

XSLT Tutorial

XML Finland 2001

Kimmo Rytönen

TietoEnator, Telecom & Media

XSLT Tutorial

- **Intended audience**
 - programmers
 - Web application developers
 - XML implementors
- **Preknowledge**
 - HTML and XML basics
- **Objectives**
 - to introduce the basics of XSLT programming
- **Content**
 - XSLT standard
 - XSLT language essential
 - examples
 - tips
 - XSLT programs
 - xt, XML Spy
- **Material**
 - in proceedings
 - handouts

Glossary

- **XSL stylesheet consists of presentation and transformation rules**
- **XSL(T) parser interprets XSL stylesheets**
- **XSLT program is an XSL stylesheet**
- **XSL instruction element is an element in the XSL namespace**
`<xsl:template>This element is the XSLT template rule </xsl:template>`
- **Literal result element is an element or text not in the XSL namespace**
`<html><body><p>This element belongs to XHTML namespace</p></body></html>`
- **Node tree is XML document's internal presentation in an XML parser**

XSLT = XSL and Transformations

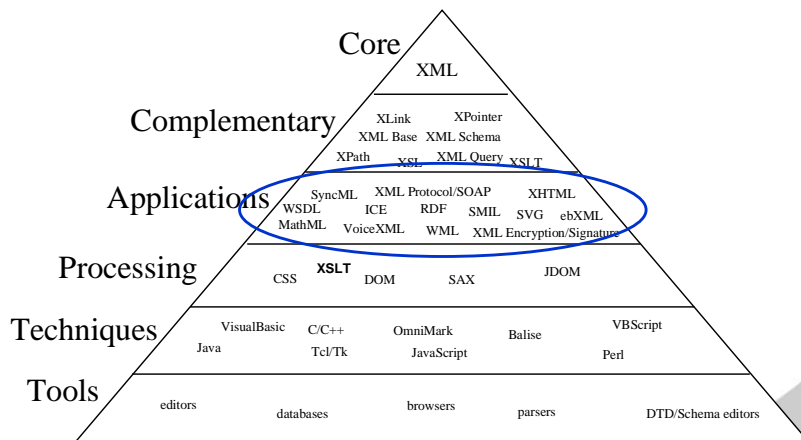
- **eXtensible Stylesheet Language: Transformations**
 - a part of style sheet language: XSL , [W3C recommendation v1.0 \(15.10.2001\)](#)
 - high-level declarative scripting language
 - W3C recommendation v1.0 (16.11.1999)
 - W3C working draft v1.1 (24.8.2001) ->[requirements for v2.0\(14.2.2001\)](#)
 - it has been designed to perform transformations from any XML source into target format

XSLT is a language for transforming the structure of an XML document for varying purposes like presenting data and exchanging data

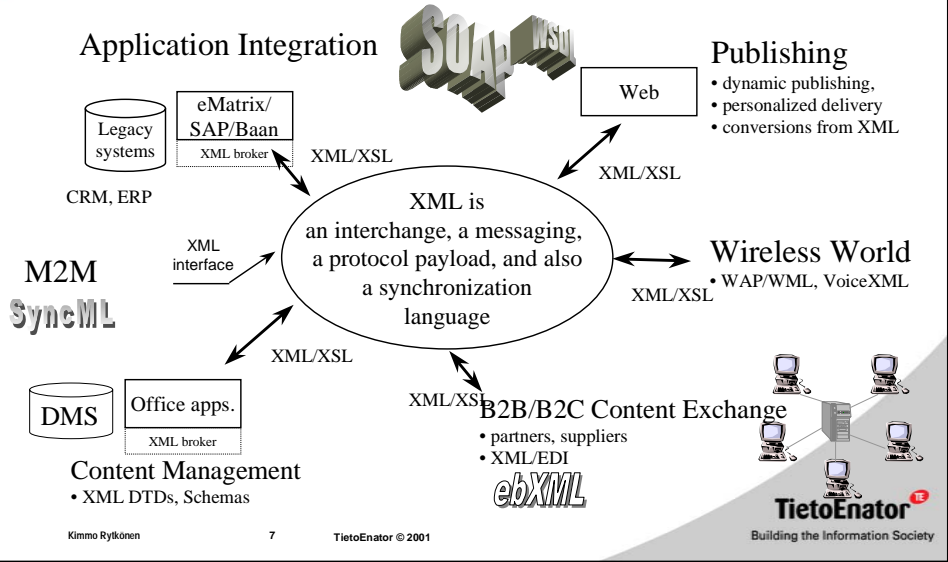
XML is a family of standards

- **The core standard: XML language for describing the syntax and the grammar for XML instances and document type definitions**
 - subset of ISO8879 : SGML - Standard Generalized Markup Language
- **The complementary standards:**
 - standard for styling: XSL language for describing the processing rules to compose XML documents
 - standard for transforming: XSLT language for producing conversion programs
 - standards for linking: XPointer, XLink, XPath for describing links and associating XML structures
 - standard for modeling: XML Schema for describing data models of XML documents
- **The application standards:**
 - use XML core standard and complementary standards

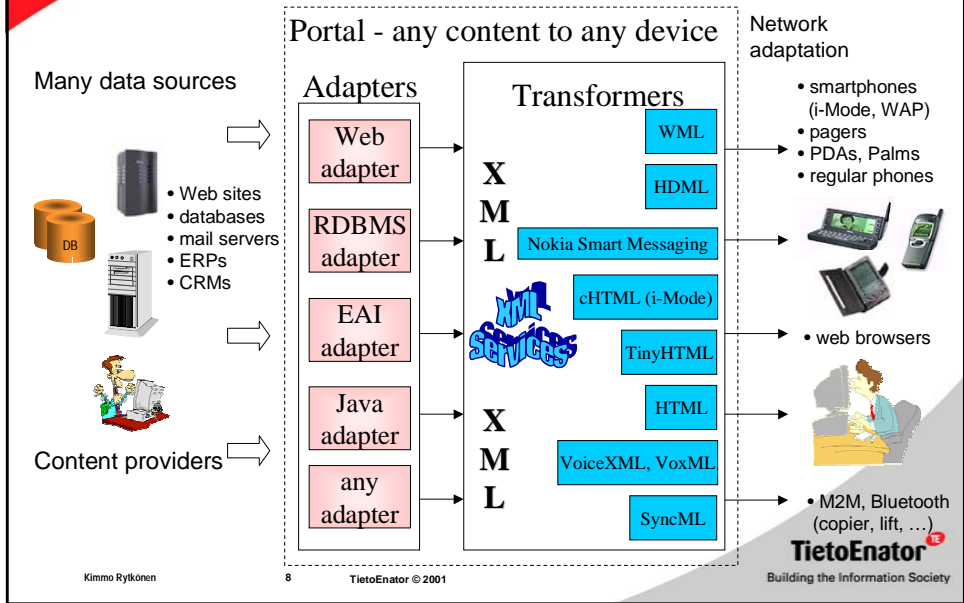
XML is a system independent, and a vendor independent language



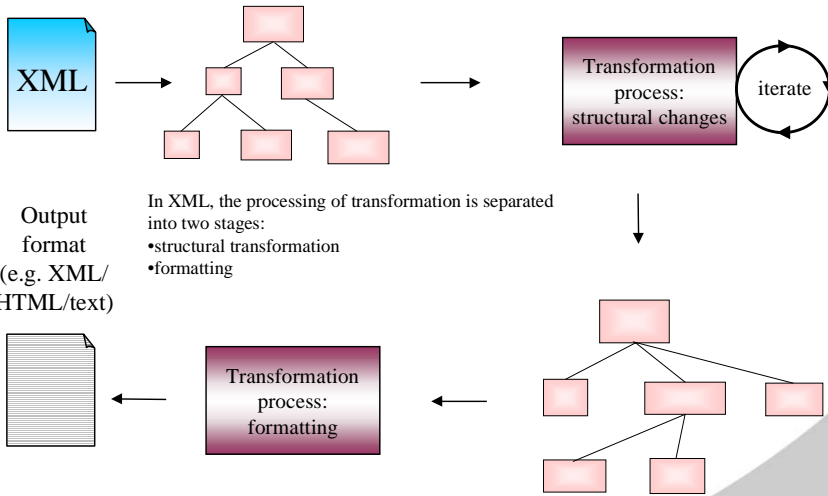
XML as a universal language



Publishing/delivering content needs transformations

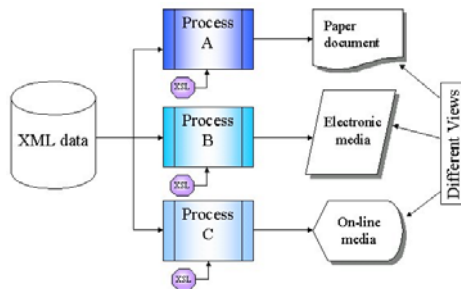


Implementing XML transformations



When to apply XSL Transformations?

- **Same information is used by many applications and consumed by different ways**
 - different target media with multiple views
 - data should be processed independently from applications specific constraints
 - HTML is not always the practical solution



Characteristics of XSL Transformations

- **Document model and vocabulary independent**
 - an XSLT stylesheet is independent of any DTD or schema that may have been used to constrain the instance being processed
 - an XSLT processor can process well-formed XML documents without a model
 - behavior is specified against the presence of markup in an instance as the implicit model, not against the allowed markup prescribed by any explicit model
 - one stylesheet can process instances of different document models
 - multiple instances following different document models can be used in a single transformation
 - different stylesheets can process a given single instance to produce different result

Characteristics of XSL Transformations

- **Validation unnecessary (but convenient)**
 - an XSLT processor need not implement a validating XML processor (do not look at DTD)
 - must implement at least a non-validating XML processor to ensure well-formedness
 - validation is convenient when debugging stylesheet development

XSL Transformations

- **XSLT program**

- is an XSL stylesheet that includes formatting instructions as well as transformation instructions
 - style sheet language:
 - formatting instructions (tags) from flow object name space
 - e.g. <fo:block>
 - transformation language:
 - transformation instructions (tags) from XSL name space
 - e.g. <xsl:for-each>
 - output productions:
 - literal result elements
 - e.g. <html>

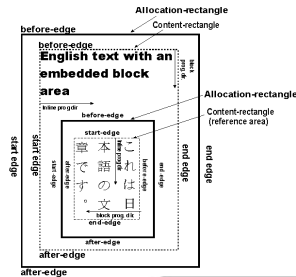
XSL - style sheet language

- **A catalogue of formatting objects**

- Addresses basic word-processing-level pagination.
- Semantic model for formatting (XML syntax).
- Managed and interpreted by the formatting process.
- 90 % of CSS properties.
 - new properties for pagination and layout

- **Formatting model**

- Rectangular areas of content.
- Spaces between content areas.
- 51 objects defined for pagination, layout, blocks, inline, table, list, links.
- 246 different properties.

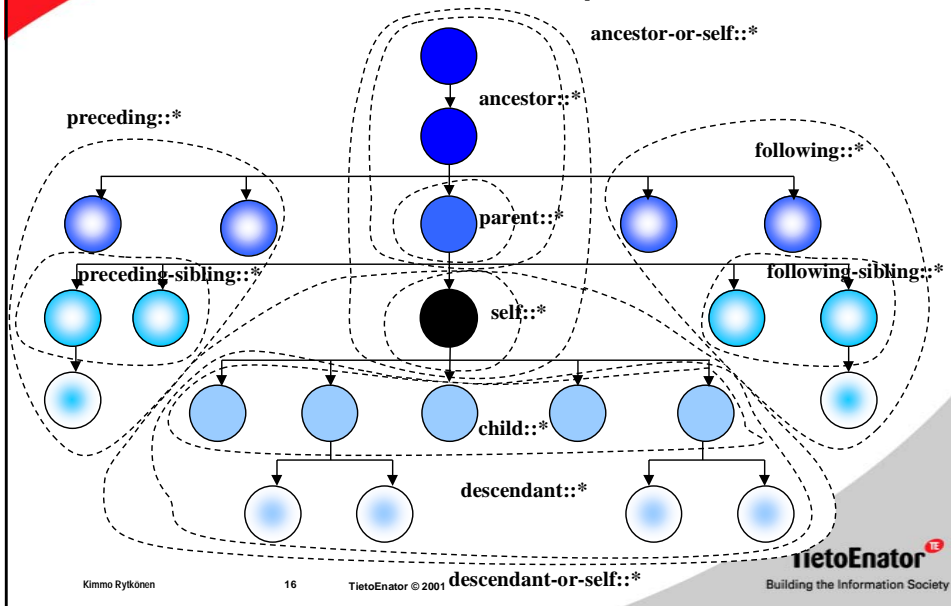


XSL - transformation language

- **XSLT program**

- can be written with any ASCII-based editor or XSL stylesheet editor
- is processed by an XSLT processor that relies on an XML parser
 - parser converts XML document into a tree structure (not always DOM compliant node tree)
- operates with an XML tree
 - uses XPath language to navigate in an XML node tree
 - XPath is W3C Recommendation and does not use the XML syntax
 - navigates around a node tree
 - select specific nodes and applies transformations rules on selected nodes
 - output result as a node tree, where
 - element nodes = represents XML elements
 - attribute nodes = attributes of XML elements
 - text nodes = represents XML element's content

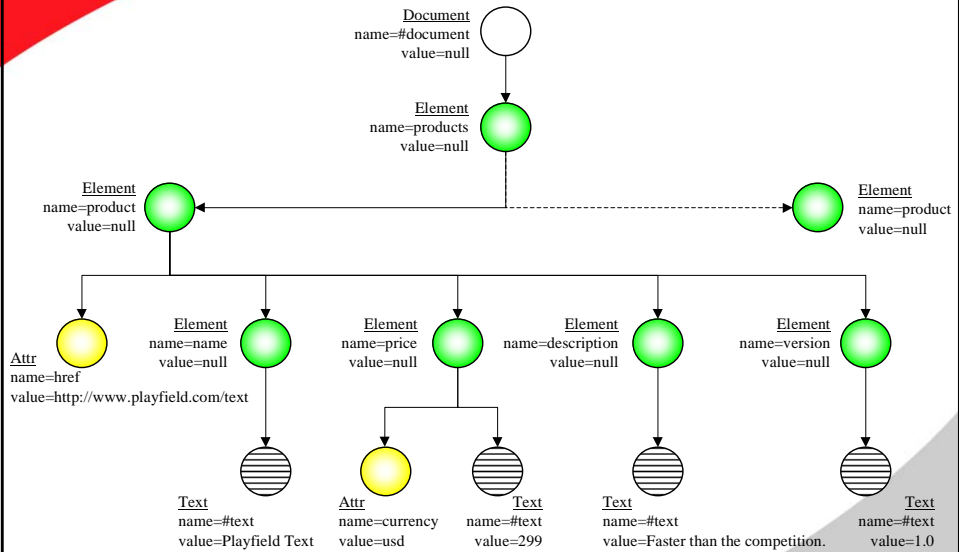
Areas of XML node tree are expressed with XPath



Example XML document

```
<?xml version="1.0"?>  
<?xml-stylesheet type="text/xsl" href="products.xsl"?>  
<products>  
  <product href="http://www.playfield.com/text">  
    <name>Playfield Text</name>  
    <price currency="USD">299</price>  
    <description>Faster than the competition.</description>  
    <version>1.0</version>  
  </product>  
  <product href="http://www.playfield.com/virus">  
    <name>Playfield Virus</name>  
    <price currency="EUR">199</price>  
    <description>Protect yourself against malicious code.</description>  
    <version>5.0</version>  
  </product>  
</products>
```

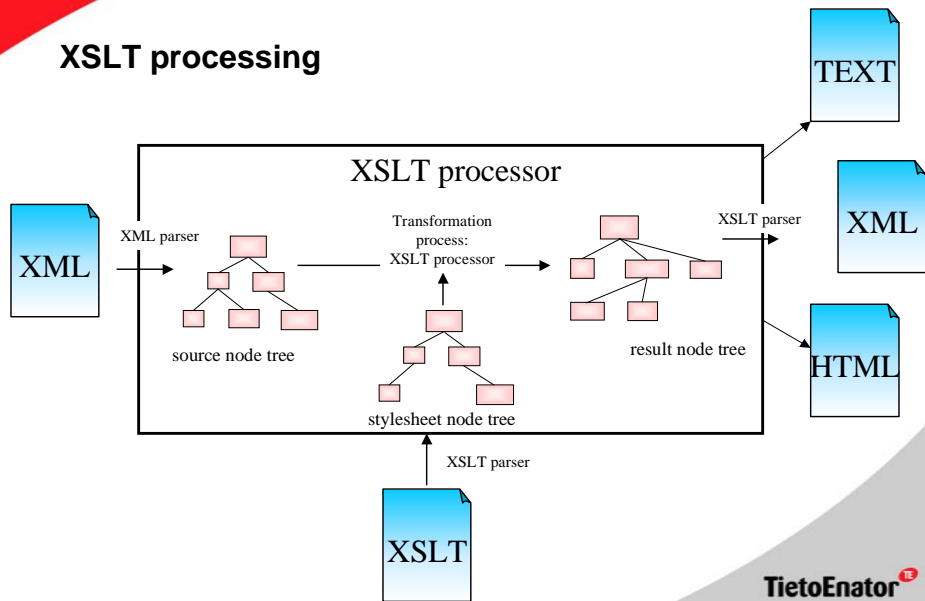
XML node tree



XPath expressions on the XML node tree (abbreviated syntax)

- **product** matches any product element
- ***** matches any element
- **name | price** matches any name element and any price element
- **products/product** matches any product element with a products parent
- **products//name** matches any name element with a products ancestor element
- **/** matches the root node
- **text()** matches any text node
- **processing-instruction()** matches any processing instruction
- **node()** matches any node other than an attribute node and the root node
- **id("11")** matches the element with unique ID value 11
- **product[1]** matches any product element that is the first product child element of its parent
- ***[position()=1 and self::product]** matches any product element that is the first child element of its parent
- **product[last()=1]** matches any product element that is the only product child element of its parent
- **products/product[position()>1]** matches any product element that has a products parent and that is not the first product child of its parent
- **product[position() mod 2 = 1]** would be true for any product element that is an odd-numbered product child of its parent.
- **product[@href="http://www.playfield.com/text"]/price** matches any price element with a product ancestor element that has a href attribute with value <http://www.playfield.com/text>
- **@currency** matches any currency attribute (not any element that has a currency attribute)
- **@*** matches any attribute

XSLT processing



Apache's Xalan parser - a Java XSLT processor

```

...
import org.xml.sax.SAXException;
import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessor;

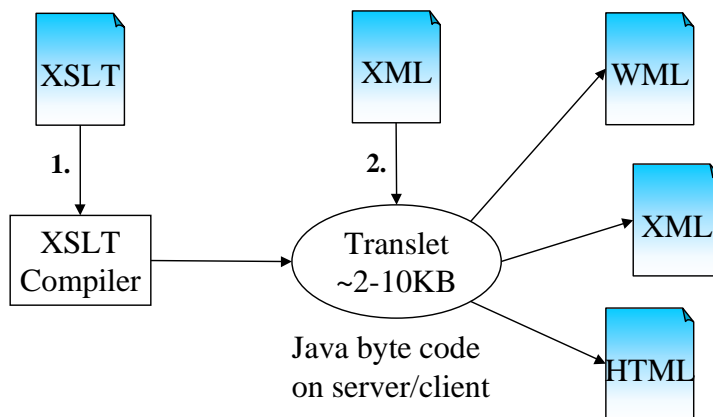
public class XSLTransform {
    public XSLTransform () {

    }

    // Processes XML file using XSL file
    public void XSLProcess ( String XMLInputFile, String XSLFile, String XMLOutputFile ) {
        try {
            // Have the XSLTProcessorFactory obtain a interface to a new XSLTProcessor object.
            XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
            // Have the XSLTProcessor processor object transform "XMLInputFile" to "XMLOutputFile", using the
            // XSLT instructions found in "XSLFile".
            processor.process( new XSLTInputSource( XMLInputFile ), new XSLTInputSource( XSLFile ), new
                XSLTResultTarget( XMLOutputFile ) );
        } ...
    }
}

```

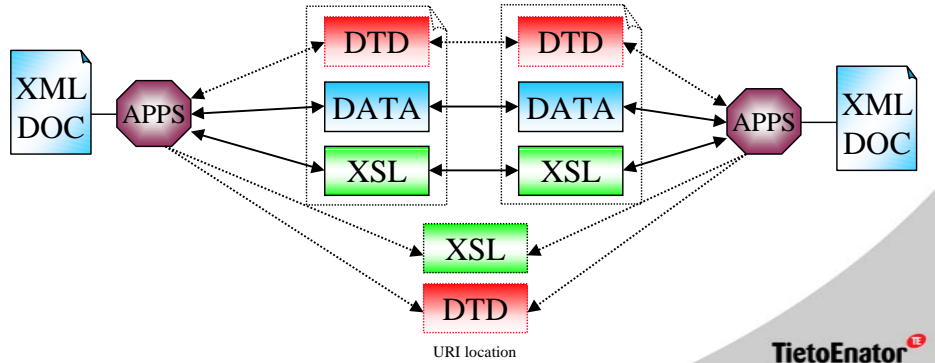
XSLT processing



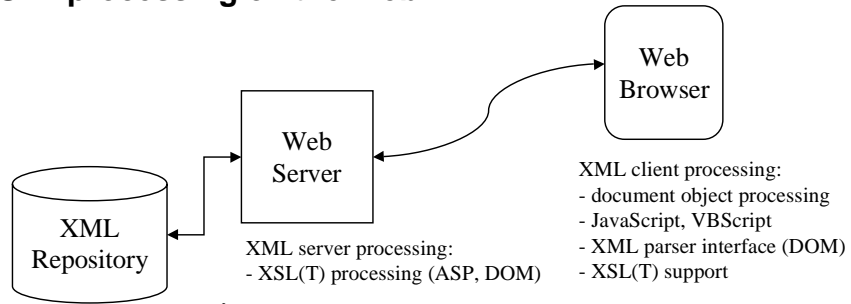
Exchanging XML documents

- **Associating XSL style sheets**

- <?xml-stylesheet href="myStyles.xml" type="text/xml"?>
- <?xml-stylesheet href="#myStyles" type="text/xml"?>



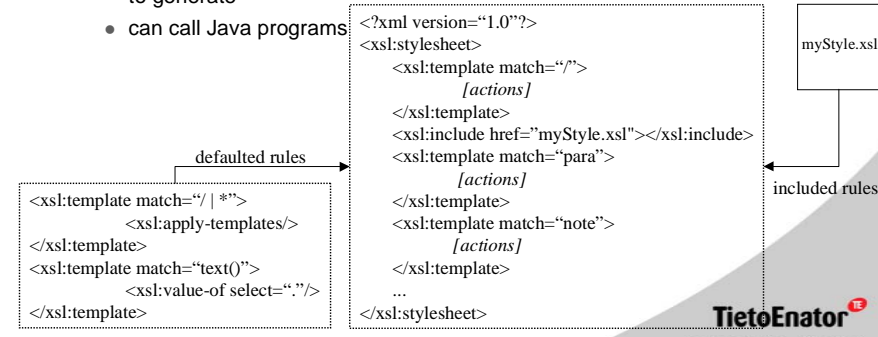
XSLT processing on the Web



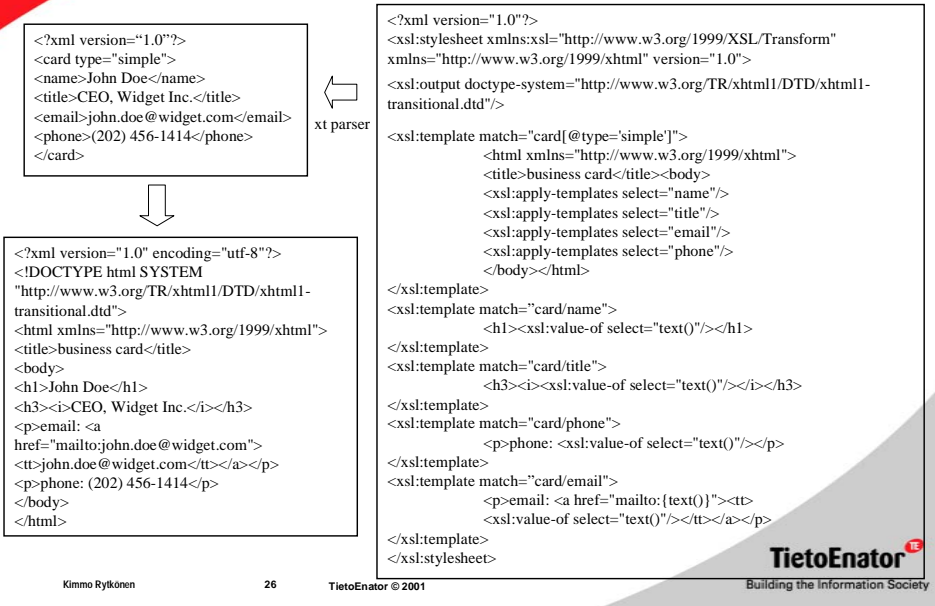
```
<%
'Load the XML
set source = Server.CreateObject("Microsoft.XMLDOM")
source.async = false
source.load(Server.MapPath("simple.xml"))
'Load the XSL
set style = Server.CreateObject("Microsoft.XMLDOM")
style.async = false
style.load(Server.MapPath("simple.xsl"))
Response.Write(source.transformNode(style))
%>
```

Model of XSLT program

- **XSLT is quite straight-forward and easy-to-learn scripting language**
 - consists of template rules (unordered, modularized)
 - defines how particular element type is being processed and what output to generate
 - can call Java programs:



Example XSLT program



XSLT software

● XSLT processors

- XT, SAX-compliant parser
 - Java application, Windows 32-bit executable includes XP Java XML parser with SAX support
- Saxon, SAX-compliant parser
 - Java application, Windows 32-bit executable
- MSXML3
 - works with IE 5
 - download latest version from <http://www.microsoft.com/xml>
- Xalan stylesheet processor (<http://xml.apache.org/>) (Java, C++)
- XML Parser for Java v2 (<http://technet.oracle.com/tech/xml/>)
- XSLT Compiler (<http://www.sun.com/xml/>)
- 4XSLT, XSLT implementation written in Python
- Java APIs for XML Processing (JAXP)
 - (http://java.sun.com/xml/xml_jaxp.html)
- check also other implementations at <http://www.w3.org/Style/XSL/>

XSLT software

● XSLT editors

- XML Spy, XML/DTD/XSL/Schema editor and parser
 - <http://www.xmlspy.com/>
- Excelon Stylus, XSL editor
 - <http://www.exceloncorp.com/>

Other choices to implement XML Transformations

- **Programming languages like Java, C, and C++ and scripting languages like Perl and JavaScript**
 - there are two standard API for processing XML data:
 - DOM for tree manipulation
 - SAX for event-based processing
 - standard includes interface definitions for Java and JavaScript
 - software implementations exist
- **Programming languages like OmniMark and Balise**
 - developed for SGML and XML data processing
 - up-translation, inter-translation, down-translation
 - have strong pattern-matching feature
 - not based on any standard API

Some XSLT resources

- **<http://www.wrox.com/>**
 - book: "XSLT Programmer's Reference", Michael Kay
 - downloadable XSLT source code
- **<http://www.w3.org/Style/XSL/>**
 - W3C specifications
- **XSLT sites**
 - <http://www.xslinfo.com/>
 - <http://www.oasis-open.org/cover/xsl.html>
 - <http://www.xml.com>
 - <http://msdn.microsoft.com/xml/default.asp>
 - <http://www.xfront.com>

XSLT language with examples

Example #1: hello.xsl

```

<?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>
  <head>
  <title>Today's greeting</title>
  </head>
  <body>
  <p><b><xsl:value-of select="greeting"/></b></p>
  </body></html>
</xsl:template>
</xsl:stylesheet>

```

} standard XML heading

} standard XSLT heading as in the W3C XSLT specification

} template element defines a template rule

} Literal result elements output HTML

Template body defines what output to generate.
value-of instruction with **select** attribute:
 "finds the set of all greeting elements that are children of the root node that this template rule is processing"
 "selects the content (text nodes) of node specified by select attribute"



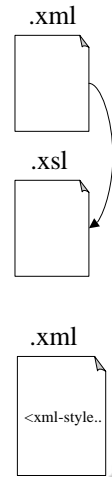
XSL stylesheet and XML document

- **External stylesheet**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="style.xml"?>
<root-element>...
</root-element>
```

- **Embedded stylesheet**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="#style1"?>
<!DOCTYPE root-element SYSTEM "doc.dtd">
<!-- doc.dtd refers to xsl dtd. -->
<root-element>...
<xsl:stylesheet id="style1" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
...
</root-element>
```



XSL stylesheet - structure

- **XSL stylesheet consists of**

- top-level elements
 - also user-defined top-level elements (check XSLT standard)
- template rules
 - defines how particular element type is being processed and what output to generate
 - a sequence of elements and text nodes
 - elements are either literal result elements or XSL instruction elements or extension elements (check XSLT parser implementation)
 - the order of template rules is insignificant

- **Stylesheet elements contains attributes**

XSL stylesheet - top-level elements

- **Some top-level elements**
 - **xsl:stylesheet** tai **xsl:transform**: the outermost element
 - **xsl:template**: contains element type specific processing instructions
 - **xsl:include**: does a textual inclusion of the referenced stylesheets
 - **xsl:import**: does the same as xsl:include, but with lower import precedence of imported definitions
 - **xsl:variable**: defines a global or a local variable that can be referenced in its scope
 - **xsl:output**: indicates in what format the output should be
 - **xsl:strip-space** & **xsl:preserve-space**: indicates that whitespace nodes in the source document are ignored or not

Example #2: top-level.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="imports.xsl"></xsl:import>
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
    indent="yes"/>
  <xsl:strip-space elements="*" />
  <xsl:include href="greetings.xsl"></xsl:include>
</xsl:stylesheet>
```

import instruction: imports definitions with lower priority
output instruction: controls output of result tree (serialization)
include instruction: imports definitions with the same priority
strip-space instruction: controls whether spaces are dropped out or not



XSL stylesheet - template rules

- **<xsl:template> element and attributes**
 - **match**: defines the matching element type
 - **name**: defines the template name
 - **mode**: groups template rules
 - used to select a certain set of template rules
 - there is a need to process same element types differently in the source tree
 - **priority**: if there are more than one matching template rule, the priority value (either system-allocated priority value or user-allocated priority value) determines the used template rule (higher number = higher priority)

```
<xsl:template match="title" priority="-1">
```

XSL stylesheet - template rules

- **Example #3:**
 - template rules
 - pattern matching (element selection using XPath)
 - conflict resolution (priority)
 - modes
 - applying template rules
 - calling template rules

Example #3: template.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="books">
    <html><body><h1>A list of books</h1><table width="440"><xsl:apply-templates/></table></body></html>
  </xsl:template>
  <xsl:template match="book[@category='reference']">
    <xsl:call-template name="reference"/>
  </xsl:template>
  <xsl:template match="book[@category='fiction']">
    <tr><xsl:apply-templates mode="fiction"/></tr>
  </xsl:template>
  <xsl:template name="reference">
    <tr><xsl:apply-templates/></tr>
  </xsl:template>
  <xsl:template match="book/author">
    <td><b><xsl:value-of select="."/></b></td>
  </xsl:template>
  <xsl:template match="title" priority="1">
    <td><tt><xsl:value-of select="."/></tt></td>
  </xsl:template>
  <xsl:template match="title" priority="-1">
    <td><u><xsl:value-of select="."/></u></td>
  </xsl:template>
  <xsl:template match="price">
    <td><u><xsl:value-of select="."/></u></td>
  </xsl:template>
  <xsl:template match="author | title | price" mode="fiction">
    <td><i><xsl:value-of select="."/></i></td>
  </xsl:template>
</xsl:stylesheet>
Kimmo Rytkönen
```

39

TietoEnator © 2001

match attribute uses XPath to locate constructs from the XML node tree
call-template instruction: calls a named template rule
priority and **mode** attributes determine the precedence of the template rule



TietoEnator^{TE}
 Building the Information Society

XSL stylesheet - instruction elements

- They appear in template rules, and may have attributes and subelements
 - generating content (elements, attributes, text)
 - copying content
 - numbering
 - repetition
 - conditional processing
 - sorting
 - variables, parameters
- They also use functions

XSL stylesheet - instruction elements

Elements define template rules:

<xsl:apply-imports>
 <xsl:apply-templates>
 <xsl:call-template>

Elements copy information:

<xsl:copy>
 <xsl:copy-of>

Elements for conditional processing:

<xsl:choose>
 <xsl:if>
 <xsl:for-each>
 <xsl:when>
 <xsl:otherwise>

Elements define variables and parameters:

<xsl:variable>
 <xsl:param>
 <xsl:with-param>

Elements generate content:

<xsl:value-of>
 <xsl:attribute>
 <xsl:element>
 <xsl:comment>
 <xsl:processing-instruction>
 <xsl:text>
 <xsl:message>

Elements for sorting and numbering:

<xsl:number>
 <xsl:sort>

Example #4: instruction1.xsl



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <xsl:element name="html">
      <xsl:element name="body">
        <xsl:element name="h1">A list of books</xsl:element>
        <xsl:element name="table">
          <xsl:attribute name="width">440</xsl:attribute>
          <xsl:call-template name="table-header"/>
          <xsl:comment>This is the body of the table</xsl:comment>
          <xsl:for-each select="//book">
            <xsl:call-template name="book"/>
          </xsl:for-each>
        </xsl:element>
      </xsl:element>
      <xsl:message>All books processed.</xsl:message>
    </xsl:template>
    <xsl:template name="table-header">
      <xsl:comment>This is the header of the table</xsl:comment><th>
        <td><b><xsl:text>Author</xsl:text></b></td><td><b><xsl:text>Title</xsl:text></b></td>
        <td><b><xsl:text>Price</xsl:text></b></td></th>
      </xsl:template>
    <xsl:template name="book">
      <tr><td></td><td><xsl:value-of select="author"/></td><td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="price"/></td></tr>
    </xsl:template>
  </xsl:stylesheet>
```

Element, attribute, comment, message, text instructions:
 generates appropriate content

Example #5: book.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="books">
    <html><body><h1>A list of books</h1><table width="440">
      <xsl:for-each select="book"><tr>
        <xsl:if test="@category='reference'"><xsl:apply-templates mode="reference"/></xsl:if>
        <xsl:if test="@category='fiction'"><xsl:apply-templates mode="fiction"/></xsl:if></tr>
      </xsl:for-each></table></body></html>
    </xsl:template>
    <xsl:template match="author | title | price" mode="reference">
      <xsl:variable name="element" select="name()"/><td>
        <xsl:choose>
          <xsl:when test="$element='author'"><b><xsl:value-of select="."/></b></xsl:when>
          <xsl:when test="$element='title'"><i><xsl:value-of select="."/></i></xsl:when>
          <xsl:otherwise><u><xsl:value-of select="."/></u></xsl:otherwise>
        </xsl:choose></td>
      </xsl:template>
    <xsl:template match="author | title | price" mode="fiction">
      <xsl:variable name="element" select="name()"/><td>
        <xsl:choose>
          <xsl:when test="$element='author'"><i><xsl:value-of select="."/></i></xsl:when>
          <xsl:when test="$element='title'"><u><xsl:value-of select="."/></u></xsl:when>
          <xsl:otherwise><b><xsl:value-of select="."/></b></xsl:otherwise>
        </xsl:choose></td>
      </xsl:template>
    </xsl:stylesheet>
```



Kimmo Rytkönen

for-each, if, choose, when, otherwise instructions:
elements for conditional processing

43

TietoEnator © 2001

TietoEnator^{TE}
Building the Information Society

Example #6: poem.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body><p><xsl:apply-templates select="/poem/stanza"/></p></body></html>
  </xsl:template>
  <xsl:template match="stanza">
    <p><table><xsl:apply-templates/></table></p>
  </xsl:template>
  <xsl:template match="line">
    <tr><td width="240"><xsl:value-of select="."/></td><td width="200">
      <xsl:variable name="line-nr">
        <xsl:number level="any" from="poem" format="i"/>
      </xsl:variable>
      <xsl:value-of select="$line-nr"/><xsl:text> (</xsl:text>
      <xsl:if test="position() = 1"><xsl:text>The first line of stanza</xsl:text></xsl:if>
      <xsl:if test="position() = last()"><xsl:text>The last line of stanza</xsl:text></xsl:if>
      <xsl:if test="position() != 1 and position() != last()"><xsl:text>The middle line of
    stanza</xsl:text></xsl:if>
      <xsl:text></xsl:text>
    </td></tr>
  </xsl:template>
</xsl:stylesheet>
```

number instruction: element for numbering



Kimmo Rytkönen

44

TietoEnator © 2001

TietoEnator^{TE}
Building the Information Society



Example #7: product.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <h1>A list of products</h1>
      <table>
        <tr><td>Product name</td><td>Total sales</td></tr>
      <tbody>
        <xsl:for-each select="//product">
          <xsl:sort select="sum(region/@sales)" data-type="number" order="ascending"/>
          <xsl:variable name="sales" select="sum(region/@sales)"/>
          <tr><td><xsl:value-of select="@name"/></td>
            <td><xsl:value-of select="format-number($sales, '###0.00')"/></td></tr>
        </xsl:for-each>
      </tbody>
    </table>
  </body></html>
</xsl:template>
</xsl:stylesheet>
```

sort instruction: element for sorting

XPath and XSLT functions

Functions for data type conversions:
 boolean()
 format-number()
 number()
 string()

Aggregation:
 count()
 sum()

Boolean functions:
 false()
 true()
 not()

Arithmetic functions:
 ceiling()
 floor()
 round()

Functions for getting node names and identifiers:
 generate-id()
 lang()
 local-name()
 name()
 namespace-uri()
 unparsed-entity-uri()

Functions that find nodes:
 document()
 key()
 id()

Functions for string manipulation:
 concat()
 contains()
 normalize-space()
 starts-with()
 string-length()
 substring()
 substring-before()
 substring-after()
 translate()

Functions that return information about the context:
 current()
 last()
 position()

Functions that provide information about the processor:
 element-available()
 function-available()
 system-property()

Other XSLT examples

- **Example #8**
 - transforming book.xml into HTML
 - <xsl:choose>, <xsl:number>
- **Example #9**
 - transforming proc.xml into HTML
 - <xsl:apply-templates>,
<xsl:value-of>
- **Example #10**
 - transform hockey.xml into HTML
 - <xsl:sort>, <apply-templates>,
<xsl:for-each>
- **Example #11**
 - sort content of sort.xml
 - <xsl:sort>