XML, XPath, XSLT implementations as SXML, SXPath, and SXSLT

Oleg Kiselyov[1] and Kirill Lisovsky[2]

[1] FNMOC oleg@okmij.org
[2] MISA University lisovsky@acm.org

Abstract. This paper describes S-expression-based implementations of W3C XML Recommendations, the embedding of XML data and XML query and manipulation tools into Scheme, and our extensive practical experience in using these tools. The S-expression-based data format and query and transformation tools are conformant with W3C Recommendations: XML, XPath, XSLT, and XPointer. The core of our technique is SXML: an abstract syntax tree of an XML document. SXML is also a concrete representation of the XML Infoset in the form of S-expressions. SXML fully supports XML Namespaces, processing instructions, parsed and unparsed entities. An SSAX parser and pretty-printing tools convert between SXML and the angular-bracket-format of XML documents.
All SXML query and manipulation tools are implemented in Scheme. XPath expressions, XSLT patterns and XPointer notation are translated into Scheme data structures and functions. SXML tree is a data structure, but it can be directly evaluated as if it were an expression. These features make XML data and XML processing a part of Scheme and eliminate a lexical barrier for XML processing. A combination of W3C conformance with seamless integration into a programming language is a distinctive feature of our tools.
We have used our approach in real-life commercial and government applications for over two years. We provide *production* examples of a weather data dissemination system, XML transformation systems, and web services.
Keywords: XML, SXML, SXPath, SXSLT, SXPointer, STX, Scheme.

1 Introduction

In this paper, we describe S-expression-based implementations of W3C XML Recommendations: XML, XPath, and XSLT. We have written these tools out of practical necessity, and we have used them in real-life projects for over two years. Our work of developing weather data web services and supply-chain management systems requires advanced manipulation of XML documents. For example, element selection predicates may have to relate points and areas on the globe, to query external data sources, and to evaluate complex business logic. Although some of these algorithms could, in principle, be written in XSLT, the corresponding code will be beyond comprehension. XSLT has never been intended to be a general-purpose programming language [21]. On the other hand, we have to preserve an investment into XSLT presentation stylesheets and XSLT skills. Therefore, we needed tools that "natively understand the data structures inherent in XML and enable optimized algorithms for processing it" [1]. At the same time the tools must be conformant and compatible with the W3C Recommendations. Ideally the tools should let us blend XPath and XSLT on one hand, and the advanced XML processing on the other hand.

The goal of a seamless extension of the W3C XML processing model cannot be easily achieved with Java: Java by nature is an imperative language, whereas XSLT is not. We have reached our goal with Scheme. The choice of Scheme arises from an observation that XML as a datatype follows the nested containment model. Hierarchies of containers that comprise text strings and other containers can be naturally described by S-expressions. S-expressions are easy to parse into an efficient internal representation that is suitable for transformation, manipulation, and evaluation. These properties make S-expressions suitable to represent abstract syntax of XML. On the other hand, S-expressions turned out to be convenient programming notation (much to the surprise of their inventor). S-expressions that represent XML data, SXML, can therefore be evaluated as if they were code. Thus a real, mature programming language for manipulating XML data structures looks exactly like the data structures themselves.

Section 2 describes the XML data model and introduces abstract syntax of XML. Section 3 discusses SXML query language SXPath which is also an implementation of XPath. Section 4 introduces (S)XML transformations. SXPointer is introduced in [12]; space constraints prevent us from providing more detailed discussion. All examples in Sections 2-4 are borrowed from real-life, production use of SXML and SXML-based technologies. Section 5 considers one such example in more detail. The final two sections briefly mention other approaches to XML transformations and conclude.

2 SXML

XML has brought S-expressions back to the fore, albeit in a veiled format and burdened with heavy syntax. Stripping off the syntax – that is, parsing an XML document – yields its abstract syntax tree: a hierarchy of containers comprised of text strings and other containers. Salient features of this hierarchy are described by the XML Infoset [18]. A Document Object Model (DOM) tree, produced by many XML parsers, is one concrete realization of the XML Infoset. The Infoset can also be realized by S-expressions, which are eminently suitable for representation of nested containers. SXML Specification [9] defines such a realization. The fact that SXML represents "parsed" XML makes SXML concise and its applications simple. On the other hand, the fact that SXML is concrete lets us store and communicate SXML expressions, print them out and even write them by hand.

We must stress that SXML represents XML in every detail, including namespaces, external entities, processing instructions. SXML is an extensible format and permits additions of items that, while preserving XML "semantics", can speed up processing: backpointers to the parent or other ancestor elements, hashes of element ids, etc. An XML document represents attributes, processing instructions, namespace specifications and other meta-data differently from the element markup. By contrast, SXML represents element content and meta-data uniformly – as tagged lists. This uniform treatment of attributes and elements is argued by James Clark [2] as a virtue. The uniformity of the SXML representation for elements, attributes, and processing instructions simplifies queries and transformations.

An XML document can be converted into SXML by any parser. In particular, an SSAX parser [8] includes a dedicated function `SSAX:XML->SXML` for this purpose.

Figures 1 and 2 show a sample XML document and its representation in SXML. The XML document is a 24-hour Terminal Aerodrome Forecast (TAF)

for a Paris/Orly airport. Such forecasts are issued by every airport and are distributed by a network of weather services, e.g., Metcast [5]. Metcast is distinguished by being the first production service to disseminate synoptic observations, forecasts and advisories in XML. The particular markup format, OMF, is described in [4]. The document in Fig. 1 is an actual reply of a Metcast server.

```
<!DOCTYPE Forecasts SYSTEM "OMF.dtd">
<Forecasts TStamp="1004046956">
<TAF TStamp='1004029200' LatLon='48.716, 2.383' BId='71490'
     SName='LFPO, PARIS/ORLY'>
<VALID TRange='1004032800, 1004065200'>251700Z 251803</VALID>
<PERIOD TRange='1004032800, 1004065200'>
   <PREVAILING>21007KT 9999 SCT025 BKN035</PREVAILING>
     <VAR Title='BECMG 1820' TRange='1004032860, 1004040000'>
          18005KT CAVOK</VAR>
     <VAR Title='BECMG 0002' TRange='1004054400, 1004061600'>
          6000 SCT035 SCT090 BKN250</VAR>
</PERIOD></TAF></Forecasts>
```

Fig. 1. A sample TAF report in XML

```
(*TOP* (Forecasts (@ (TStamp "1004046956"))
    (TAF (@ (TStamp 1004029200) (LatLon "48.716, 2.383")
            (BId "71490") (SName "LFPO, PARIS/ORLY"))
      (VALID (@ (TRange "1004032800, 1004065200")) "251700Z 251803")
      (PERIOD (@ (TRange "1004032800, 1004065200"))
        (PREVAILING "21007KT 9999 SCT025 BKN035")
        (VAR (@ (Title "BECMG 1820") (TRange "1004032860, 1004040000"))
          "18005KT CAVOK")
        (VAR (@ (Title "BECMG 0002") (TRange "1004054400, 1004061600"))
          "6000 SCT035 SCT090 BKN250")))))
```

Fig. 2. The sample TAF report in SXML

Representation of XML Namespaces in SXML is demonstrated in Figure 3. The XML snippet is actually taken from the XML Namespaces Recommendation. It is very close to shipping documents that we encountered in some of our projects. SXML treats namespaces in a manner intended by the XML Namespaces Recommendation. This subject is discussed in [9] in more detail.

3 Queries

XPath [20] is a basic query language of XML, which is used to access an abstract XML data structure in XSLT, XPointer, and XLink W3C Recommendations. Queries over SXML documents are likewise expressed in SXPath. Just as SXML is closely related to XML, so is SXPath to XPath. XPath addresses an abstract XPath data structure; SXPath queries SXML, which is a concrete representation of the XPath data structure. Both the abbreviated and full-form XPath notations find their counterpart in SXPath. Just as XML can be translated into SXML,

```
<RESERVATION
  xmlns:HTML=
   'http://www.w3.org/TR/REC-html40'>
<NAME HTML:CLASS="largeSansSerif">
    Layman, A</NAME>
<SEAT CLASS='Y'
  HTML:CLASS="largeMonotype">33B</SEAT>
<HTML:A HREF='/cgi-bin/ResStatus'>
    Check Status</HTML:A>
<DEPARTURE>1997-05-24T07:55:00+1
</DEPARTURE></RESERVATION>
```

```
(*TOP* (*NAMESPACES*
          (HTML "http://www.w3.org/TR/REC-html40"))
  (RESERVATION
    (NAME (@ (HTML:CLASS "largeSansSerif"))
      "Layman, A")
    (SEAT (@ (HTML:CLASS "largeMonotype")
             (CLASS "Y"))
       "33B")
    (HTML:A (@ (HREF "/cgi-bin/ResStatus"))
       "Check Status")
    (DEPARTURE "1997-05-24T07:55:00+1")))
```

Fig. 3. A sample XML document with XML namespaces, and its SXML form

XPath expressions can similarly be translated into SXPath. This translation makes SXPath a compliant implementation of the XPath Recommendation.

Similarly to XPath, SXPath has several levels: lower-level predicates, filters, selectors and combinators; and higher-level, abbreviated SXPath notation. The latter has two formats: a *list* of location path steps ("native" SXPath), or a character string of a XPath expression ("textual" SXPath). The format of the latter string is defined in the W3C XPath Recommendation [20]. Native SXPath expressions can be regarded a "parsed" form of XPath. Low-level SXPath functions constitute a virtual machine into which both kinds of higher-level expressions are compiled.

For example, given a TAF SXML expression in Fig. 2, we can determine the validity time ranges of the forecasts by evaluating:

```
((sxpath '(// TAF VALID @ TRange *text*)) document)
```

using the native higher-level SXPath notation, or

```
((txpath "//TAF/VALID/@TRange/text()") document)
```

with the textual SXPath. The first argument of `txpath` is a fully XPath-compliant location path. Both functions `sxpath` and `txpath` translate into a low-level SXPath expression:

```
((node-join
    (node-closure (node-typeof? 'TAF))
    (select-kids (node-typeof? 'VALID))
    (select-kids (node-typeof? '@))
    (select-kids (node-typeof? 'TRange))
    (select-kids (node-typeof? '*text*)))
 document)
```

Such SXPath expressions are used in a Scheme servlet that translates OMF TAF documents sent by a Metcast server into a simple web page for presentation on a PalmPilot-like handheld equipped with a wireless receiver. The servlet is part of an aviation weather query tool [7] for operational users and small plane pilots.

SXPath expressions can be more complex: for example, given a booking record in Fig. 3 we can select the names of the passengers with confirmed reservations as follows:

```
((sxpath
  `(// (RESERVATION (
    ,(lambda (res-node)
       (run-check-status
            ; URL of the confirmation script
        (car ((sxpath '(HTML:A @ HREF *text*)) res-node))
        ((select-kids (node-typeof? '(NAME SEAT DEPARTURE)))
         res-node)))
    )) NAME *text*)) document)
```

We have seen that full-form or abbreviated SXPath expressions are S-expressions. The full-form SXPath is actually a library of primitive filters, selectors and combinators that can be combined in arbitrary ways – as well as combined with standard Scheme and user-defined functions. Abbreviated SXPath expressions are S-expressions that may include procedures and lambda-expressions, which thus extend the set of selectors and predicates defined in XPath. For example, the reservation query above relies on a custom predicate to retrieve a dynamic web page and check the status. The predicate invokes high and low-level SXPath functions to access particular fields of the reservation record. The ability to use Scheme functions as SXPath predicates, and to use SXPath selectors in Scheme functions makes SXPath a truly extensible language. A user can compose SXML queries following the XPath Recommendation – and at the same time rely on the full power of Scheme for custom selectors.

4 S-expressions and transformations

The W3C XML transformation language XSLT [21] defines XML transformations as converting a source (abstract XML syntax) tree into a result tree. The conversion can be thought of as a recursive traversal/mapping process: we visit tree nodes, locate the corresponding node mapping handler in the transformation environment and execute the handler. If we traverse the tree in pre-order the handler is applied to the branch of the source tree. We can also employ a post-order mode: once we enter a branch, we first traverse its children and apply the handler to the transformed children. The handler can be located by the name of an element node or by more complex criteria. If an XML tree is realized as an SXML expression, the described process is literally implemented by a function pre-post-order whose complete code is part of the SSAX project [15]. The function takes a source SXML tree and the transformation environment and yields the result tree. The environment is a list of associations of node names with the corresponding transformers. Special bindings `*text*` and `*default*` may be used in order to define catch-all transformation rules for elements and atomic data. The transformation environment passed to pre-post-order is essentially a transformation *stylesheet*. It is also an S-expression. Thus the document to transform and its transformation are expressed uniformly in the same language.

We should note that a transformation of an SXML tree in post-order is equivalent to its evaluation as if it were a piece of Scheme code. Pre-order transformations are similar to macro-expansions of the tree – again, as if it were a piece of Scheme code. The pre-order traversal is built into XSLT.

One example of SXML transformations is pretty-printing of SXML into HTML, XML or LaTeX. Incidentally, this facility lets us author a text in SXML and then convert it into a web page or a printed document. When writing a text in SXML we can use higher-order tags (similar to LaTeXmacros) to make

the markup more logical. Compared to LaTeXmacros, SXML markup is more expressive: for example, the process of expansion of one SXML tag can re-scan the entire SXML document with a different stylesheet. Such a reflexive behavior is a necessity while generating hierarchical tables of contents or resolving cross- and citation references. The SXML Specification is an example of such a higher-order markup. The specification is written in SXML itself and relies on higher-order tags to describe SXML grammar in a concise and abstract way. The present paper is also written in SXML and then converted into PDF through LaTeX.

Another example of SXSLT is an XML transformation system STX [11], which, on the surface, is a processor for a frequently-used subset of XSLT and therefore compatible with the W3C XSLT Recommendation [21]. Behind the scene, STX translates both the source document and the stylesheet into SXML, and then literally applies one to the other. Therefore, STX permits embedding of Scheme functions into XSLT templates (`scm:eval`) to express complex business logic. More importantly, STX allows an advanced user to write template rules as regular first-class Scheme functions (`scm:template`) similarly to the SXML transformers in the pre-post-order stylesheets described above.

STX was originally designed as one of the development tools for ECIS (Extranet Customer Information System) project 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.

5 Detailed Example: Concise XML Authoring

In this section, we elaborate a non-contrived and less-trivial example of building XML documents from S-expressions. It is straightforward to convert `<tag>data</tag>` into `(tag "data")` and vice versa. The SSAX parser and the SXML manipulation tools can do that easily. However, exporting relational sources into XML often runs into an impedance mismatch: XML by nature is a hierarchical database. We will provide an example of generating XML from S-expressions that involves denormalizations, table joins and third-order tags. The S-expression format turns out to be not only more understandable and insightful, but also four times shorter.

The example is based on a real-life project of preparing a submission of a synoptic markup format OMF [4] for U.S. Department of Defense's XML registry. The registry [3] accepts submissions of XML formats as collections of DTD/Schema documents, textual descriptions, sample code, etc. Every submission package must include a Manifest.xml file, which describes all containing documents as well as every markup element and its attributes.

Figure 4 shows a representative example of the manifest file. The snippets are taken from the actual Manifest.xml [22] and edited for brevity. They are still hardly readable. The OMF submission contained 109 resources; therefore, preparing such a manifest manually was out of the question. We will see shortly how that ugly document can be represented in a concise and pleasing form.

The manifest file is essentially a collection of resource descriptions and of statements of resource relationships. It seemed logical then to make such a structure explicit. Figure 5 shows SXML code that corresponds to the superset of the XML snippets. The SXML code is translated into Manifest.xml by a function pre-post-order described in Section 4. The SXML code and a transformation stylesheet for pre-post-order make up the file Manifest.scm [23].

Description of a sample XML document:

```
<AddTransaction><EffectiveDate>27 February 2001</EffectiveDate>
<Definition>OMF Example: METAR/SYNOP/SPECI</Definition>
<Namespace>MET</Namespace>
<InformationResourceName>OMF-sample.xml</InformationResourceName>
<InformationResourceVersion>OMF2.2</InformationResourceVersion>
<InformationResourceTypeXMLSample>
<InformationResourceLocation>OMF-sample.xml</InformationResourceLocation>
<Relationships><DescribedBy><Namespace>MET</Namespace>
<InformationResourceName>OMF-SYNOP.html</InformationResourceName>
<InformationResourceVersion>OMF2.2</InformationResourceVersion>
</DescribedBy></Relationships>
</InformationResourceTypeXMLSample></AddTransaction>
```

Description of one XML element (element BTSC) within the submitted collection:

```
<AddTransaction><EffectiveDate>12 April 2000</EffectiveDate>
<Definition>an observation report on temperature, salinity and
currents at one particular location on the ocean surface, or in
subsurface layers</Definition><Namespace>MET</Namespace>
<InformationResourceName>BTSC</InformationResourceName>
<InformationResourceVersion>OMF4.1</InformationResourceVersion>
<InformationResourceTypeXMLElement><DataTypeContainer><Contains><Namespace>MET</Namespace>
<Contains><Namespace>MET</Namespace>
<InformationResourceName>BTLEVELS</InformationResourceName>
<InformationResourceVersion>OMF4.1</InformationResourceVersion>
</Contains></DataTypeContainer>
<Relationships><IsQualifiedByAttribute><Namespace>MET</Namespace>
<InformationResourceName>TStamp</InformationResourceName>
<InformationResourceVersion>OMF4.1</InformationResourceVersion>
</IsQualifiedByAttribute>
<IsQualifiedByAttribute><Namespace>MET</Namespace>
<InformationResourceName>Depth</InformationResourceName>
<InformationResourceVersion>OMF4.1</InformationResourceVersion>
</IsQualifiedByAttribute>
<DescribedBy><Namespace>MET</Namespace>
<InformationResourceName>OMF-BATHY.html</InformationResourceName>
<InformationResourceVersion>OMF1.4</InformationResourceVersion>
</DescribedBy></Relationships>
</InformationResourceTypeXMLElement></AddTransaction>
```

Description of an attribute, TStamp, which annotates a BTSC element:

```
<AddTransaction><EffectiveDate>12 April 2000</EffectiveDate>
<Definition>Time Stamp</Definition><Namespace>MET</Namespace>
<InformationResourceName>TStamp</InformationResourceName>
<InformationResourceVersion>OMF4.1</InformationResourceVersion>
<InformationResourceTypeXMLAttribute><DataTypeInteger><IntegerLength>10</IntegerLength>
<IntegerUnitMeasure>second</IntegerUnitMeasure>
</DataTypeInteger>
<Relationships><DescribedBy><Namespace>MET</Namespace>
<InformationResourceName>OMF.html</InformationResourceName>
<InformationResourceVersion>OMF2.2</InformationResourceVersion>
</DescribedBy></Relationships>
</InformationResourceTypeXMLAttribute></AddTransaction>
```

Fig. 4. Manifest of an XML registry submission

```
   (Resource "OMF.html"
             "Weather Observation Definition Format (OMF) Document"
             "10 March 2000" "2.2")
   (DescribeDoc "OMF.html")

   (Resource "OMF-SYNOP.html"
             "Surface Weather Reports from land and sea stations"
             "12 April 2000" "2.2")
   (DescribeDoc "OMF-SYNOP.html")

   (Resource "BTSC" "an observation report on temperature, salinity
and currents at one particular location on the ocean surface, or
in subsurface layers" "12 April 2000" "4.1")
   (Resource "BTID" "identification and position data, which
constitute Section 1 of FM 62 - 64." "12 April 2000" "4.1")
   (Resource "BTLEVELS" "a sequence of BTLEVEL elements for each
particular (sub)surface level described in a whole BTSC report"
"12 April 2000" "4.1")

   (XMLElement "BTSC" (DTContainer "BTID" "BTCODE" "BTLEVELS")
               "OMF-BATHY.html"
               (Attlist "TStamp" "LatLon" "BId" "SName" "Title" "Depth"))

   (XMLElement "BTID" (DTString 40) "OMF-BATHY.html"
               (Attlist "DZ" "Rec" "WS" "Curr-s" "Curr-d" "AV-T" "AV-Sal"
                        "AV-Curr" "Sal"))

   (Resource "TD" "The dew-point temperature" "12 April 2000" "4.1")
   (Resource "TRange" "Time Interval" "12 April 2000" "4.1")
   (Resource "TStamp" "Time Stamp" "12 April 2000" "4.1")

   (XMLAttr "TD" (DTFloat 6 2 "deg C") #f "OMF-SYNOP.html")
   (XMLAttr "TRange" (DTString 30) #f "OMF.html")
   (XMLAttr "TStamp" (DTInt 10 "second") #f "OMF.html")
```

Fig. 5. The Manifest file in SXML

Let us consider two S-expressions from Fig. 5 in more detail:

```
(Resource "OMF-SYNOP.html"
          "Surface Weather Reports from land and sea stations"
          "12 April 2000" "2.2")
(DescribeDoc "OMF-SYNOP.html")
```

If we take a naive view of SXML-to-XML transformations, we might think that the corresponding Manifest.xml document [22] will include tags `<Resource>` and `<DescribeDoc>`. In fact, the XML manifest document contains neither. The S-expression `Resource` merely declares the resource and serves as a container of its attributes: its name, documentation string, modification date and version. During the SXML transformation, the S-expression translates to nothing: the following is the handler for the `Resource` tag in the SXML transformation stylesheet.

```
(Resource . ,(lambda (tag name title date version)
    '()))               ; null expansion
```

Resources are described differently depending on their type. For example, `(DescribeDoc "OMF-SYNOP.html")` tells that `OMF-SYNOP.html` is a textual document. This S-expression will be transformed according to the following stylesheet rule:

```
(DescribeDoc    ; Describe a document resource
 . ,(lambda (tag name)
      (generate-XML
       `(AddTransaction
         (Resource-descr ,name)
         (InformationResourceTypeDocument
          (InformationResourceLocation ,name)
          )))))
```

The rule expands `DescribeDoc` into a set of SXML elements required by the registry (e.g., `InformationResourceTypeDocument`), which are distinguished by their unwieldy names. The `DescribeDoc` handler invokes the transformation function recursively, to effect the second pass. The pass will transform `(Resource-descr "OMF-SYNOP.html")` according to the rule:

```
; Locate a named resource and expand into its full description.
(Resource-descr
 . ,(lambda (tag name)
      (let-values* (((name title date version)
                     (lookup-res name)))
        (generate-XML
         (list
          (list 'EffectiveDate date)
          (list 'Definition title)
          '(Namespace "MET")
          (list 'InformationResourceName name)
          (list 'InformationResourceVersion "OMF" version))))))
```

The other tags and text strings will be handled by the default rules of the stylesheet:

```
(*default* . ,(lambda (tag . elems) (apply (entag tag) elems)))
(*text* . ,(lambda (trigger str)
             (if (string? str) (string->goodXML str) str)))
```

The end result will be the difficult-to-read XML fragment in Fig. 4.

The processing of `(Resource-descr "OMF-SYNOP.html")` invokes a function `lookup-res`, which queries the SXML Manifest for `(Resource "OMF-SYNOP.html" ...)`. The fields of the found S-expression are used to generate the proper resource description. In database terms, the expansion of `(DescribeDoc "OMF-SYNOP.html")` is a join of two tables. The set of `(Resource ...)` S-expressions in Manifest.scm represents one table. S-expressions `(DescribeDoc ...)` make up the other table. The resource's name (which is unique in a submission package) is the primary key in both tables. The expansion of `DescribeDoc` involved two re-writing steps, therefore, `DescribeDoc` is a third-order tag.

It seems that Manifest.scm describes the collection of OMF resources in a more readable and understandable manner. It is instructive to compare the file sizes of Manifest.scm and Manifest.xml: Manifest.scm is 25831 bytes long, of which 9220 bytes are the transformation stylesheet and the related code. The size of the file Manifest.xml is 90377 bytes.

The Manifest files in S-expression and XML formats are part of the OMF submission into the DoD XML Registry. They can be retrieved from the registry [22], [23].

6 Related work

The fact that XML is S-expressions with significantly more syntax was recognized fast [16]. This realization was followed by actions to recover the original beauty of S-expressions: [10], [6], [13]. Even some OpenSource projects chose to use S-expressions in their pristine format [14].

There are many ways to author, convert, or query XML documents, relying either on template expansions, or embeddings of XML/HTML into or around a host language. Examples include XSLT/XPath, ColdFusion, Microsoft's Active Server Pages (ASP), Java Server Pages (JSP), Hypertext Preprocessor (PHP). They all however introduce a particular mini-language with a peculiar, non-XML-like syntax and ad-hoc substitution semantics. The linguistic gap is wide. Representing an XML/HTML document as code that upon execution will generate the document is also a popular approach. CGI.pm in Perl, LAML in Scheme [13], htmllib in Tcl, or Element Construction Set in Java are only a few examples. Often such code even looks remotely like S-expressions, albeit greatly more verbose and cumbersome, and difficult to process reflexively.

7 Conclusions

Immense popularity of XML is the testimony to the flexibility of S-expressions. A great variety of data structures can be represented by S-expressions; many operations can be regarded as transformations of S-expressions. There is no reason, however, to hide S-expressions behind layers of syntax. The most important benefits of this approach are the use of a mature programming language for creation and processing of semi-structured data and the disappearance of the linguistic gap between XML data and processes to create and handle them.

It does not seem likely, however, that the world would abandon XML for S-expressions overnight. Therefore, we have to embrace the existing formats and tools and be able to extend them. Our tools meet this challenge. Internally we transform and evaluate S-expressions and yet we accept and generate valid XML with Namespaces, entities and other complexities. We can use many of the frequently-used XSLT stylesheets as they are [11], and can extend them through embedded Scheme code and SXML-transformer-style templates. We can write W3C-compliant XPath expressions along with their S-expression-formatted counterparts, which we can extend with powerful predicates in Scheme. We have used in practice such a transition path from pure XML to more S-expression-based formats and tools. Due to a unique combination of the expressive power of the Scheme language and compatibility with W3C Recommendations, such an approach provides the most sophisticated XML processing techniques along with a good protection of investments in XML/XSLT solutions.

The SXML parsing, query and transformation tools are released into public domain as part of the SSAX project [15].

References

1. Adam Bosworth: A Programming Paradox. XML Magazine, February 2002. http://www.fawcette.com/xmlmag/2002_02/magazine/departments/endtag/
2. James Clark, The Design of RELAX NG. December 6, 2001. http://www.thaiopensource.com/relaxng/design.html
3. DoD XML Registry http://diides.ncr.disa.mil/xmlreg/user/index.cfm
4. Oleg Kiselyov: Weather Observation Definition Format. March 8, 2000. http://zowie.metnet.navy.mil/~spawar/JMV-TNG/XML/OMF.html
5. Oleg Kiselyov: Implementing Metcast in Scheme. Proceedings of the Workshop on Scheme and Functional Programming 2000. Montreal, 17 September 2000.
6. Oleg Kiselyov. XML and Scheme. An introduction to SXML and SXPath; illustration of SXPath expressiveness and comparison with XPath. September 17, 2000. http://pobox.com/~oleg/ftp/Scheme/SXML-short-paper.html
7. Oleg Kiselyov: Aviation total weather and SIGMET advisory queries. June 12, 2000.
   http://zowie.metnet.navy.mil/cgi-bin/oleg/get-sigmet-light
   http://zowie.metnet.navy.mil/~dhuff/pqa/FNMOC.html
8. Oleg Kiselyov. A better XML parser through functional programming. LNCS 2257, pp. 209-224. Springer-Verlag, January 2002.
9. Oleg Kiselyov. SXML Specification. Revision 2.1. March 1, 2002. http://pobox.com/~oleg/ftp/Scheme/SXML.html
10. Shriram Krishnamurthi, Gray, K.E. and Graunke, P.T.: Transformation-by-Example for XML. Practical Aspects of Declarative Languages, 2000.
11. Kirill Lisovsky. STX: Scheme-enabled Transformation of XML data. http://pair.com/lisovsky/STX/
12. Kirill Lisovsky. SXPath and SXPointer. http://pair.com/lisovsky/sxml/sxpath/
13. Kurt Normark. Programming World Wide Web Pages in Scheme. ACM SIGPLAN Notices, vol. 34, No. 12 - December 1999, pp. 37-46. http://www.cs.auc.dk/~normark/laml/
14. Black Parrot: Re:Scheme as an XML Translation Language. October 12, 2001. A comment in the thread "Ask Kent M. Pitman About Lisp, Scheme And More" http://slashdot.org/comments.pl?sid=22519&cid=2422286
15. S-exp-based XML parsing/query/conversion. http://ssax.sourceforge.net/
16. Philip Wadler: The Next 700 Markup Languages. Invited Talk, Second Conference on Domain Specific Languages (DSL'99), Austin, Texas, October 1999.

17. World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. October 6, 2000. http://www.w3.org/TR/REC-xml
18. World Wide Web Consortium. XML Information Set. W3C Recommendation. 24 October 2001. http://www.w3.org/TR/xml-infoset
19. World Wide Web Consortium. Namespaces in XML. W3C Recommendation. January 14, 1999. http://www.w3.org/TR/REC-xml-names/
20. World Wide Web Consortium. XML Path Language (XPath). Version 1.0. W3C Recommendation. November 16, 1999. http://www.w3.org/TR/xpath
21. World Wide Web Consortium. XSL Transformations (XSLT). Version 1.0. W3C Recommendation November 16, 1999. http://www.w3.org/TR/xslt
22. http://zowie.metnet.navy.mil/~spawar/JMV-TNG/XML/Manifest.xml http://diides.ncr.disa.mil/xmlreg/package_docs/Public/MET/OMF_package/997725059053/Manifest.xml
23. http://zowie.metnet.navy.mil/~spawar/JMV-TNG/XML/Manifest.scm http://diides.ncr.disa.mil/xmlreg/package_docs/Public/MET/OMF_package/997725059053/Manifest.scm