An Analysis Of Bytecodes Produced By XSLTC

Allan Kielstra,

Henry Zongaro

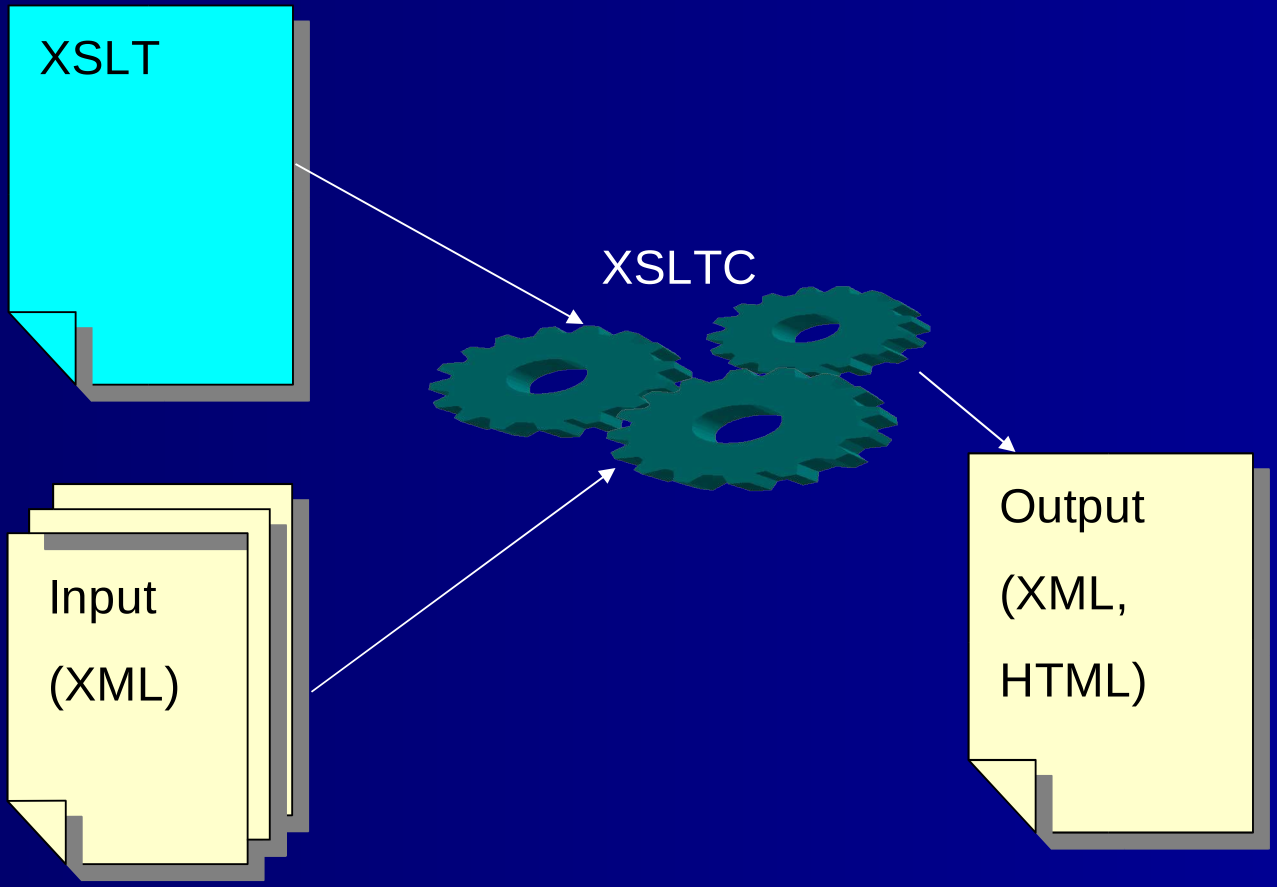
Overview

Overview of XSLT

Overview of the compiler

Characteristics of machine generated bytecodes

XSLTC Processing



XSLT

XSLTC

Input

(XML)

Output

(XML,

HTML)

XSLT: in brief

- Processor transforms XML to XML or HTML
- Models input document as tree of nodes
- XSLT stylesheet consists of templates
  – specify patterns against which processor attempts to match nodes in input
  – if node matches a template’ s pattern, template is executed for that node
  – usually will create part of new XML output

Example XSLT:

http://www.w3schools.com/xsl/xsl_transformation.asp

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0“
xmlns:xsl=http://www.w3.org/1999/XSL/Transform” >
<xsl:template match="/">
  <html>
  <! – Deleted from example HTML>
  <xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
  </tr>
  </xsl:for-each>
  <! – Deleted from example: HTML >
  </html>
 </xsl:template>
 </xsl:stylesheet>
```

XSLT: in brief

1. Processing begins with root of input tree as *context node*
2. Attempt to match context node against templates
3. If no match, make each child of node the context node in turn, and recursively apply step 2
4. Otherwise, matched template executes for context
   - may create part of result
   - may request recursive jump to step 2 for any set of nodes in input (or even in any other document); most often, just children or nothing

XSLT: In brief

- Nodes are selected and matched using *XPath expressions*
- XPath is hierarchical navigation system for XML, similar to file paths/URIs
  - a/b/c selects “ c” element children of “ b” element children of “ a” elements that are children of the context node

Example XPath

```
<cnode>
<a>
   <b>
        <c> element </c>
        <c> element </c>
   </b> <b>
        <c> element </c>
   </b>
</a></cnode>
```

XSLTC Details

Several XSLT processors, including Apache XSLTC (XSLT Compiler)

- compiles stylesheet to Java class(es)
- compiles templates to methods
- methods implement mutually recursive processing described
- relies very much upon run-time support classes

XSLTC Details

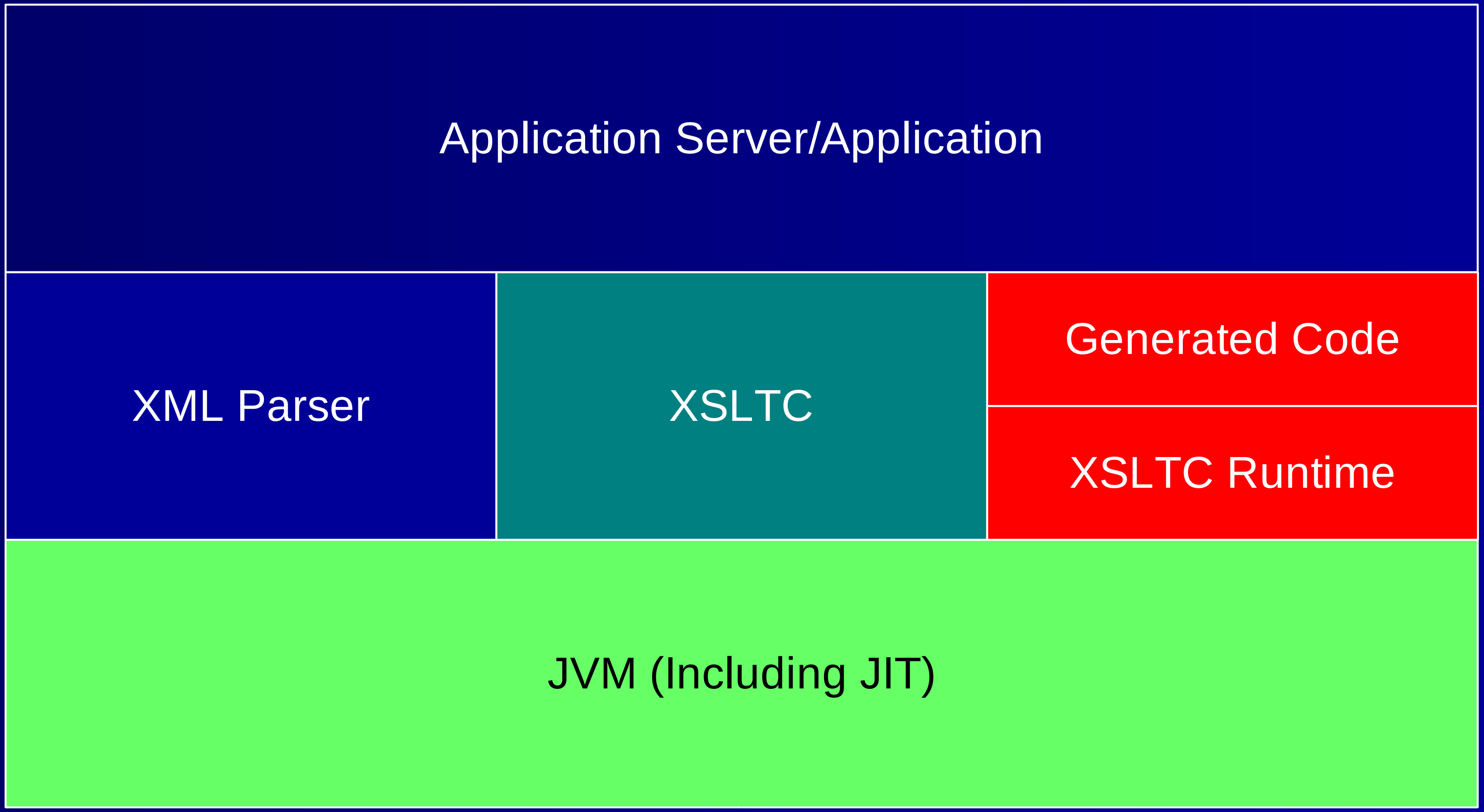
Run-time support includes evaluation of each step in XPath expressions

- in expression like a/b/c, *iterator* is responsible for evaluation of each step
- generated code asks run-time to create iterator to evaluate that step, relative to context node
- iterators are composed to evaluate complete path expression

Some Other Bytecode Generators

- JSP Compiler
  - Sovereign optimizer runs in time O ($2.3^{\text{bytecodes * number trys}}$)
- Bytecode obfuscators
  - JITs work best with reducible CFGs

Elements Of Interest



Application Server/Application

XML Parser

XSLTC

Generated Code

XSLTC Runtime

JVM (Including JIT)

Tracking Names

- Many operations on data type X have a predilection for using the name “ x”
  - Save current “ x” on object of type “ X”
  - Initialize “ x” for new use
  - Use “ x” for some purpose
  - Copy saved value back to “ x”
- Result of compiler “ pushing” current state

Parameter passing

If it requests recursive processing of nodes, template can pass parameters

- but pattern matching involved
- which parameters are passed and which expected by matching templates not known statically
- a stack class is used to push parameters at call site, then template that matches checks stack for parameters it needs

Parameter passing

For convenience, same mechanism was used for explicit procedural calls

- should really use parameters on Java method
- no confusion about which method is invoked

Parameter passing

```
pushParamFrame();
integer = …;
addParameter(integer);
proc();
popParamFrame();

proc() {
  integer = …;
  Object object =
   addParameter
   (integer);
}
```

```
Object integer = …;
proc(integer);

proc(Object arg) {
}
```

Large, General Purpose Methods

- It’ s easier for the compiler to generate a single method for a single template.
- Template parameters are not guaranteed to have known types.
- This leads to a lot of downcasts followed by instanceof predicates

Example (concrete 1)

```
void m()
  {
  Concrete c1;
  templ_1(c1);
  }

void templ_1(Object o)
  {
  rt_methhd(o);
  }
```

```
void rt_methd(Object o)
  {
  if (o instanceof
      Concrete)
      {
      // do something
      }
  }
```

Example (concrete 2)

```
void m()
  {
  Concrete c1;
  templ_c(c1);
  }

void templ_c(Concrete c)
  {
  rt_concrete(c);
  }
```

```
void rt_concrete(
  Concerete c)
  {
  // do something
  }
```

Use of Larger Data Types

- Virtually all (user) numeric scalars are of type double.
- Chars are frequently treated as single character strings.
  - Leads to uses of string.equals(“ a”)

Caching Of Iterators

- Some complex transformations require iterating over the same document elements multiple times
- It’ s easier for XSLTC to use fresh iterators every time
- Can lead to excessive GC

Example (Iterators 1)

```
void m()
  {
  for ( ; ; )
     {
     draw_item()
     }
  }
```

```
void draw_item()
  {
  i1 = new Iterator();
  i2 = new Iterator();
  //
  for ( ; i1 ; ) //
  for ( ; i2 ; ) //
  }
```

Example (Iterators 2)

```
void m()
  {
  c = new Cache();
  c.i1 = new Itr();
  c.i2 = new Itr();
  for ( ; ; )
      {
      draw_item(c)
      }
  }
```

```
void draw_item(Cache c)
  {
  renew c.i1;
  renew c.i2;
  //
  for ( ; c.i1 ; ) //
  for ( ; c.i2 ; ) //
  }
```

Recursive Application Of Templates

- Document elements are often described recursively
- Required transformations can, therefore, be applied recursively
- Document/Iterators are always in the form of a tree
- Each document is only visible to a single thread

Example (Recursion 1)

```
void apply_template(DOM d, Iterator itr) {
   {
   while ((i = itr.next()) > 0)
       {
       switch (d.gettype(i))
             {
             case a: //
             case b: //
             case x:
                    apply_template(d,d.child(i));
                    break;
             }
       }
   }
```

Example (Recursion 2)

```
void apply_template(DOM d, Iterator itr) {
   {
   itr.parent = null;
   for (sitr = itr; itr; sitr = sitr.parent)
      {
      while (sitr.next() > 0)
           {
           switch
              {
              case a: //
              case x:
                 d.child(sitr.getPosition()).parent =
                 sitr;
                 sitr = d.child(sitr.getPosition());
              }
           }
   }
```

Results (Including B2B)

	XSLTC 2.5.1	Latest
dbonerow	424.38	319.08
queens	16.58	8.57
identity	62.96	48.62

Results (Xerces Parser)

	Old	New
queens	16.58	10.11
identity	62.96	58.90
SalesSearch	317.02	121.51
ViewItem	3.48	2.93

Tricks That XSLTC Could Play

- Caching and re-use of objects
- Specialization of templates

Cheats: Communications between XSLTC and JITC

- Elimination of checks such as array bounds checks
- Specification that specific allocations are thread local
- Set compilation threshold/hotness.