring to it later on in this text.

The second one uses textual W3C-compatible representation of XPath expressions, thus we will use the name "textual" SXPath processor for it and `txpath` for its highest-level function. Not unlike the original `sxpath`, `txpath` is a higher-order function which returns a function which has to be applied to nodeset. (The real situation is a little bit more complicated, as where are different variants of `txpath` function, and some of them make use of some additional parameters such as a pointer to the root node, a hash for attributes of type ID, etc.) It is worth a note that the function returned by `txpath` may be used in a list of parameters for native `sxpath`, and these two functions may be combined.

Both these functions may be considered as compilers from abbreviated XPath to SXPath primitives, the only difference is in representation of abbreviated XPath.

Since XSLT uses XPath expressions in the W3C textual representation and STX is designed for conformance with XSLT stylesheets, thus textual SXPath processor is a tool of primary importance in implementation of STX. It transforms a pattern (which is an XPath expression) to a function which evaluates to a nodeset corresponding to this XPath expression when applied to an SXML tree.

Pattern matching in STX is performed by applying `txpath` to a pattern, transforming it into a nodeset selection function, and applying this function sequentially to the matched node and its parents until the matched node is a member of a resulting nodeset or root node is reached. In the first case the node matches a pattern, in the second case it does not.

5.2 Fast access to parent nodes

Since a query for the parent of a given node is a common operation in XSLT matching its effective processing is vital for overall performance of the transformational system.

Due to this reason a pointer to the parent is included in auxiliary lists of SXML nodes and is used by the `sxml:node-parent` function for fast access to a parent node of a given one. Such pointers may be added to the existing SXML tree by using the function `sxml:add-parents` or they may be inserted during XML parsing using customized XML to SXML transformation function of the SSAX parser.

However, inclusion of a raw pointer to a parent node in an S-expression creates circular lists and complicates its processing significantly. Fortunately auxiliary lists in SXML

Infoset Information Item	XML	SXML	XPath Node
Document	Document [1]	<TOP>[1]	Root 5.1
Element	Element [39]	<Element>[2]	Element 5.2
Attribute	Attribute [41]	<attribute>[4]	Attribute 5.3
Processing Instruction	PI [16]	<PI>[6]	Processing Instruction 5.5
Unexpanded Entity Reference	EntityRef [68]	<entity>[8]	No – Entity references are always expanded
Character	CharData [14]	Scheme String	Text 5.7
Comment	Comment [15]	<comment>[7]	Comment 5.6
Namespace	Defined in "Namespaces i XML"	<namespaces>[12]	Namespace 5.4

Table 1: Representation of XML Information Set in SXML, XML and XPath

data may contain information of any type and it is possible to wrap these pointers to parent nodes in procedures of no parameters. Thus, we store a parent pointer as a lambda and its application as a procedure without parameters yields the required pointer.

6. SCHEME-ENABLED TRANSFORMATION OF XML DATA

Seamless XSLT and Scheme integration was the primary objective for the design of STX. It is achieved using a common data model and computational paradigm for both components of STX.

XSLT templates are essentially pure functions – each template defines a fragment of the output tree as a function of a fragment of the input tree and produces no side effects [8].

STX transforms its templates into lists of Scheme functions. In every list, the first function is an `txpath`-generated function which is used for pattern matching as described in section 5.1 (actually, it may be any nodeset + root -¿ nodeset function or SXML node name), the second one is a template transformation function. This transformational function may be described in Scheme (in the case of `stx:template`) as well as in XSLT (in the case of `xsl:template`). In this context, XSLT may be considered as a syntactic sugar for Scheme code it is transformed to.

6.1 STX stylesheets

There are three different representations for STX stylesheets:

1. XML – XSL-style stylesheet represented in XML format which employs a subset of XSLT instructions which is extended with STX elements/attributes
2. SXML – XML stylesheet represented in SXML format.
3. STX – SXML stylesheet where Scheme code is represented as S-expressions, not text.

STX uses XML-¿SXML parser (SSAX) for transformation of stylesheets represented in XML to SXML format.

STX provides a facility (`stx:make-stx-stylesheet` function) for transformation of stylesheets represented in SXML format to STX representation.

In particular, a transformation of supported subset of XSL stylesheets (which are essentially XML) to SXML or STX representation is possible.

A literate programming tool, such as Mole [12], may be used for automated transformation of a plain Scheme code (formatted in accordance with an appropriate formatting style) to a STX stylesheet.

6.2 XSLT-style transformation language

Compatibility with a subset of XSLT was one of the primary objectives in the design of the STX. A target set of legacy XSLT stylesheets was analyzed, and all the XSLT constructions used were separated into two major categories:

1. Commonly used that have to be supported. Such a basic XSLT instructions as `xsl:template`, `xsl:value-of`, `xsl:apply-templates` fell in this category.
2. Less popular and may be represented better in Scheme. This category consists mostly of "programmatic" constructs such as conditional processing instructions, variable and parameter handling.

Thus support of an admittedly small subset of XSLT instructions was sufficient enough in order to meet the immediate requirements. Its implementation in SXML/Scheme is based on a `node-dispatch` function which is based on the idea of `pre-post-order` function from Oleg Kiselyov's SXML-tree-trans library [9]. This function is optimized for pre-order traversal, and uses XSLT-style pattern-matching algorithm based on SXPath instead of trigger symbol matching used by `pre-post-order`. It is necessary to note that Scheme-defined bindings described in the next section are also processed using the same technique.

6.3 Extensions in Scheme

STX provides support for two major kinds of XSLT extensions in Scheme:

1. Scheme code embedded in XSLT-style templates using `stx:eval` elements
2. Template rules for XML data transformation coded in Scheme

6.3.1 *Embedding Scheme code in XSLT-style templates*

STX provides a simple way of including a piece of Scheme code inside a XSLT stylesheet. It evaluates the content of any `stx:eval` element as Scheme code and replace the original tag with the result of evaluation. The code is evaluated in a local environment where a set of variables is bound with pointers to the current template node, the matched XML node, the document root, and the defined global bindings.

This instruction may be used instead of "programmatic" instructions of XSLT, for example:

```
<xsl:template match="delivery">
Delivery weight is:
 <stx:eval><![CDATA[
(* (sxml:num-attr node 'weight)
   (if (equal? (sxml:attr node 'unit) "tonne")
       1000
       1))
 ]]></stx:eval>
</xsl:template>
```

It is necessary to note that since a Scheme code may contain XML-offending characters thus nesting the content of `stx:eval` (and `stx:template` as well) within CDATA section is a good practice.

While `stx:eval` is a handy way to insert some snippets of Scheme code in XSLT stylesheets, it intermingles markup with code and violates the paradigm of their separation. Due to this reason, the employment of templates and functions defined in Scheme is considered as a preferable technique for larger applications.

6.3.2 *Template rules programmed in Scheme*

In addition to usual XSLT-style template STX provides a means to define a template in Scheme. Such a template may be included in stylesheets using `stx:template` instruction whose semantics and syntax are similar to those of `xsl:template` but whose content is Scheme code. This code is considered as the body of a `lambda` expression which is evaluated with respect to the same bindings as a content of `stx:eval` instruction. This function is applied to a node which is matched by the pattern of its template rule in the similar fashion with a regular XSLT-style templates.

A template programmed in Scheme may be also named using `name` attribute and used as a function in other XSLT-style or Scheme-coded templates. They may be invocated using `xsl:call-template` instruction or custom tags, as it will be considered in section 7.

6.3.3 *Loading external Scheme code*

STX provides a way to load Scheme code from an external file, to load external STX stylesheet from an XML file or to load a set of templates defined in Scheme from an external textual file. In the example

```
<stx:load href="srfi-1.scm"/>
<xsl:import href="summary.xsl"/>
<stx:import href="rate.sxml" type="sxml"/>
<stx:import href="pnl.stx" type="stx" prefix="pnl"/>
```

a library of Scheme functions is loaded from file `srfi-1.scm`, an external XSLT/STX stylesheet represented in XML format is imported from file `summary.xsl`, an external XSLT/STX stylesheet represented in SXML format (probably generated from XSLT/STX stylesheet) is imported from file `rate.sxml`, and a stylesheet in STX format is imported from file `pnl.stx`.

Scheme functions loaded with `stx:load` are accessible for Scheme code contained in `stx:eval` and `stx:templates`, including those imported with `xsl:import` and `stx:import` instruction.

6.3.4 *Functions as first-class objects*

XSLT may be considered as a pure functional language, but it has a serious limitation: in XSLT functions are not considered as first-class objects. This limitation is especially cruel for an XML transformation language because it dramatically reduces the class of implementable tree-processing algorithms. Naturally, there is no such a limitation in Scheme which makes Scheme-coded functions and templates much more effective and flexible in this application domain.

6.4 Processing of STX stylesheets with XSLT processors

STX stylesheets may be processed by regular XSLT processors. Such possibility may be useful for designers of HTML-extensive presentational stylesheets without significant visual elements described in Scheme. A design of such a stylesheet may be previewed using an XSL-enabled Web browser.

If the STX namespace prefix is designated as an extension namespace by using the `extension-element-prefixes` attribute on an `xsl:stylesheet` element, then all elements with such a prefix are treated as instructions rather than as literal result elements.

As a regular XSLT processor does not have implementation of Scheme-related element available, then in accordance with the XSLT specification it must perform fallback when such an extension element is instantiated. A placeholder may be specified as a replacement for such an element by means of the `xsl:fallback` instruction.

7. MARKUP AND CODE SEPARATION

XSLT originates from the intention to separate information content from presentation on the Web. It works fine as long as the content is static, and presentation is the sole purpose of the data presentation, but for more advanced applications the role of XSLT-like transformation of XML data is still twofold: it specifies the transformation and processing of source data and describes its presentation in the resulting document.

A complex transformation of XML data is a programmer's task while its presentation is usually the responsibility of the

Web designers and artists. In a small project a specialist with a universal set of skills may handle the situation. For a larger project, or for a project with higher requirements to either Web design or data transformation and processing the separation between the content and its presentation is highly desirable.

In the context of STX data processing the logic may be sophisticated enough and far beyond the scope of the Web designer responsible for data presentation, thus an aspiration to separate markup and code is of primary importance here.

Such a problem is not XSLT-specific but is a common one for modern XML processing technologies. Introduction of custom tag libraries in the Java Server Pages specification version 1.1 is an example of a technology targeted to resolve this problem in context of Java Server Pages. It allows Java programmers to define "custom" tags which may be used by the Web designer who knows its interface but is not aware of implementation details [6].

STX uses a similar approach: its named templates may be considered as customs tags by a Web designer. A named template may be applied using XSLT-style instructions `xsl:call-template` or using "custom tags".

Some parameters may be passed to a template using `xsl:call-template` and `xsl:with-param` combination in XSLT-style fashion. to call a template,

Templates called using a "custom tag" can use its content or values of its attributes as parameters for a function called. As a particular case `xsl:with-param` may be used, but in general the Web designer has freedom to specify any desired interface.

As it is already described in section 6.3.3, STX provides a mechanism for loading Scheme templates and functions from external files. These libraries, especially problem-oriented sets of Scheme functions, may be developed from the specifications with a little or no knowledge about STX stylesheets they are intended for.

8. REFERENCES

[1] SXPath – SXML Query Language. http://pair.com/lisovsky/SXML/SXPath/.

[2] XML Path Language (XPath) Version 1.0 W3C Recommendation, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[3] XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999. http://www.w3.org/TR/xslt.

[4] XML Information Set W3C Recommendation, October 2001. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[5] XSLT Requirements Version 2.0. W3C Working Draft, February 2001. http://www.w3.org/TR/2001/WD-xslt20req-20010214.

[6] N. J. Bergman. JSP taglibs: Better usability by design. http://www-106.ibm.com/developerworks/library/j-jsptags.html.

[7] D. Jacobs. Rescuing XSLT from Niche Status. http://www.xfront.com/rescuing-xslt.html.

[8] M. Kay. What kind of language is XSLT?, February 2001. http://www-106.ibm.com/developerworks/xml/library/x-xslt/.

[9] O. Kiselyov. SXML expression tree transformers. http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ssax/SSAX/lib/SXML-tree-trans.scm.

[10] O. Kiselyov. SXML Specification. http://okmij.org/ftp/Scheme/SXML.html.

[11] O. Kiselyov. XML and – Scheme. http://okmij.org/ftp/Scheme/SXML-short-paper.html.

[12] K. Lisovsky. Scheme program source code as a semistructured data. In *2nd Workshop on Scheme and Functional Programming Proceedings*, Florence, Italy, September 2001. http://kaolin.unice.fr/Scheme2001/article/lisovsky.ps.