# Reusing XML Processing Code in non-XML Applications

## Version 1.0, 16 April 2005

Oleg Paraschenko

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

**Abstract.** XML can be considered as a representation of hierarchical data, and the XML-related standards — as methods of processing such data. We describe benefits of XML view on legacy data and its processing, and suggest a method to develop XML tools and make them reusable for different tree-like structures in different programming languages.
Our approach is to use virtual machine technology, in particular, the Scheme programming language. We're taking the unusual step of using the Scheme syntax itself as a native virtual machine language. Together with the SXML format and Scheme implementations tuning, it gives us the XML virtual machine (XML VM).
Reference implementations are ready for the Python and C languages. We describe a library for XSLT-like transformations of the Python parse trees and a special version of the GNU find utility which supports XPath queries over the file system.

## 1 Introduction

The basis of some applications is hierarchical data structures plus methods of processing the data. Possible examples are compilers, interpreters and text processors. The former two may use abstract syntax trees (ASTs) to represent results of parsing a program and some other trees to deal with code transformations. Text processors may use trees to represent the structure or the formatting properties of a document.

Although the applications are from different domains, the basic operations on trees are the same. For example, getting list of children is a standard functionality for any tree. Some more advanced operations (for example, getting children with filtering) are also common. The problem is that possibly complex code for the advanced operations should be written and being updated for all the tree models, so maintenance become nightmare.

The possible solution for elimination of code duplication is to unify the tree models. As XML [1] is now extremely popular, it's reasonable to choose XML as an universal representation for trees. For needs of our work, the most fruitful

feature of XML is existence of XML-related standards which in fact are design patterns for tree processing. And one of the most useful standards is XPath [2]. In this paper, we give examples of XML and XPath and discuss benefits of seamless use of XML tools in legacy applications.

XML is not the only candidate for universal data representation. Lisp gave the world S-expressions [8], which are much older than XML, and techniques of processing S-expressions. Comparing and contrasting Lisp and XML is a flame topic [9]. Our work combines both worlds together, and we describe SXML [11] format, SXML tools, and their relationship to XML.

Having XML and Scheme described, we introduce an XML virtual machine (XML VM) which is based on the SXML library and uses the Scheme programming language itself as a virtual machine. This is a quite novel approach, and we advocate for it describing:

- why use Scheme,
- drawbacks of alternative approaches, and
- seamless integration of the VM and a host language.

The SXML tools and Scheme implementations were not intended for a such use, so their integration into the united VM was a challenging task with obstacles, some of which are mentioned.

Finally, we describe working applications which prove the idea of using XML VM in non-XML applications. These examples are not only proofs of the concept, but also are reference implementations for the Python and C programming languages.

## 2 XML

Extensible Markup Language (XML, [1]) is a text format which helps to represent a logical structure in documents. Here is the example of a possible document in the XML format:

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

Ignoring details, one can said that XML documents consist of elements and attributes. Elements specify named text extents and attributes curry properties. Elements should be properly nested, so any XML can be mapped to a tree.

One of the keys in the success of XML is interoperability. When different legacy systems talk to each other, they usually speak in XML. Any data can be mapped to XML, but the most natural for XML is to be mapped to hierarchical data.

Important side of XML is plenty of standards. They cover most of issues related to data presentation, validation, transformation and exchange. In fact, they introduce design patters for tree processing.

There are tools designed to work with XML, and sometimes instead of processing data inside an application itself, it's easier to export data to XML, process them using XML tools and import the result back.

Some of such tools are those which implement XPath [2] and XSLT [3] standards. XPath is a language for navigation over XML trees, and XSLT is a template language for tree transformation.

## 3 XPath

XPath [2] is a language for navigating, matching and querying on XML data.

An XPath expression is a series of location steps divided by the slash symbol (`/`). A location step has three parts:

- an axis, which specifies the tree relationship,
- a node test, which specifies the node type, and
- zero or more predicates, which use arbitrary expressions to filter nodes.

Here is the example of an XPath expression:

```
/child::article[attribute::id='hw']/child::para
```

This XPath consists of two location steps. The first step uses `child` axis to select all children of the root node, the node test filters those which are named `article`, and the predicate filters those which have an attribute `id` set to the value `hw`. The second step selects all the paragraphs (elements `para`) of the article selected.

There is a number of syntactic abbreviations that allow common cases to be expressed concisely. For example, the XPath above can be also written as:

```
/article[@id='hw']/para
```

## 4 Benefits

Instead of writing custom code for hierarchical data processing, we suggest to provide a seamless mapping between legacy data and XML, and use XML tools to process the legacy data through XML. This approach has a number of advantages which are discussed below. At the moment, we partially limit ourselves to XPath, but the following items are valid for other standards too.

One of the main benefits of domain-specific languages (DSLs), and XPath in particular, is separation of concerns. Tree processing usually is not a domain area of an application, but an auxiliary functionality. It's better to concentrate on a high-level logic of the applications, and leave supporting technical details to a DSL language.

One of the nice things about XPath (and DSLs in general) is that it is concise. We have experience of working with a good enough legacy library, which had a function for getting a child (any or with some name) by position (positive or

negative). The function was of more than 50 Java lines, but the corresponding XPath expression is trivial: '`name[i]`'.

Navigation over tree is quite a simple task, but anyway the code may contain bugs. Fixing them takes developers' time and resources. It's better to integrate an already existing and tested XPath library.

Another maintenance task with the library mentioned was its improving. For example, initially the function supported only positive positions. We also were in need for more sophisticated filters. Without switching to XPath, there were two ways to continue. One was to introduce new similar functions and suffer from code duplication. Another was to make the function more complicated. But this way leads to reinvention of XPath, and the quality of the result may be too poor comparing to XPath.

The issues above are technical. And here are some more items, now of the project management level.

First of all, XPath and some other XML standards are well-known and familiar to many developers. They either already know them or can learn them fast due to plenty of documentation and books. It is not so with legacy code. Developers may need much time before starting to understand and use home-grown libraries.

The next item is the mapping between legacy data and XML is not only for XPath. Implementations of other XML standards also may suit the needs of developers, providing the same benefits as for XPath.

The last but not the least is marketing. Support of popular specifications gives us yet another argument for advertising a product or software development services.

## 5 SXML, the Lisp Approach

Lisp is a family of computer languages. The most popular dialects are Common Lisp [4] and Scheme [5]. In Lisp, data is representing using so-called S-expressions [8], or Symbolic Expressions, or 'sexps', defined recursively.

Just as XML, S-expressions can represent any data structures, including hierarchical, and Lisp is good in processing them. There are a lot of discussions on comparing S-expressions, Lisp and XML (for example, see [9] and [10]). In our work we use combination of useful features of XML and Lisp worlds.

The most developed Lisp approach to XML processing is the SXML format [11] for Scheme. It defines an XML representation in terms of S-expressions and provides a growing number of tools [12].

The SXML format uses lists to represent an XML tree. In each list, the first symbol is the name of an XML element node, and other list items are the children of the node. There is a set of special symbols to specify non-element XML nodes. For example, list of attributes is stored under the special symbol '`@`'.

Here is the SXML representation of the sample XML document:

```
(article (@ (id "hw"))
```

```
(title "Hello")
(para  "Hello " (object "World") "!"))
```

Some XML standards are adopted to SXML. For example, there is SSAX [13], an XML parser in Scheme with SAX interface, SXPath [14], an analogue of XPath, STX [15], a compiler for a subset of XSLT, and there is possibility for (X)Querying XML in Scheme [16].

A useful feature of SXML tools is possibility to mix processing enforced by a standard with processing in Scheme. It can be used to write concise and effective programs. For example, the paper [17] demonstrates practical tasks for which SXSLT suits better than XSLT.

While most of the SXML tools are just analogues of XML standards, some core XML standards are implemented without deviations. Among these tools are the SSAX XML parser [13] and the DDO SXPath [18] library for XPath processing.

## 6 XML Virtual Machine

The previous sections advocated that instead of developing legacy methods for hierarchical data processing, it's better to use approaches based on XML standards.

The simplest approach is to export data as XML, process it and import back. While it works well for one-time tasks, permanent serialization and deserialization is not an efficient solution.

The way of implementing a given XML standard in given programming language and data structure is not effective due to plenty of different software platforms and data formats. Developing XML standards complaint code is quite a complex task which shouldn't be re-done from scratch each time.

What is required is a retargetable library for XML processing which can be adopted to any environment. We are not aware about XML tools which position themselves so. The best matches are limited to Java or C# worlds and can't be used, for example, in C.

For our developments we decided to use a simple virtual machine (VM) which can be implemented in any language without significant efforts. Mapping between VM XML nodes and actual legacy tree nodes is a part of VM itself. So our code for XML processing isn't polluted by low-level details management.

While searching for an appropriate VM, we got an idea that instead of writing a program for a chosen VM, it's better to write an application in some best-suiting, maybe specially created, language and design a specific VM specially for the application.

The language of choice was Scheme. The main reasons for selecting it were its excellence in symbolic computations (in particular, XML-like processing) and the already existing SXML library [12]. Then, having code in Scheme, it was quite natural to select the core Scheme itself as a VM.

That's what we call the XML Virtual Machine: SXML library plus Scheme VM plus seamless integration of S-expressions and legacy data.

There is plenty of Scheme implementations, many of them are embeddable (for example, Guile [6] is for C, SISC [7] is for Java), so we already have a solid basis for having different XML VM implementations.

The biggest problem in this approach is the seamless integration. Code of a Scheme implementation should be written in such a way that supports a transparent mapping between S-expressions and legacy data structures. It's required for some Scheme functions, especially such as `car` or `map`, to work as over legacy data as over S-expressions.

There are other issues which should be taken in account when using a Scheme implementation as a VM. Of the main importance are garbage collection, continuations and error handling. These and related issues are discussed in more details in the separate paper [19].

The SXML library can be used in the XML VM without modifications, but some tweaking is desired. Due to the nature of S-expressions, the SXML tools can't get the parent or the preceding nodes without tricks which are sometimes quite inefficient. But, having a mapping, it's better to implement straightforward VM-level functions `native:parent`, `native:preceding-sibling` and similar, and make the SXML library to use them.

## 7 Sample: Python AST as XML

There is an example of usage of the XML VM in the Python programming language for representing Python parse trees [20] (abstract syntax trees, AST) as XML on top of the Scheme implementation Pysch [21].

The distribution package contains a tool to export Python AST as XML and a transformation example written both in XSLT and SXSLT. Both code perform conversion of an AST tree to a graph description for a graph drawing tool.

Pysch is a custom Scheme implementation for Python. The main purpose for writing it was to check our statement that a new Scheme implementation (and so an XML VM) can be created easily and fast. Experience shows that it's possible to prototype a compliant system in two-three working weeks.

Pysch is written in the object oriented manner. All navigation over S-expressions is localized in the `pair` class in the functions `car` and `cdr`. We derive the `lazypair` class from it to simplify lazy instantiation of S-expressions on demand. The further derivative is a family of `astpair` classes that lazily map AST objects to SXML.

SXML was not tweaked in this development because we didn't need it. However, it is quite a trivial task to do if required.

## 8 Sample: GNU 'find' with XPath over the Filesystem

Many XPath tutorials suggest an analogue between XPath expressions and UNIX file system paths, and in our developments we have added XPath capability to the standard GNU utility `find` [22]. Example:

```
$ ./find -xpath '/bin/*[@size > /bin/bash/@size]'
/bin/ipv6calc
/bin/rpm
```

But making the analogue alive is not the main idea of this sample. Instead, we demonstrated that it's possible to take third-party software that deals with hierarchical data and add an XML functionality without complete rewriting of the software.

In this sample the Scheme VM is a special version of Guile, Scheme implementation in C, which supports lazy pairs [23]. The nodes of the global S-expression which represents the filesystem as SXML are instantiated on demand.

In order to switch between the file metadata (C layer) and nodes in the SXML tree (Scheme layer), the application maintains a map between them. One of the uses of the map is to apply `find` predicates to nodes selected by an XPath query.

XPath expressions are evaluated by the txpath [12]. This lirary is not conformant to the XPath specification, so in the future we will integrate the DDO SXPath library [18]. Anyway, txpath works quite well, and we demonstrate how to tweak `parent` and similar problematic axes to use the native versions instead of the library default versions.

## 9 Related Work

First of all, there is plenty of applications which give access to internals by exporting XML. It is sufficient for off-line processing, but it may degrade performance a lot.

As for adopting to different tree models, it is quite common now. For example, Jaxen [24], Java XPath engine, is positioned as 'a universal object model walker, capable of evaluating XPath expressions across multiple models.' [25] There is also Saxon [26], XSLT and XQuery processor, which uses a tree navigator interface to support different types of trees. It should be possible to use Jaxen and Saxon with non-XML trees, but we are not aware about developments in this area.

The step beyond XML world was made by the Java system JXPath [27] from the Apache Software Foundation and by .NET API `XPathNavigator` [28] from Microsoft. JXPath applies XPath expressions to graphs of objects of all kinds: Java Beans, Maps, Servlet contexts, DOM etc, including mixtures thereof. The `XPathNavigator` facilitates the ability to perform XPath queries over any store implementing the `IXPathNavigable` interface. It is noteworthy that MSDN article [28] provides an example in which the tree structure of the file system is mirrored in the tree structure of the exposed XML Infoset, which is somehow similar to our example of XPath over the file system.

The main limitation of the systems mentioned is their attachment to a specific language or a platform. They don't suit an application if it is written in another language. The libraries described cover only Java and .NET, and we are not aware about similar developments for other languages.

Another limitation is that these libraries (except Saxon) support only XPath and don't support XSLT or XQuery.

As for virtual machines, authors of the XPath and XSLT systems sometimes express the idea that their internal representation of parsed expressions and execution plans looks like a basis for a virtual machine, but there are no traces of developments in this area.

The only real XML-related virtual machine we aware of is XSLTVM [29] from Oracle. XSLTVM is the software implementation of a 'CPU' designed to run compiled XSLT code. The objectives are very similar to our ones, but the XSLTVM paper doesn't say anything about seamless binding of legacy data and VM data.

The are a few developments which use the phrase 'XML virtual machine', but in their case it means not a basis for developing XML tools, but some frameworks for XML processing.

## 10 Conclusion and Future Work

In this paper we suggested the idea of XML view on legacy hierarchical data and proposed to use XML tools over this view. For technical implementation we introduced the XML virtual machine based on the Scheme programming language. The approach is tested by developing reference implementations and the results look good.

There are several directions for future work.

One is to make more reference implementations for other popular platforms such as Java and .NET.

The SXML XPath implementation is good, but still needs more testing and debugging. We are going to port test suits from other XPath processors to check the SXML implementation of XPath.

Interesting tasks are optimization of evaluation of XPath expressions and compilation of them into source or executable code.

Finally, we'd like to start working on a retargetable implementation of the XSLT 1.0 standard.

## References

1. World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/
2. World Wide Web Consortium. XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116
3. World Wide Web Consortium. XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116
4. LispWorks Ltd. Common Lisp HyperSpec. http://www.lispworks.com/documentation/HyperSpec/

5. R. Kelsey, W. Clinger, J. Rees (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/
6. Free Software Foundation. Guile (About Guile). http://www.gnu.org/software/guile/guile.html
7. Scott Miller. SISC - Second Interpreter of Scheme Code. http://sisc.sourceforge.net/
8. Ess Expressions. http://c2.com/cgi/wiki?EssExpressions
9. Lisp Vs Xml. http://c2.com/cgi/wiki?LispVsXml
10. Xml Isa Poor Copy Of Ess Expressions. http://c2.com/cgi/wiki?XmlIsaPoorCopyOfEssExpressions
11. Oleg Kiselyov. SXML Specification. http://okmij.org/ftp/Scheme/xml.html#SXML-spec
12. Oleg Kiselyov. S-exp-based XML parsing/query/conversion. http://ssax.sourceforge.net/
13. Oleg Kiselyov, A better XML parser through functional programming, LNCS 2257, pp. 209-224, Springer-Verlag, January 2002.
14. Oleg Kiselyov and Kirill Lisovsky. XML, XPath, XSLT implementations as SXML, SXPath, and SXSLT. International Lisp Conference: ILC2002, October 2002.
15. Kirill Lisovsky. STX: XSLT-like XML transformation in Scheme. http://www196.pair.com/lisovsky/transform/stx/
16. Jim Bender. (X)Querying XML in Scheme. http://celtic.benderweb.net/webit/docs/xquery-pre/
17. O. Kiselyov and S. Krishnamurthi. SXSLT: Manipulation Language for XML. In PADL, LNCS 2562, 2003.
18. Dmitry Lizorkin. DDO SXPath. http://modis.ispras.ru/Lizorkin/ddo.html
19. Oleg Paraschenko. Using DSLs on top of Scheme VM. To be published soon on the author's website http://uucode.com/
20. Oleg Paraschenko. Python AST as XML. http://pysch.sourceforge.net/ast.html
21. Oleg Paraschenko. Pysch: Scheme runtime environment in Python. http://pysch.sourceforge.net/
22. Free Software Foundation. findutils - GNU Project - Free Software Foundation (FSF). http://www.gnu.org/software/findutils/findutils.html
23. Oleg Paraschenko. Lazy pairs for Guile. http://uucode.com/texts/lazypair/
24. jaxen: universal java xpath engine - jaxen. http://jaxen.org/
25. jaxen: universal java xpath engine - FAQ. http://jaxen.org/faq.html
26. Michael Kay. The SAXON XSLT and XQuery Processor. http://saxon.sourceforge.net/
27. The Jakarta Project. The JXPath Component. http://jakarta.apache.org/commons/jxpath/index.html
28. Microsoft Corporation. XPathNavigator in the .NET Framework. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconXPathNavigatorOverDifferentStores.asp
29. A. Novoselsky and K. Karun. XSLTVM - an XSLT Virtual Machine. XML Europe 2000, Paris, France, 12-16 June 2000. http://www.gca.org/papers/xmleurope2000/papers/s35-03.html