ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

Ενότητα 6: XML, XSLT and XPATH - 1

Χρήστος Νικολάου
Τμήμα Επιστήμης Υπολογιστών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

XML
XSLT and XPATH
605.444 / 635.444

David Silberberg

Lecture 12

Transformation of Data

- XML is a generic data representation language
  - Not concerned, per se, about presentation
  - Not concerned about tailoring representation to any particular application
  - It represents the data from a particular perspective
  - Data can be represented in a vendor-neutral way
    - Easy to parse (will see later)
    - Portable
    - Can be used in a Java environment (parsers, transformers, etc.)
  - But, how do you use XML in an application?
  - Furthermore, how do we use XML across multiple applications???

Transformation Tools

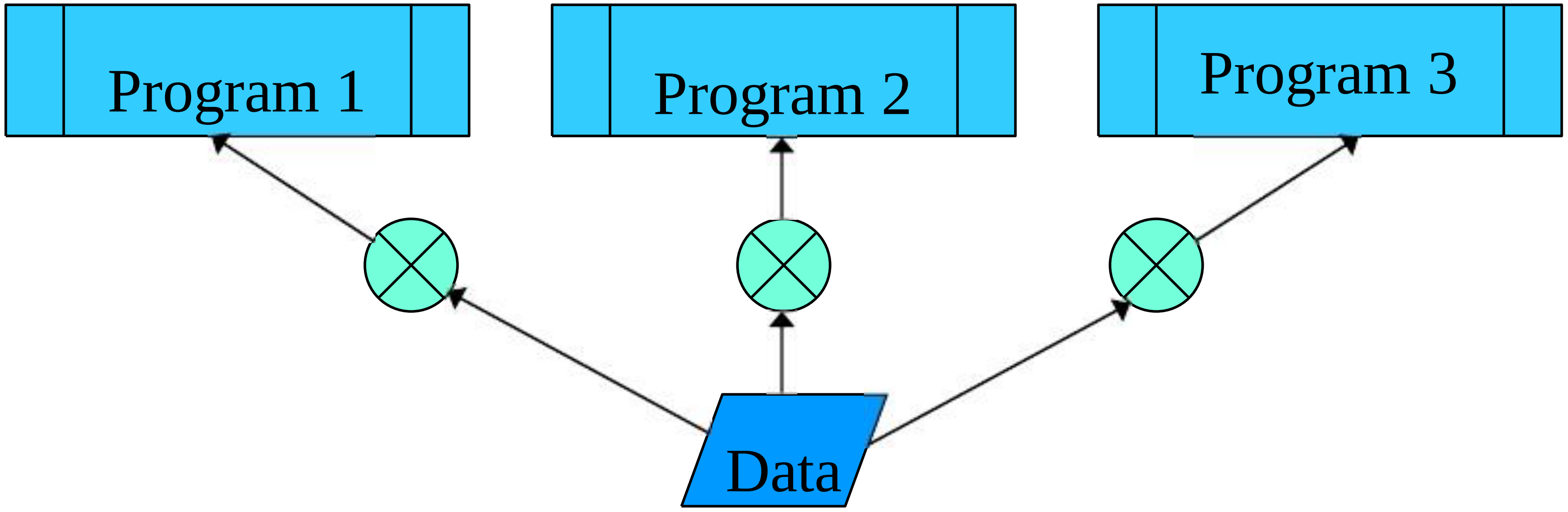
- There are parsers (SAX and DOM)
  - Can be used in conjunction with Java to parse XML
  - Programming language approach to transformation
  - Programs are very general and powerful
  - Programs are hard to maintain
  - Programs and their subsets cannot easily be reused
- We would like a simple declarative language for expressing transformations
  - Easy to use and maintain
  - Easy to share
  - Powerful transformation constructs

Why Transform?

- XML represents a pure data layer
- Different applications need similar data
  - Data conceptually the same
  - Each application needs a different structure to operate on
- Different data architecture models
  - Central abstract data repository
    - Applications retrieve data and reformat as necessary
    - Often requires each application to write transforms
    - Applications pass data through data representation

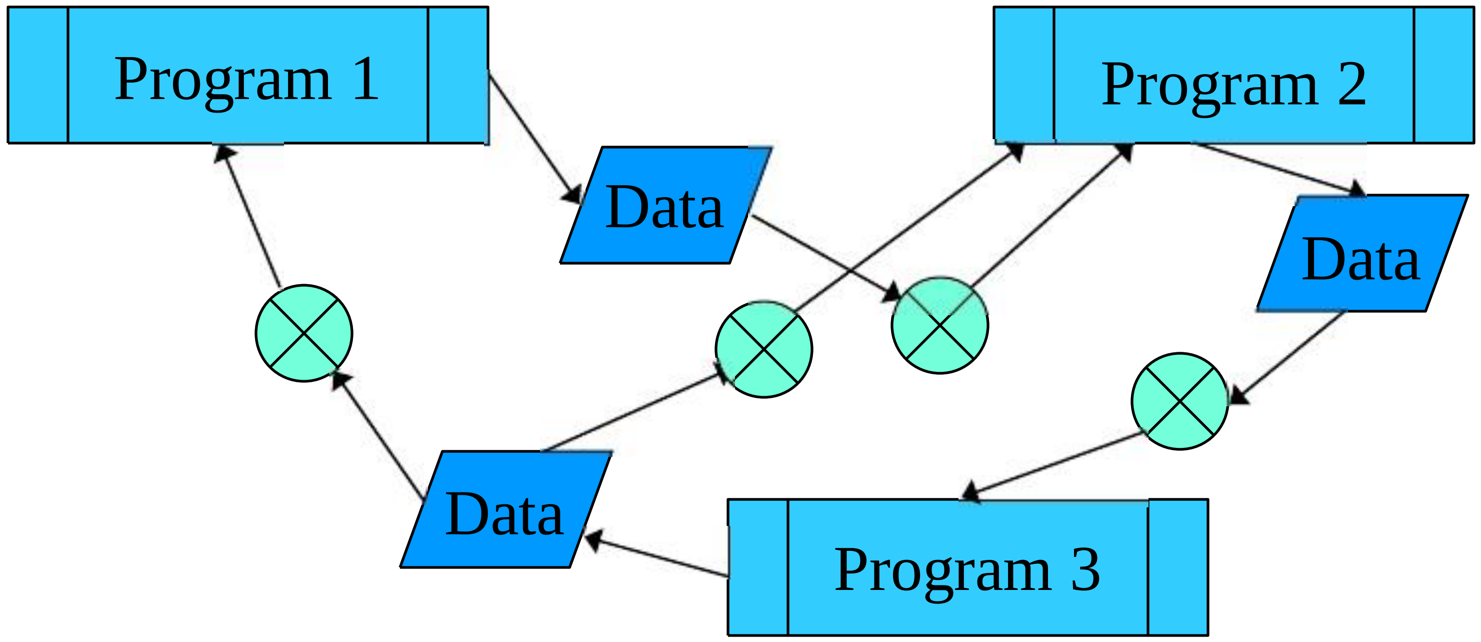
Transformation Architectures

- Different data architecture models
  - Central abstract data repository



Transformation Architectures (cont.)

– Enterprise data exchange architecture
  - Applications pass data and reformat as necessary
  - Often requires each application to write multiple transforms
  - There may not be a central repository



Data Transformation Architectures

- Any system can be composed of any combination of these constructs
- Data may undergo several transformations as they move from application to application
- Each transformation alters the structure of the data
- Data produced by a transformation is in a form usable by their corresponding program(s)
- Data may also need to be transformed for display
  - Pure HTML
  - Other display languages

General Data Structures

- Most often, program designers create data structures to satisfy their system needs
- Usually, the design is based on a narrow focus, which is usually geared toward their specific application
- System designers do not often know how others might use the data
- In the Web environment, it is even more difficult to know or anticipate the uses of the data by other systems
- Thus, there is a need to transform data from one representation to another in a general and powerful way

Example - Two Applications

- Application 1 - Sales Record
- The Animal World TM store takes orders for pet equipment
- The store is concerned with recording the activities of the sales people.
  - To calculate commissions
  - To know who is performing well
- Animal World TM also needs to keep a record of its customers so that it can “spam” them with sales circulars once a week.
- Animal World TM also needs to generate orders to their suppliers.

Application 1 - sales.xml

```
<?xml version="1.0"?>
<transaction>
    <salesman>
        <lastname>Smith</lastname>
        <firstname>Fred</firstname>
        <mi>P</mi>
    </salesman>
    <customer>
        <name>Frank Thomas</name>
        <address>10 Maple Street</address>
        <city>Columbia</city>
        <state>MD</state>
        <zip>22222</zip>
    </customer>
```

Application 1 - sales.xml (cont.)

```
        <date>
                <year>2005</year>
                <month>12</month>
                <day>22</day>
        </date>
        <item>
                <name>Cat Chow</name>
                <size>30</size>
                <qty>2</qty>
                <unitprice>9.95</unitprice>
        </item>
</transaction>
```

Application 2

- Application 2 - Supplier
- The Chow Train TM company fills orders from pet supply stores
- Chow Train TM sends the orders directly to the purchasers
- Chow Train TM bills the purchasers
- Chow Train TM sends the store a percentage of the price collected

Application 2 - supplier.xml

```
<?xml version=“1.0”?>
<order>
        <date>2005-12-22</date>
        <company>Animal World</company>
        <item>
                <code>CC-10456</code>
                <description>Cat Chow</description>
                <qty>2</qty>
        </item>
        <for>
                <name>Frank Thomas</name>
                <street>10 Maple Street</street>
                <city>Columbia, MD  22222</city>
        </for>
</order>
```

Extensible Stylesheet Language (XSL)

- XSL is a general language for
  - Transforming the structure of a document into another structure
  - Formatting the document for display
  - Specifications can be found at: http://www.w3c.org/TR/xslt
- Three standards are used
  - Extensible Stylesheet Language for Transformation (XSLT)
    - Language for transforming XML structures
  - XSL formatting objects (XSL-FO)
    - Browser-neutral publishing language for formatted outputs
      - Sets headers, footers, margins, etc.
      - Enables embedded graphics, etc.
      - Deals with non-Western writing – bidirectional lines, top-to-bottom, etc.
    - Not covered in this course – we will use CSS to format the output of XSLT
  - XML Path Language (XPath)
    - Language for describing node elements in an XML hierarchy

Where to get an XSLT Tool

- Different tools available – I have not tested all of them!
- The class will use the oXygen tool for XSLT tranformation
- Sun's version has been donated to apache.org
  - http://xml.apache.org/xalan-j/
  - Read instructions for operation
- James Clark's version for Windows
  - http://www.jclrk.com/xml/xt.html
  - Need Microsoft's Java VM (usually comes with IE)
  - *xt source stylesheet result*
- MSXML
  - http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp

XSL is in XML Format

- XSL and its subset XSLT are XML languages
- Therefore, XSLT stylesheets must obey the formatting rules of XML.
- XSLT uses the *xsl:* namespace for all of its commands
  - Thus, an XSLT parser/processor can look at the tags of an XML document and determine which it must process.
  - The namespace is located at the W3C site.
  - http://www.w3.org/1999/XSL/Transform
- Other namespaces may be defined in the XSLT document, as well.
- The XSLT <u>stylesheet</u> declaration must come after the initial XML declaration <?xml … ?>.

Example Stylesheet Declaration

```
<?xml version=“1.0” ?>

<xsl:stylesheet
    xmlns:xsl=“http://www.w3.org/1999/XSL/Transform”
    xmlns:abc=“some other namespace”
    version=“1.0”>

<!-- transformation rules -->

</xsl:stylesheet>
```

XSLT Transformations

- XSLT built upon templates
- Tells the transformation engine
  - What to look for
  - What to output based on the input
- Each template matches (usually) one specific thing in the XML document and transforms it

```
<xsl:template match="[Xpath expression]">
    <!-- Format rules -->
</xsl:template>
```

XSLT Template

- Begin and end tags are
  - <xsl:stylesheet ...>
  - </xsl:stylesheet>
- Match tells the transformation tool what to match
- XPath is a specification language to identify places in an XML document.
- Format rules specify the output and its format.
- Typical stylesheets have multiple templates.
- If multiple templates match an XML tag, there are implicit rules about which template will be chosen.

Simple Match

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">

    <xsl:template match="transaction">A transaction was found.
    </xsl:template>

</xsl:stylesheet>
```

Output when applied to XML document 1:

A transaction was found.

What Just Happened

- XSLT processor found all tags labeled “transaction”.
- Then, it printed the string “**A transaction was found.**”
- The transformation rule can be applied 0 or many times to the document.
- If there were two transaction tags, the XSLT processor would have printed:

  A transaction was found.

  A transaction was found.
- Of course, there can only be one document root element.

XSLT Template and Output

- XSLT transformation

```
<xsl:template match=“transaction”>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match=“name”>A name tag was found.
</xsl:template>
```

- Output of XSLT processor

```
A name tag was found.
A name tag was found.
```

What Just Happened

- When a match is encountered, the entire hierarchy of that tag and below is read in.
- The next rule “xsl:apply-templates” tells the XML processor to apply the rest of the templates to the hierarchy matched and read in.
- If no rule was there:

<xsl:template match=“transaction”/>

or

<xsl:template match=“transaction” >

</xsl:template>

The entire tree is read in and the <u>data</u> is printed out.

Example of Hierarchy Match

- Applying this to sales.xml:

```
<xsl:template match="transaction" >
</xsl:template>
```

- Produces:

```
Smith
Fred
P

Frank Thomas
10 Maple Street
Columbia
MD
22222
etc.
```

Simple Web Page Transformation ...

- Applying this to sales.xml:

```
<xsl:template match="transaction" >
    <html>
      <head>
         <title>Transaction Web Page</title>
      </head>
      <body>
         <xsl:apply-templates/>
      </body>
    </html>
</xsl:template>
```

Becomes this Web Page

- Yields:

```
<html>
   <head>
      <title>Transaction Web Page</title>
   </head>
   <body>
          Smith
          Fred
          P

          Frank Thomas
          10 Maple Street
          Columbia
          etc.
   </body>
</html>
```

Selection of Items

- Let’s do something a bit more complicated
- Print good customers from a file called *customer.xml*

```
<?xml version=“1.0”?>
<customers>
        <good>
                <name>Jones, Fred</name>
                <name>Li, Sue</name>
                <name>Carnot, John</name>
        </good>
        <bad>
                <name>Tell, William</name>
                <name>Carr, Sam</name>
        </bad>
</customers>
```

Print All Customers

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:template match="customers">
        <html><head><title>Customer Web Page</title></head>
           <body>
              <xsl:apply-templates/>
           </body>
        </html>
    </xsl:template>

    <xsl:template match="name">
        <xsl:value-of select="."/>
    </xsl:template>

</xsl:stylesheet>
```

Print All Customers Output

```
<html><head><title>Customer Web Page</title></head>
  <body>
        Jones, Fred
        Li, Sue
        Carnot, John
        Tell, William
        Carr, Sam
  </body>
</html>
```

What Happened

- The outer template was matched by:

  ```
  <xsl:template match="customers">
  </xsl:template>
  ```

- The HTML text was printed when the match was encountered
- The command <xsl:apply-templates/> caused the <customer> element and its children to be scanned.
- The name tags were matched by

  ```
  <xsl:template match="name">
  </xsl:template >
  ```

- The name data was printed by

  ```
  <xsl:value-of select="."/>
  ```

Print Good Customers

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

        <xsl:template match="/">
            <html><head><title>Good Customer Web Page</title></head>
              <body>
                  <xsl:for-each select="/customers/good/name">
                      <p><xsl:value-of select="."/></p>
                  </xsl:for-each>
              </body>
            </html>
        </xsl:template>

</xsl:stylesheet>
```

Good Customer Output

```
<html><head><title>Good Customer Web Page</title></head>
    <body>
        <p>Jones, Fred</p>
        <p>Li, Sue</p>
        <p>Carnot, John</p>
    </body>
</html>
```

What Happened

- The root of the document was matched by:

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

- The HTML text was printed when the match was encountered
- The following command caused all the tags in the hierarchy */customer/good/name* to be found

```
<xsl:for-each select="/customers/good/name">
</xsl:for-each>
```

- The *customer/good/name* data was printed by

```
<xsl:value-of select="."/>
```

Sort the Bad Customers

```
<?xml version=“1.0” ?>
<xsl:stylesheet xmlns:xsl=“http://www.w3.org/1999/XSL/Transform” version=“1.0”>

        <xsl:template match=“/”>
            <html><head><title>Bad Customer Web Page</title></head>
              <body>
                <xsl:for-each select=“/customers/bad/name”>
                    <xsl:sort select=“.”/>
                    <xsl:copy-of select=“text()”/> <BR/>
                </xsl:for-each>
              </body>
            </html>
        </xsl:template>

</xsl:stylesheet>
```

Sorted Bad Customers

```
<html><head><title>Bad Customer Web Page</title></head>
        <body>
                Carr, Sam<BR/>Tell, William<BR/>
        </body>
</html>
```

XSLT Has No Side Effects

- In most programming languages
  - The order of operations is important
  - Operations leave side effects (e.g., variable values change)
- In XSLT
  - We are not concerned about the order of operations
  - XSLT processor looks for matches and applies applicable templates
  - The order of the template definitions do not matter
  - If multiple templates apply to a tag, there are implicit rules as to which template is selected.
  - However, this is not dependent on the order of template definitions
  - No variable values are set by the templates

Τέλος Ενότητας