XML View on Hierarchical Data Using SXML and Scheme

Oleg Paraschenko

Saint-Petersburg State University
7-9, Universitetsjaya nab,
Saint-Petersburg, Russia
olpa@uucode.com
http://uucode.com/

Abstract. Hierarchical data could be viewed and processed as XML using the SXML format and Scheme language. We introduce a symmetry constraint on this approach, reveal the weak points of the SXML representation, and discuss mapping between XML and SXML.

1 Introduction

Applications like compilers and text processors, which intensively work with tree-like data, could benefit of integrating XML technologies. The data could be viewed as XML and processed using XPath, XSLT or XQuery. To make it real, re-usable XML processing libraries are required.

Code of such libraries should be adaptable to different programming languages and tree structures. Our approach [1] is to embed an XML virtual machine to a host application. The virtual machine has the programming language Scheme [2] as its native byte code. XML processing code is written in Scheme and uses the SXML format [3] as a view of application's data.

A glue code between host application and Scheme layers establishes a transparent mapping of data between the layers. It means that the same data is represented using native data structures in the host application and as SXML in Scheme. This paper summarizes issues of this approach.

Our experience is based on three projects: Python AST as XML [4], GNU find with XPath [5], and XSieve [6]. The first two are research prototypes, and the latter is a language and a practical tool for XML data transformation. We hope our paper can be used as a checklist and food for thought for those who want to implement a similar system.

Our XSieve implementation is built on top of XSLT processor xsltproc [7] and Scheme interpreter Guile [8]. The host application is xsltproc (more precisely, libxml2 and libxslt libraries), its tree-like structure is XML itself. The paper is written with XSieve in mind, but the paper's points are applicable to any tree-like structures, not only to XML.

We start the paper with an example of equivalent XML and SXML representations. Then we add a constraint on our approach: the property of symmetry.

Fortunately, most practical use cases for our approach satisfy the requirement. Then we describe problems of converting namespace and attribute nodes.

The next section is a summary of issues with representing XML data as SXML. Parent pointers and the equality property add some complexity, and lazy instantiation is an essential problem. These issues could be avoided by using some other XML representation, but we want to re-use big code base of existing SXML tools [9], and therefore we have to use SXML.

The rest of the paper is about mapping between equivalent XML and SXML nodes. We list the needs for the map and raise the question of memory management. Finally, we discuss issues and possible optimizations when mapping data for use in XPath and XSLT.

2 XML and SXML

2.1 Symmetry

The following example demonstrate how the same data is represented in XML and SXML formats. XML:

```
<article id="hw">
  <title>Hello</title>
  <para>Hello, <object>World</object>!</para>
</article>
```

SXML:

```
(article (@ (id "hw"))
  (title "Hello")
  (para "Hello, " (object "World") "!"))
```

Depending on the mode (an application or Scheme), the data is automatically represented either as XML or as SXML. Unfortunately, this symmetry might be broken if the both representations are instantiated, and one of the copies is modified.

In this work, we limit ourselves to simple, but important case when trees are read-only, and the property of symmetry is satisfied automatically. XPath, XQuery and XSLT implementations fit to this case.

2.2 Issues of Converting

According to our experience, the hardest part of a converter is namespace processing. Some of the troubles are handing scope of prefixes, supporting the default namespace, prefix rewriting, and so on. The main problem is that it's easy to create an SXML tree with free namespaces, when some namespace prefixes are not defined. There is no correct workaround. In XSieve, we leave free namespaces as is, hoping that upon adding the tree to a bigger tree, the binding will occur and become correct.

Returning attribute and namespace nodes is another sort of issues. In SXML, it is possible to create these nodes without creating an owning element. Such independence might be impossible in the host application. Libxml is an example of a library with this issues. In XSieve, we create a fake owner element before converting an independent attribute or namespace node.

3 SXML Issues

3.1 Lazy Instantiation

Scheme code might need to process only top-level nodes of an XML subtree, ignoring deep child nodes. In this case converting the whole XML subtree is an overhead. Alternative is to instantiate Scheme values on demand.

SXML format isn't compatible with lazy instantiation. To navigate over XML tree, the core Scheme functions, such as car, cdr or map, are used. In most Scheme implementations, these functions doesn't support delayed values.

For Guile, we created a patch which allows lazy lists (S-expressions). We redefined macro `SCM_CAR` and `SCM_CDR` to check if the head or tail of a pair are delayed values, and automatically evaluate them. Special care is taken to make sure that implicit instantiation doesn't happen during garbage collection.

Other Scheme implementations might natively support lazy lists. At least, this possibility is mentioned in the Scheme R5RS standard [2].

3.2 Parent pointers

Parent pointers is an essential mismatch between XML and SXML. In XML, each node (except a document root) has one and only one parent, but SXML nodes doesn't have links to parents.

To implement XPath parent and ancestor axis, we should be able to find the parent of an SXML node. Several approaches are proposed by Oleg Kiselyov [10], but each proposal has drawbacks.

As we have control over mapping of nodes between layers, we can use a better way. While converting XML to SXML, we can create a map between XML and SXML nodes. When the parent of an SXML node is required, a Scheme function can find the corresponding XML node in the map, then its parent, and then return the corresponding SXML node.

This method of getting the parent node works only if an SXML node was not constructed by Scheme code, but was converted from XML. It's not a problem because we want to apply XML technologies to application data, not to Scheme data.

Another issue appears when returning a subtree from the Scheme layer to the application layer. The application might need to insert the subtree to a tree. As the root subtree node can't have two parents, a copy of the subtree should be created and added to the tree.

3.3 Equality of Nodes

The same XML nodes in a host application should be the same SXML nodes in Scheme. This requirement is important, for example, for implementing XPath. Each XPath step should return only unique nodes, and it's convenient to use the Scheme operator "eq?" to filter out duplicates in a node set.

Using "eq?" works well for element, root, comment and processing instruction nodes, but attribute and namespace nodes require more sophisticated equality. When XPath returns an attribute node, the SXML node looks something like this:

```
(@ (attr-name "attr-value"))
```

On the other side, consider an element node with attributes:

```
(elem-name (@ (attr-name "attr-value") (attr-name-2 "attr-value-2")))
```

After getting the attribute "attr-name" using the functions car and cdr, the SXML node looks like the following:

```
(attr-name "attr-value")
```

Obviously, "`(@ (attr-name "attr-value"))`" and "`(attr-name "attr-value")`" are not equal. To have the property of equality of attribute nodes, we demand that the common part of the both expressions "`(attr-name "attr-value")`" is the same Scheme value.

Namespace nodes have the same problem, which is handled by analogue.

As result, a node comparator isn't just a call to "eq?". It should correctly handle different forms of attribute and namespace nodes.

A natural way for supporting the equality property is to remember results of converting XML nodes to SXML nodes. If an XML node is already converted, Scheme code gets already existing SXML node.

4 Mapping Between XML and SXML

A need for mapping between XML and SXML nodes is already mentioned during discussion of parent pointers and equality of nodes. Yet another argument is a need to switch from a Scheme result to XML nodes after running code in the Scheme layer.

The mapping also optimizes conversion between XML and SXML nodes. Once a tree is converted, the result is remembered. Subsequent requests for conversion of these nodes return the stored result immediately instead of re-evaluating.

Constructing a two-side map is a simple task. It is enough to add a mapping pair after converting each XML or SXML node.

Memory management is the main problem with the mapping. Application-dependent methods should be used to make sure that the references in the map can't become invalid. In case of XSieve, libxml2 memory management and Guile garbage collection were considered. In libxml2, it is possible to register a callback on deleting a node, and delete also the corresponding mapping pair. In Guile, to avoid deleting values, each root of converted trees is marked as protected from garbage collection.

5 Issues of mapping for XPath and XSLT

XPath data model and XSLT processing model have the properties which add constraints and allow optimizations.

In XPath, a text node never has an immediately following or preceding sibling that is a text node. The glue code should prevent immediate text siblings or at least don't fail on them. For example, if one adds a new text node to a tree, libxml checks if the last node is text, moves the text content from the new node to the last text node, and frees the new node. The node reference in the corresponding mapping pair become invalid, so the pair should be deleted too.

In XSLT, input and output XML trees are independent. If XML is put to the output, it can't appear as the input. It means that after converting data to XML, an application doesn't need SXML representation. Therefore, there is no need to update the map during SXML to XML conversion.

The most part of XSLT processing operates on a context node. If XSLT traversing has passed by a node, then, most likely, the node will never be used again, and several entries in the map are useless. To avoid excessive growth of the map, it can be implemented using a weak map. Garbage collector automatically deletes pairs from a weak map if the pairs reference unneeded values.

In the worst case, one XML node can be converted to SXML several times, each time represented by a different Scheme value. This doesn't break the property of equality. Indeed, the life time of these values is different, and no two values exist simultaneously, so it doesn't make sense to say that they are equal or not equal.

6 Related Work

The main features of our approach are:

- XML view on non-XML data,
- a virtual machine for XML processing,
- using Scheme for XML processing,
- embedding Scheme,
- symmetry of data.

Each individual aspect isn't new, but the combination of them is unique.

The first feature, XML view on arbitrary data, can be found in the Java system JXPath [11] from the Apache Software Foundation and in .NET API

XPathNavigator [12] from Microsoft. These solutions are limited only to Java and .NET, respectively.

Authors of XML libraries sometimes notice that the core of a library is a sort of a virtual machine, but they don't work on the idea. The only real virtual machine we aware of is XSLT Virtual Machine (XVM) [13] from Oracle. Unfortunately, there is no information if XVM can be embedded and used for processing arbitrary tree-like data structures.

The most part of XML processing in Scheme is performed using SXML tools [9]. The home page of the tools has a collection of links to Scheme XML projects. We are not aware about any notes on the equality property of SXML nodes. Lazy XML processing in the tools means lazy code evaluation, not lazy data instantiation.

Embedding Scheme is a well-known topic. Many implementations were designed for embedding, or at least support it. Specific details come with documentation. For Guile, there is also a useful guide by William Morgan [14].

Mapping data representations is a common task. It is done, for example, each time when extending a high-level language by C libraries. In most cases, bindings provide an API to navigate over data, and don't attempt to avoid API in favor of compound native data structures with the property of symmetry.

References

1. Paraschenko, O.: Reusing XML Processing Code in non-XML Applications. http://uucode.com/texts/genxml/genxml.html
2. Kelsey, R., Clinger, W., Rees, J. (eds.): Revised5 Report on the Algorithmic Language Scheme. Higher-Order and Symbolic Computation, Vol. 11, No. 1, August, 1998. http://www.brics.dk/ hosc/11-1/
3. Kiselyov, O.: SXML Specification. http://okmij.org/ftp/Scheme/xml.html#SXML-spec
4. Paraschenko, O.: Python AST as XML http://pysch.sourceforge.net/ast.html
5. Paraschenko, O.: Find with XPath over file system. http://uucode.com/texts/xfind/index.html
6. Paraschenko, O.: XSieve book. http://xsieve.sourceforge.net/
7. Veillard, D.: Libxslt. http://xmlsoft.org/XSLT/
8. Free Software Foundation, Inc.: Guile (About Guile). http://www.gnu.org/software/guile/guile.html
9. Kiselyov, O.: S-exp-based XML parsing/query/conversion. http://ssax.sourceforge.net/
10. Kiselyov, O.: On parent pointers in SXML trees. http://okmij.org/ftp/Scheme/xml.html#parent-ptr
11. The Apache Software Foundation: JXPath — JXPath Home. http://jakarta.apache.org/commons/jxpath/index.html
12. Microsoft Corporation: XPathNavigator over Different Stores. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconXPathNavigatorOverDifferentStores.asp
13. Novoselsky, A.: The Oracle XSLT Virtual Machine. In Proc. XTech, 2005. http://idealliance.org/proceedings/xtech05/papers/04-02-01/

14. Morgan, W.: Incorporating Guile into your C program. http://www.masanjin.net/ wmorgan/IncorporatingGuileIntoYourCProgram.txt