XSL STYLESHEET COMPILER

Marek BĚHÁLEK, Doctoral Degree Programme (1)
Department of Computer Science, FEI VŠB - TU Ostrava
E-mail: marek.behalek@vsb.cz

Supervised by: Dr. Miroslav Beneš

ABSTRACT

XML is a flexible way how to create self-describing data and share them trough the Internet. XSLT provides a complementary language describing how to perform a transformation of a source XML document into a resulting new XML document. XSLT defines transformations which should be processed but not how to do it. Almost all XSLT processors (applications that perform transformations) that are currently available work like interprets. Similarly like in programming languages, using a compiler instead of an interpret can also dramatically improve performance. This article describes different approaches for creating such a compiler.

1 INTRODUCTION

When we are building an information system we must think about a support for small devices like a PDA or a cell phone. Nearly every person has a cell phone. These devices are smarter from year to year. Now they support Java and it's only a matter of time when we will truly need a support of such devices in our information system. PDAs are really computers, they are not only devices for viewing data but can run complex applications. Some PDAs are with their computational performance and amount of memory comparable to desktop computers from not so far history. Way how we present a data and how we process a transformation of this data is even more important for these devices with restricted performance. Technologies based on XML seem to be the right solution.

1.1 XML

XML - eXtensible Markup Language - is well known language that describes a set of rules for defining semantic tags that break a document into parts and identify different parts of the document. It is a meta-markup language that defines the syntax used to define other domain-specific, semantic, structured markup languages. It is very well documented by W3C [4], can be 100% ASCII text and is freely available. It is commonly known and nearly every modern programming language integrates tools and libraries for processing

XML documents. XML is a flexible way how to create *self-describing data* and to share both the format and the data on the World Wide Web.

1.2 XSL

XSL is a language for expressing stylesheets. It consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. In this article we discuss the first one, used for transforming XML documents. Well-formed XML document has a tree structure. A transformation expressed in XSLT describes rules for transforming a source XML tree into a resulting XML tree. This operation covers a wide range of processing tasks including: transforming XML document to HTML page, searching for data in XML documents or creating *document views* (same data sorted by different criteria, extracting sensitive information, etc.).

An XSLT stylesheet is an XML document that uses elements from XSLT vocabulary to describe a transformation. The document element is an `<xsl:stylesheet>` element whose contents is a set of rules describing the transformation to be performed. Each rule in stylesheet contains an associated XPath pattern that defines the node in the source document to which the rule should apply. Each rule is called a template and is represented by an `<xsl:template>` element with a `match="pattern"` attribute for XPath pattern. Each rule is called a template because the literal elements and attributes contained inside the body of the rule act as a model for constructing some part of the result tree.

XPath is a language for addressing parts of an XML document. This is a primary purpose of this language, but it also provides basic facilities for manipulations of strings, numbers and booleans. XPath pattern is a location path in a tree formed by XML document, where each step (separated by the slash) selects a set of nodes relative to the context node.

2 DESCRIBING THE PROBLEM

Let us consider the model situation: Our application is based on Client-Server architecture. The Server is a powerful computer with a lot of memory and a huge computational performance. The Client can be a device like Pocket PC, with limited memory and computational performance, which is connected to the Internet by a cell phone. This connection is expensive and rather slow. Such a client worries about amount of data that must be downloaded and stored. It does not want to be *on-line* all the time. In these cases XML and XSLT can help as a powerful tool. Client downloads data like an XML document only once and when he wants a different view on the same data, he simply applies some XSL transformation. XSLT stylesheet is smaller because it only describes an operation that must be performed and does not contain all data again. Another advantage is that the client can store this XSLT stylesheet and use it when needed *off-line*. Using XML for data representation and XSLT for creating different views over this data can save usage of memory and amount of data for downloading but now it is the client that must accomplish the transformation. This may consume a lot of time.

Most of XSLT processors work like an interpret.We can dramatically increase performance when we compile XSLT stylesheets (see Fig. 1). The reason is the same as in Java or C languages. Execution of binary form is much faster. Another advantage is that

execution of binary form does not require the presence of the original translator, only an appropriate physical or virtual machine. In our example it means that the transformation is quickly performed on the server side and only a relatively small binary form is sent to the user. Also the XSLT processor, the application that performs the transformation, is divided into two parts — a compiler that can be only on the server side and a virtual machine which can be very simple.

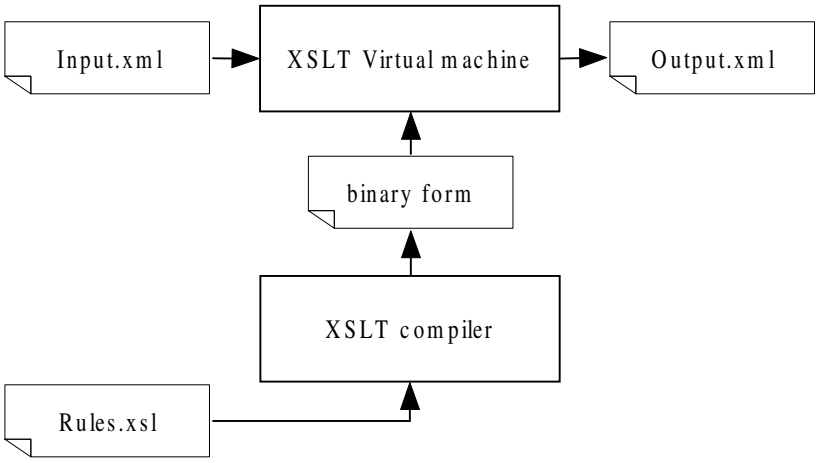

Figure 1: XSLT compiler scheme

Using a compiler instead of an interpreter clearly separates compile-time computations from run-time computations. The separation line depends on the level of machine instructions. Splitting a heterogeneous, high-level instruction into number of atomic, low-level instructions exposes some run-time checks and allows compiler to take care of them. For example, a general CMP comparison instruction checks operand types at run-time. If it is replaced with few type-specific variants then the compiler checks operand types at compile-time and generates appropriate type casting instructions if needed.

Another advantage of such an approach is that once compiled the XSLT stylesheet in binary form can be used for this concrete transformation many times, without need of translation of the original XSL stylesheet again. This may be very useful, considering a situation when a Web server generates HTML pages from XML documents using XSLT stylesheets. This is an easy way how to add presentation qualities to pure content-oriented XML. The interpreter must parse XSLT and then perform the transformation. Time for parsing XSLT can be comparable to time when relatively small XML tree is processed. Compilation of XSLT can greatly improve performance of such a Web server when it is used by many users at the same time.

3 RELATED WORK

The concept of compiling XSLT to binary form was proposed by A. Novoselsky and K. Karum [1]. Their article presents a concept of an XSLT Virtual machine. Their XSLT Virtual machine is a software implementation of an imaginary CPU designed to run a compiled XSLT code. The instruction set and the memory management of this imaginary

CPU is adapted for processing XSLT stylesheets. This XSLT processor was presented only like a concept. Exact specification does not exist, there is only a short paper with proposed solution. Currently there also does not exist any implementation of such a virtual machine or XSLT compiler.

The idea of XSLT compiler is built into XSLT processors of same companies: Oracle Inc., Ambrosoft Inc. or Apache Software Foundation. All these XSLT compilers follow the same approach. The binary form that they produce is a Java *class* file. All these products are based on Java.

Another platform commonly used with PDAs is a restriction of Microsoft .NET framework called Compact.NET. This environment should also support XSLT compilation techniques, but in the current version 1.0 it does not seem to be available yet.

4 PROPOSED SOLUTION

Using a *class* file as the XSLT compiler binary form binds it too close to Java Virtual Machine. It is nearly impossible to add support for a different platform (for example Microsoft .NET Framework). I propose to compose the output binary form from an instruction set of a virtual CPU for an XSLT stylesheet processing. Then a binary form generated by XSLT compiler is a sequence of these XSLT CPU's instructions and the runtime environment is a virtual machine processing this instruction set.

This approach has one great advantage. The platform used by a provider of such a compiled stylesheet is independent of the client's platform. The compiler makes a binary form — XSLT *program*. Once compiled XSLT *programs* can be run many times on different XSLT Virtual machines running with different platforms. Also, XSLT *programs* become independent of the implementation language. In other words, there should be no difference between code generated from Java XSLT compiler and code from C/C++/C# XSLT compiler, they all should be able to run on the same XSLT Virtual machine.

This approach may bring another improvement in the process of optimization. For example in an XSLT Virtual machine implementation we can support special abilities of a destination platform.

4.1 XSLT CPU SPECIFICATION

The full description of the XSLT CPU is beyond possibilities of this paper. We present only main properties of XSLT CPU that was created and implemented like a virtual machine.

Virtual machine has a stack-based architecture. This goes hand-in-hand with the tree structure of an XSLT stylesheet.

The compilation consists of two relatively independent activities — creating the representation of XPath elements and XSL templates.

As mentioned, separation between compile-time computations and run-time computations depends on the level of machine instructions. Low-level instructions (comparable to JVM instructions) was decided to use for purpose of XSLT CPU. XSLT stylesheets are commonly short. The size of a generated binary form is generally small and the compiler can perform a lot of checks at a compile time. Instructions are mostly *stack-based*.

They take their operand's values from the top of the stack and replace them with the result. They can be divided into three groups: system instructions (loops, conditional instructions, instructions for arithmetic operations,...), instructions for processing XPath elements and instruction for processing XSL templates. When creating the instruction set it is not necessary to take care about elementary XML processing. For these activities high level instructions are used. We suppose that nearly every platform implements some libraries for XML processing that can be simply mapped into these instructions in the virtual machine implementation.

5 EXPERIMENTAL RESULTS

A prototype of the presented XSLT compiler is currently implemented reusing some parts of the open-source product Xalan written in Java. The most important change was in a process of binary form creation. Instead of a Java class file the new compiler generates a file with instructions for the presented XSLT CPU — XSLT *program*. The run-time environment (virtual machine) is implemented in C# to be able to run in the Microsoft .NET Compact Framework used on the client side of our information system. The full implementation of this XSLT compiler and run/time is still in progress.

6 CONCLUSIONS

The XSLT stylesheet compiling technique can be useful also for desktop applications. But it brings a great advantage for XSL transformations processing when small devices (like PDAs) are used. Then this technique can greatly improve performance and minimize a limited memory usage. That can make XSL transformations even more useful and bring them to areas where no one would expect them.

REFERENCES

[1] Anguel Novoselsky, K. Karun.: *XSLTVM - an XSLT Virtual Machine.*
See: http://www.gca.org/papers/xmleurope2000/papers/s35-03.html/.

[2] Jacek R. Ambroziak.: *Gregor, the next generation XSLT compiler.*
See: http://www.ambrosoft.com/.

[3] The Apache Software Foundation *Xalan-Java documentation*
See: http://xml.apache.org/xalan-j/.

[4] World Wide Web Consortium.: *Extensible Markup Language (XML) 1.0.*
See: http://www.w3c.org/xml/

[5] World Wide Web Consortium.: *XSL*
See: http://www.w3c.org/xsl/

[6] Oracle Corporation.: *Transforming XML with XSLT.*
See: http://technet.oracle.com/tech/xml/techinfo.html.