Programming with XSLT

Venkat Subramaniam
venkats@agiledeveloper.com

September 2003

Presentation and examples can be downloaded from
http://www.agiledeveloper.com/download.aspx

Abstract

Abstract Transforming an XML document using XSLT provides several advantages. However, it is not some thing that is too easy to begin with. This presentation is intended to demystify programming with XSLT. The speaker presents easy to understand analogies between XSLT and notions most of us are very familiar with. He then presents a quick over view of XPath notation and shows how to perform transformations and effectively render the contents of an XML document.

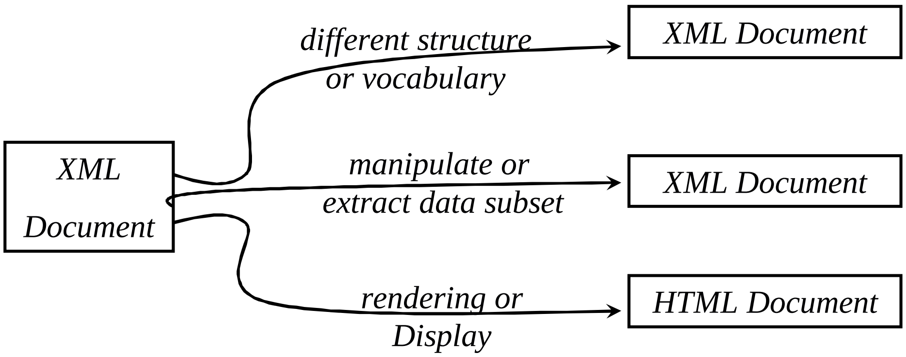
Speaker Dr. Venkat Subramaniam, founder of Agile Developer, Inc., has trained and mentored more than 2,500 software developers around the world. He has significant experience in architecture, design, and development of distributed object systems. Venkat is an adjunct professor at the University of Houston and teaches the Professional Software Developer Series at Rice University's Technology Education Center. He may be reached at venkats@agiledeveloper.com.

Examples Any page with a ★ has an example attached
Download from http://www.agiledeveloper.com/download.aspx

Programming with XSLT

- **Why Transform?**
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- Server side Processing
- Conclusion

Why Transform?



XML Document

different structure or vocabulary → *XML Document*

manipulate or extract data subset → *XML Document*

rendering or Display → *HTML Document*

Need to transform

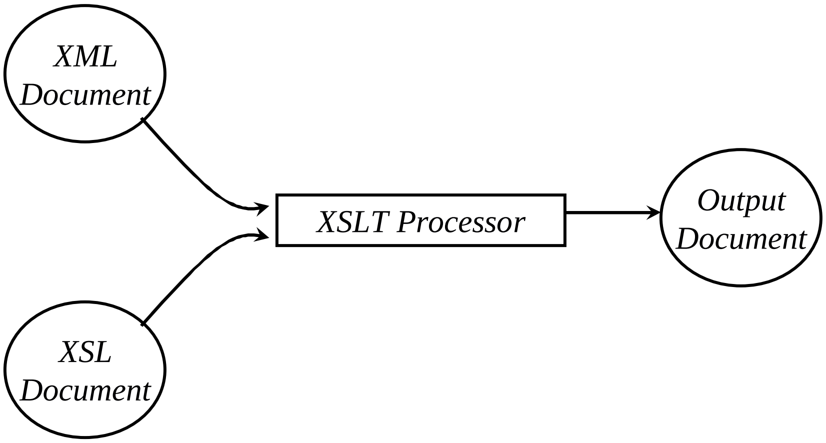
- May have to represent information
  - in a different format
  - in a different structure
  - in a different vocabulary
- Transformation may be
  - Structural transformation
    - May want to go from one vocabulary to another
  - Dynamic document generation
    - May want to sort, filter
  - Rendering
    - May want to present information in a browser, etc.

Programming with XSLT

- Why Transform?
- **XSLT**
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- Server side Processing
- Conclusion

XSLT

- An XML based language to transform XML



XSL

- eXtensible Stylesheet Language
- XML based language for transformation
- It comes in three flavors:
  - Transformation (XSLT)
  - Rendition (XSLF)
  - Accessing underlying structure (XPath)

- XSL servers as a root for
  - Cascading Style Sheets (CSS)
  - Document Style Semantics and Specification Language (DSSSL)

XSLT

- XSLT is a language written in XML
  - A well-formed XML document that tells how to transform an XML document
- An XSLT processor will do actual transformation
  - They manipulate the structure of a document not its contents
- They work on the ***Grove*** of a document

XSLT Traits

- Declarative in nature
  - you say what you want, not how to do things
- Works on the structure of three documents
  - input, stylesheet and output documents
- Works by matching nodes and templates
- Not too easy
- Requires quite some abstract thinking to write it correctly

XSLT Processor and Trees Models

- The source document is an XML document with an underlying tree model of the structure
- The destination document is an XML document with an underlying tree model of its structure
- The XSLT style sheet is also an XML document which also has an underlying tree model
- An XSLT processor works with three models

XSL Processing

- XSL is declarative
  - It specifies what the result should look like
  - not how to do it (this is the job of a processor)
- The style sheet is a template that specifies how each node of the source tree must look in the result tree
  - specifies a pattern of match and replace
  - also includes applying a "template" on matching a node

Goals of XSL

- Support browsing, printing, interactive editing
- Both work with traditional and web environment
- Support interaction with structure information
- Support both visual and non-visual presentation
- Should be declarative
- Support simple formatting without precluding complex ones
- Support extensiblity
- Should be in XML format
- Should be human readable, and reasonably clear

Programming with XSLT

- Why Transform?
- XSLT
- **XPath**
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- Server side Processing
- Conclusion

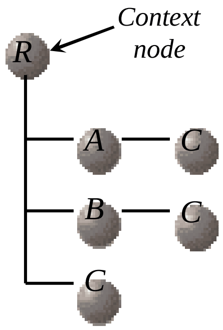
XSLT uses XPath

- XPath is the **query** language used in XSLT
- First imagine a XML document as a tree of nodes
- Then imagine standing on a node to perform query
  - this is your *context* node
- Now issue a query to select zero or more nodes to process
- How do the queries look?
  - here we only look at an abbreviated form of XPath

Very easy if you catch an analogy!

- Think of XPath query like a directory query on Unix
- How do you refer to
  - a file with name 'A' in current directory? `"A"`
  - all files in the current directory? `"*"`
  - all files named 'C' one level down from current dir? `"*/C"`
  - the current directory? `"."`
  - the parent of current directory?`'.."`
  - the root directory? `"/"`
- You got most of the syntax for XPath!

Lets try!



- How to get
  - child node A? `"A"`
  - child C of A? `"A/C"`
  - all grandchildren C? `"*/C"`
  - all nodes C under R? `"//C"`

Programming with XSLT

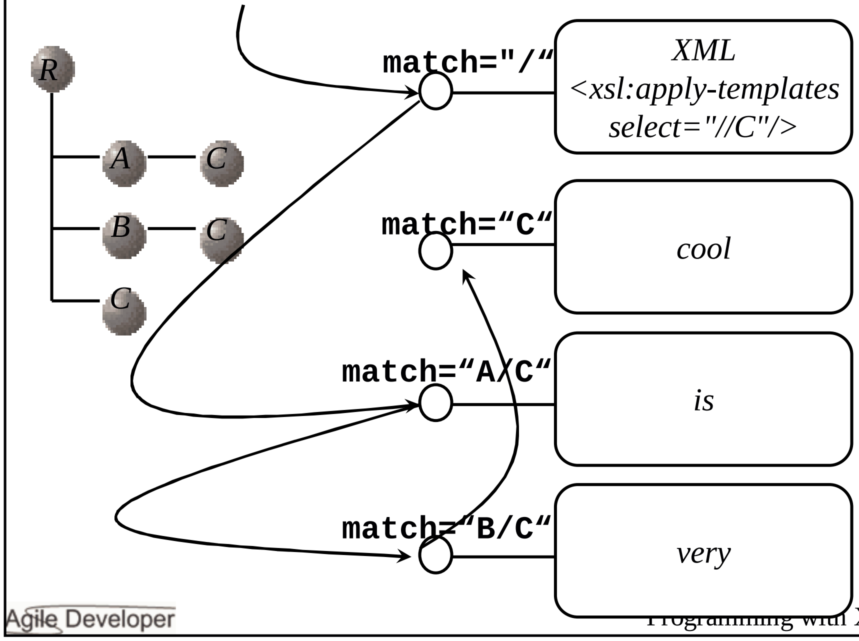
- Why Transform?
- XSLT
- XPath
- **Template Matching**
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- Server side Processing
- Conclusion

Template Match

- XSLT Processor sets the document root as context
    - this is the parent of root element (R’s parent in example)
- Looks for a template for "/"
    - A template contains contents to copy with plugged in values and instructions
        - Just like a template sample we would write for a report
- Executes the template, branching to other templates as instructed

---

Template Matching



`match="/“`

XML <xsl:apply-templates select="//C"/>

`match=“C“`

cool

`match=“A/C“`

is

`match=“B/C“`

very

Abbreviations

- Commonly used constructs have shorthand's
- child axis is the default if axis not specified
- You may use @ instead of attribute:: axis
- // may be used instead of /descendent-or-self::node/ axis
- . may be used instead of self::node()
- .. may be used instead of parent::node()
- *n* may be used instead of position() = *n*

```
child::Pizza[position() = 1]/
child::Toppings[position() = 4]
```

`may be written as`

`Pizza[1]/Toppings[4]`

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- **Writing Stylesheet**
- Rules
- Flow of control
- Insertions
- Server side Processing
- Conclusion

xsl:stylesheet

```
<xsl:stylesheet>…</xsl:stylesheet>
```

- Root node of every stylesheet
- Here is where all the templates would be written

xsl:template

```
<xsl:template match="context" >
 …
 </xsl:template>
```

- Formatting, transformation actions will be specified here
- match specifies the node(s) to which the template must be applied

xsl:apply-templates

```
<xsl:apply-templates
 select="context">
```

- Invokes the application of other xsl:template elements
- select attribute tells the node whose template must be invoked
- Recursively applied on children of source element

---

xsl:value-of

- <xsl:value-of select="Pattern">
- Used to insert text value of the node matching the pattern

Writing a Template

- XSL allows you to write a template
- Template provides the substitution text for a matching pattern
- Output format may be defined and an appropriate substitution may be specified

Quiz Time

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- **Rules**
- Flow of control
- Insertions
- Server side Processing
- Conclusion

Templates and Rules

- A template is specified as
  `<xsl:template match="…">`
- match attribute refers to a *pattern* of matching node

- "`/`" matches with root node (the document node)
- "`Book`" matches with an element with that name
- "`Book/Title`“ matches Title element whose parent is Book
- XSLT processor builds a tree view in memory of all templates
- It then starts by applying template for root node

apply-templates

`<xsl:apply-template select="…"/>`

- Processor will
  - first select nodes based on given select criteria
  - it then applies the template rules for each selected node

`<xsl:apply-template/>`

- This is a special case, where all children nodes of current node are selected

---

apply-templates and Modes

- Specifying select results in node selection and application of matching templates
- What if you want to take more than one for a set of nodes?
- You may use modes
- Modes are
  - names you provide as a `mode` attribute in xsl:template
  - when writing apply-template, you specify which mode / matching template to use

Built-in Template Rules

- If you do not write a template rule, there is a default rule that applies
- For root node and other elements, default is to call `<xsl:apply-templates/>` so each child node is selected and its matching template rule applied
- For attributes, to copy the value of the attribute as a text
- For text nodes, to copy the text to the result tree

Conflict Resolution

- Built-in templates are not applied if you write one
- What if more than one template matches a selection?
- Templates you write override those you import
- Templates that are specific are chosen over those that are general (like use of wildcard *)
- Templates may be given priority number
  - the matching one with greatest priority number is chosen
- What if you still have more than one
  - depends on the processor; better to avoid this situation

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- **Flow of control**
- Insertions
- Server side Processing
- Conclusion

Repetition

- You may specify repetitive action by using for-each

```
<xsl:for-each select="//Book">
…
</xsl:for-each>
```

- For each node selected, perform the action specified within the for-each element
  - Provides an alternate way to using apply-templates

Sorting

- You can sort while looping through the for-each

- Simply put
  - `<xsl:sort order="ascending" select="Title">`
    - to sort on increasing order of Titles
  - `<xsl:sort order="descending" select="Title" >`
    - to sort on decreasing order of Titles

Conditional Processing

- if construct is specified using <xsl:if>
  - You may use a test attribute to check if a condition is met

- if/else construct is specified using <xsl:choose>
  - Provides facility for checking multiple conditions
  - when element specifies action to be performed when a sub-condition is met

Common Types of XSLT elements

Purpose	Elements
Template Rules and invocation	<xsl:template>
	<xsl:apply-templates>
	<xsl:call-template>
Structure of stylesheet	<xsl:stylesheet>
	<xsl:include>
	<xsl:import>
Generate output	<xsl:value-of>
	<xsl:element>
	<xsl:attribute>
	<xsl:comment>
	<xsl:processing-instruction>
	<xsl:text>
Define variable and parameter	<xsl:variable>
	<xsl:param>
	<xsl:with-param>
Copy information	<xsl:copy>
	<xsl:copy-of>
Conditional Processing	<xsl:if>
	<xsl:choose>
	<xsl:when>
	<xsl:otherwise>
	<xsl:for-each>
Ordering	<xsl:sort>
	<xsl:number>
Output formatting	<xsl:output>

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- **Insertions**
- Server side Processing
- Conclusion

Copying – xsl:copy

- Copying is useful in keeping the structure the same
- Manipulating the content being transformed from one XML document to another

- <xsl:copy> element is used

- You specify what action to perform on content while keeping the structure identical as source

---

xsl:comment

`<xsl:comment>`

- For outputting a comment
- The format of text should follow XML commenting format

xsl:pi

`<xsl:pi name="processing-instr-name">`

- Will put the processing instruction in the output document

---

xsl:element

`<xsl:element name="elementName">`

- Creates an element in the output document
- The name of the element to be created is specified as attribute

xsl:attribute

```
<xsl:attribute name="attributeName">
```

- Creates and puts it into the element in which this statement is present
- Value of the xsl:attribute element is the value of the output attribute

---

xsl:variable

- Global variables may be defined
- Local variables may be defined within a template

```
<xsl:variable name=“varName“
  select=“value“/>
```

- where variable’s name is *varName;* value is *value*
- Types of variable are String, Number, Boolean, Node-set and Tree
  - Once a value is given, you can’t change them!

xsl:call-template

- Used to invoke a named template
- Similar to a procedure or function call
- Lets you decide the template to be used rather than letting processor pick one
- You may send parameters as well

---

xsl:number

- Allows you to generate sequential number for nodes
- Also allows formatting of the number
- You may specify the
  - level
    - Single level (peer nodes), any level or multiple (hierarchic numbering)
  - count
    - a pattern for the node selection
  - from
    - a pattern for node from which sequence starts over
  - and formatting information

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- **Server side Processing**
- Conclusion

Server Side Transformation

- Examples shown till now transform on front end
- What about transforming on the server side

- You may use processors like Xalan to do just that

Quiz Time

Programming with XSLT

- Why Transform?
- XSLT
- XPath
- Template Matching
- Writing Stylesheet
- Rules
- Flow of control
- Insertions
- Server side Processing
- **Conclusion**

Conclusion

- Transforming XML has its advantages
- XSLT is pretty powerful

- XPath syntax pretty much drives it

- Some challenges between processors and versions

References

- http://www.w3.org/Style/XSL/

- XSLT Programmers Reference, Michael Kay, Wrox Publications

- http://www.agiledeveloper.com/download.aspx

- http://xml.apache.org/xalan-j/index.html