

Enterprise Application Integration

2. XML Transformations

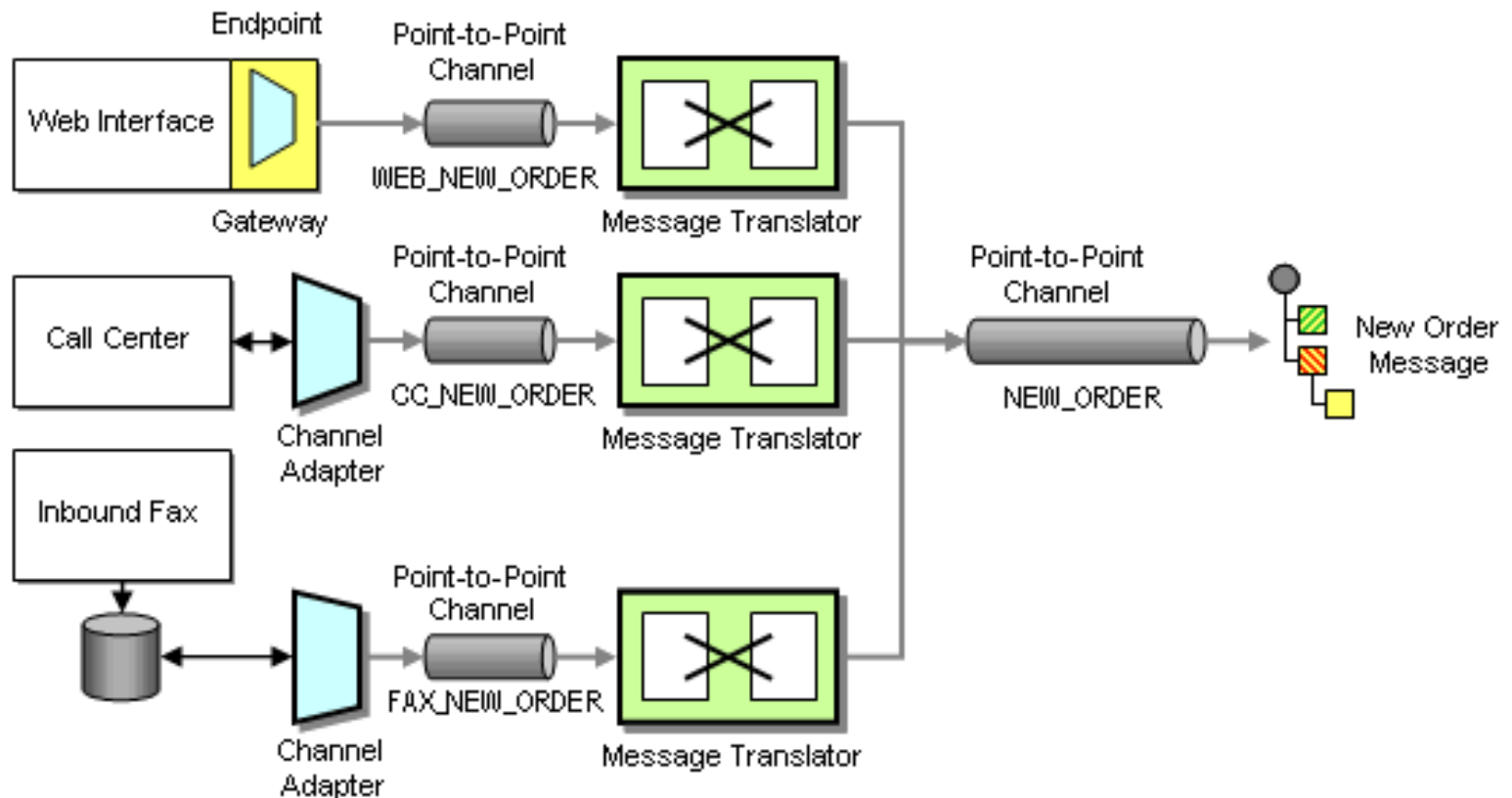
2.1. XPath and XSLT



Paulo Marques
Informatics Engineering Department
University of Coimbra
pmarques@dei.uc.pt

Motivation

- As it will be seen, a fundamental aspect for enterprise application integration is the **adaptation** and **transformation** of messages.

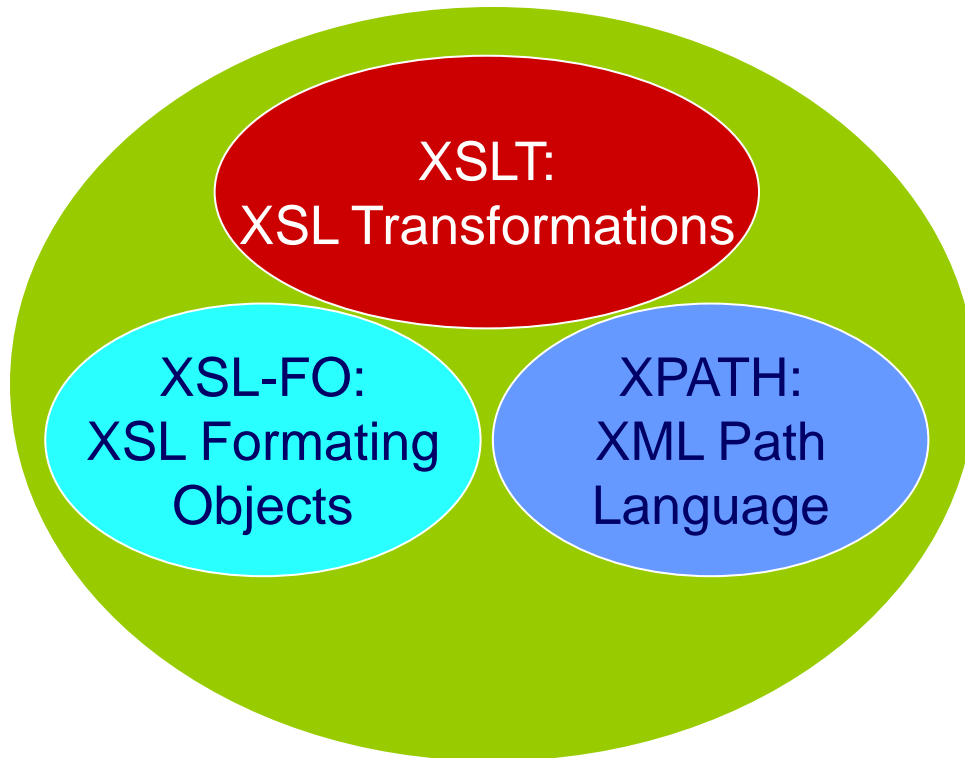


XPATH + XSLT

- XSLT (*XSL Transformations*) is a technology, among others, which allow to perform such transformation.
- It's typically used for:
 - Converting XML documents into other XML documents (i.e. converting between different XML schemas)
 - Generate user-readable documents (e.g. HTML, PDF)
- The XPATH technology allows to query for specific nodes in an XML documents

XML technologies for transformation

XSL – eXtensible Stylesheet Language



- **XSLT:**
Generic transformations from XML to other formats
- **XSL-FO:**
Transformations for producing human-readable documents, using device-independent formatting mechanisms.
- **XPATH:**
Language which allows to specify and query nodes on an XML document

XPATH

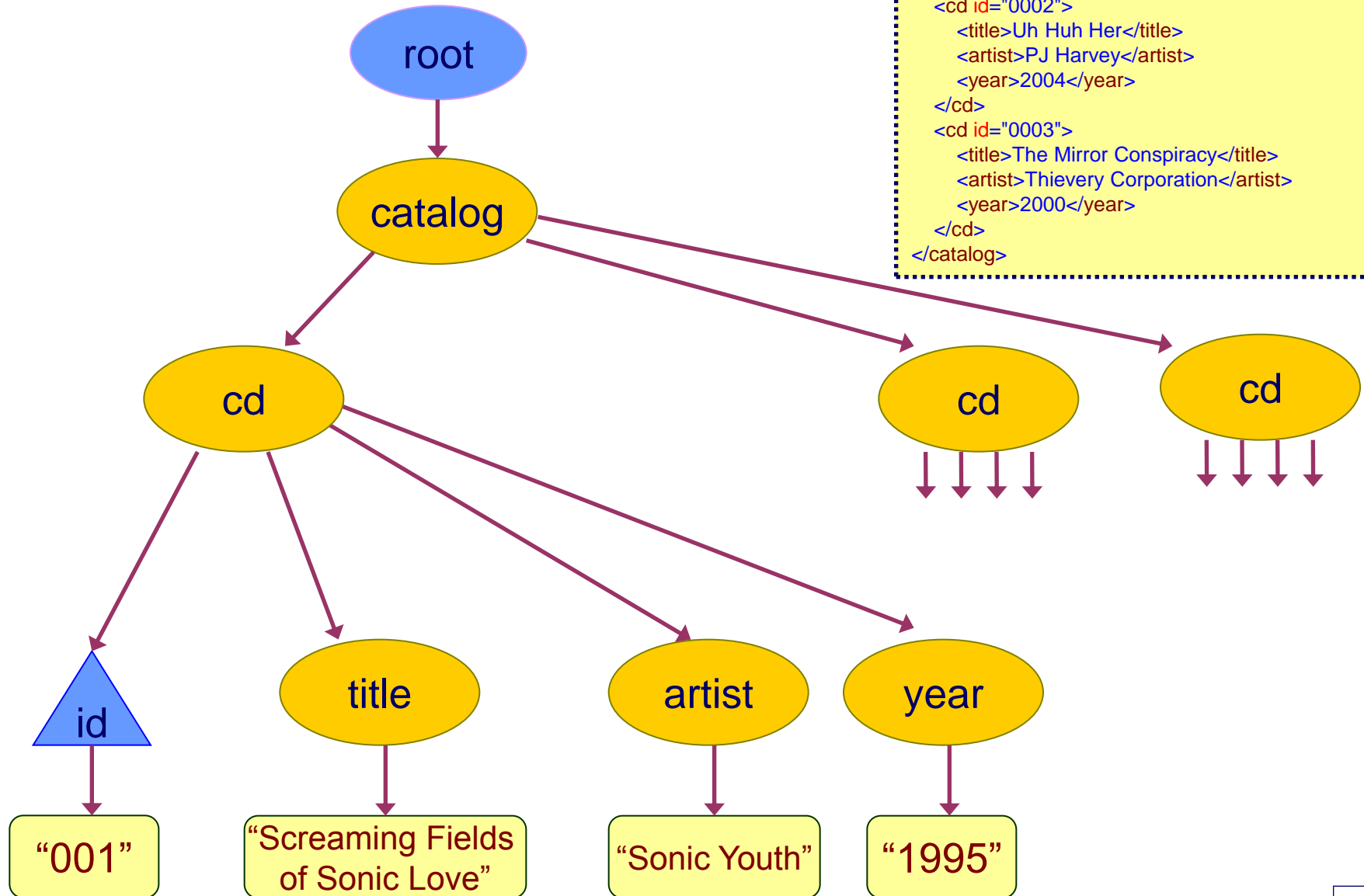
- Language used to specify parts of an XML document
- Based on paths to nodes
 - e.g. /catalog/cd/title
 - But, with an augmented syntax... e.g. //cd
- It also supports a number of powerful predefined functions

Going back to the XML example...

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

Tree Representation

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```



Node types in XPATH

- In XPATH there are seven node types:
 - Elements (e.g. "cd")
 - Attributes (e.g. "id")
 - Text (e.g. "Screaming Fields of Sonic Love")
 - Namespaces
 - Processing (e.g. "?xml")
 - Comments
 - Document (*root node*, e.g. "catalog")
- There's always a current working node
 - Equivalent to a "*working directory*" of a process
- All XPATH expressions select nodes based on the current working node

Selection Expressions

Expression	Action
cd	Select all "cd" nodes which are children of the current node
/	Root node (document). Selects the root node
//cd	Select ALL nodes "cd" independently of their position in the tree
.	Selects the current node
..	Selects the node that is above the current one
@id	Selects the "id" attribute of the currently selected node

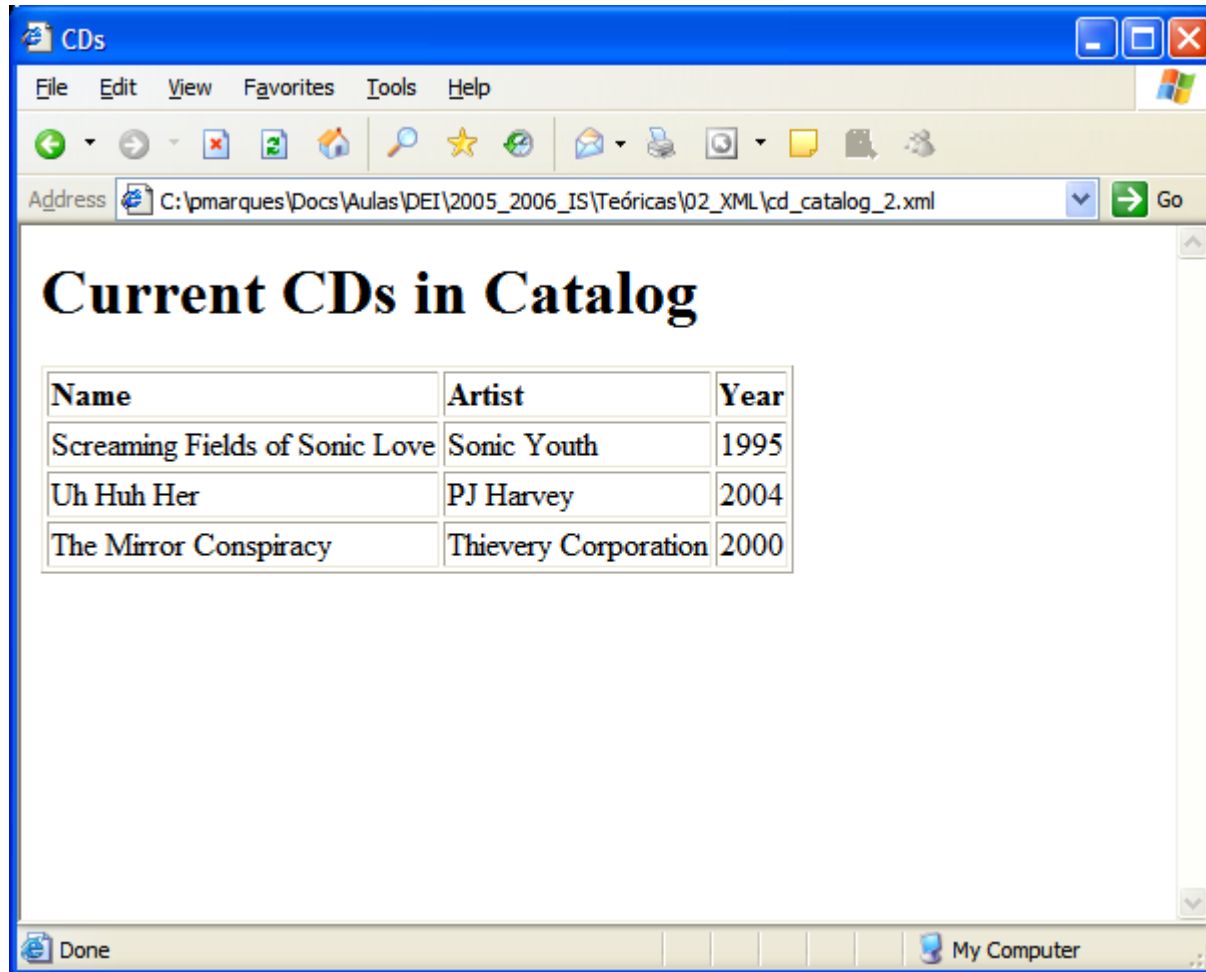
Example using XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><title> CDs </title></head> <body>
    <h1> Current CDs in Catalog</h1>
    <table border="1">
      <tr>
        <td><b> Name </b></td>
        <td><b> Artist </b></td>
        <td><b> Year </b></td>
      </tr>

      <xsl:for-each select="//cd">
        <tr>
          <td> <xsl:value-of select="title"/> </td>
          <td> <xsl:value-of select="artist"/> </td>
          <td> <xsl:value-of select="year"/> </td>
        </tr>
      </xsl:for-each>

    </table>
  </body>
</html>
```

The result



Some Examples

Expression	Action
/catalog/cd	Select <u>all</u> "cd" nodes under "catalog". Note that when used <u>directly</u> , you get a reference only for the first node. Nevertheless, you can use the result as an iterator.
/	Selects the root node
//artist	Selects all nodes named "artist" independently of their position in the document
cd/artist	Select all nodes "artist" which are children of "cd". (Note that this only works if you are at an appropriate node of the tree. Also note that if a "catalog" has several "cd"s, <u>all</u> artists associated to the cds are selected.)
/catalog//artist	Select all nodes "artist", independently of where they are on the document tree, as long as it's parent is "catalog"
//@id	Selects all "id" attributes of the document

Predicates and Functions

Expression	Action
/catalog/cd[2]	Selects the second cd of the catalog
/catalog/cd[last()]	Selects the last CD of the catalog
/catalog/cd[position()<4]	Selects the first three cds of the catalog
//cd[@id='0002']	Selects all cds which id is '0002'
//cd[year>1990]/title	Selects all titles which cds where sold after 1990
//@id	Selects all attributes id of the document

- Note that is also possible to combine several expressions (|)
 - e.g. Select all titles and years of the catalog
//title | //year

Operators and Functions

- The following operators are available in XPATH:
 - `+`, `-`, `*`, `div`, `mod`, `=`, `!=`, `<`, `<=`, `>`, `>=`, `or`, and
- There's also a huge number of functions available:
 - `number(arg)`, `abs(arg)`, `floor(arg)`, `round(arg)`, ...
 - `string(arg)`, `compare(s1,s2)`, `concat(s1,s2,...)`, `contains(s1,s2)`, `substring(s,start,len)`, `normalize-space()`, `starts-with(s1,s2)`, ...
 - `name()`, `count(item1,item2,...)`, `first()`, `last()`, `position()`, ...
 - `dateTime(date,time)`, ...

Wildcards

- It's also possible to process nodes with unknown names:
 - * → *matches* any Element
 - @* → *matches* any Attribute
 - node() → *matches* any node
- Examples:

Expression	Action
/catalog/*	Selects all nodes under "catalog"
//*	Selects all nodes in the document
//[@*]	Selects all nodes which have attributes

It's also possible to operate in terms of axis

Axis Name	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects everything in the document that is before the start tag of the current node
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Complete Expressions

- When you specify an XPATH path, absolute or relative, each step can be identified by an axis, a test node, an a selection predicate:
 - /step/step/step
 - step $\leftarrow \rightarrow$ axis::test_node[predicate]
- Examples:

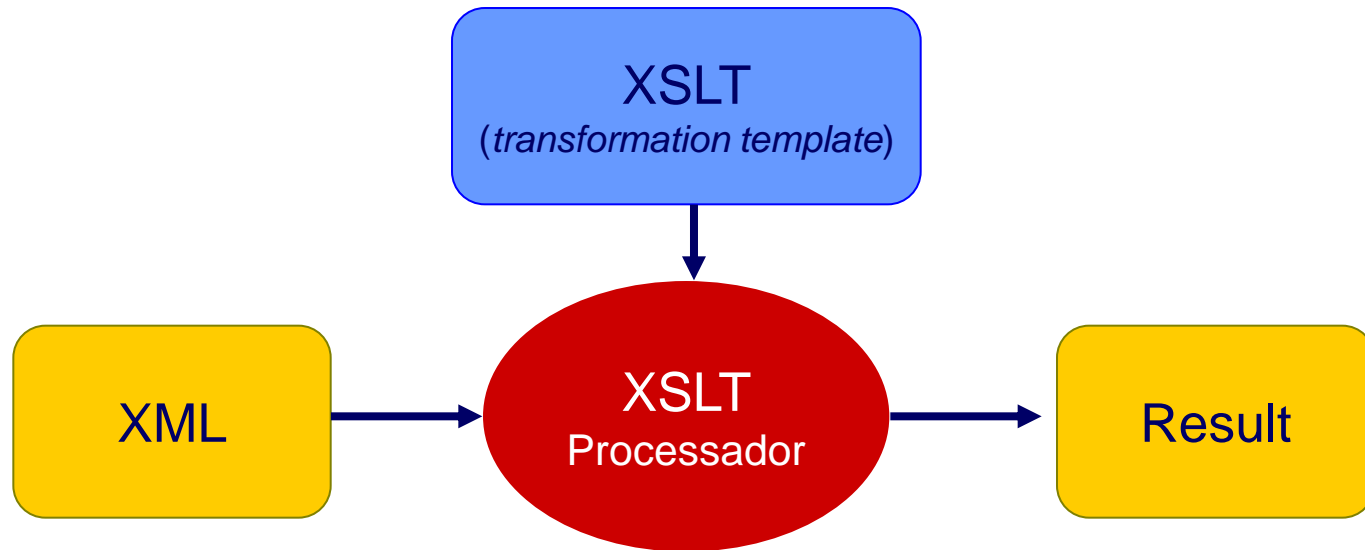
Expression	Action
child::cd	Selects all nodes under the current one which are "cd"s
attribute::id	Selects all attribute nodes of the current node which are "id"
attribute::*	...All attributes of the current node...
descendant::artist	...All descendents of the current node which are "artist"...
ancestor::cd	... All ascendants which are "cd" ...

Some final examples...

- `/catalog/cd[artist='Sonic Youth']`
 - Selects all cds in the catalog whose artist is "Sonic Youth"
- `/catalog/cd[year]`
 - Selects all cds in the catalog which have year information
- `//cd[count(artist) = 1]`
 - Selects all cds which only have one artist

XSLT

- Normally, XSLT transforms an XML tree into another XML tree
 - Nevertheless, it's not necessary that the resulting file is XML. It's not even necessary that it's a tree! Even so, typically, all transformations are based on "traversing a tree".
 - When dealing with web user interfaces is common to have: XML → XHTML



Specifying XSLT stylesheets (XSL)

- Firstly: It's not necessary to modify the source XML!
 - When calling the XSLT processor, you can specify the source and destination files to apply the XSL.
 - It wouldn't make sense to always format the files in the same way...
- Nevertheless, you can do it!

```
<?xml version="1.0" encoding="UTF-8"?>  
<?xml-stylesheet type="text/xsl" href="catalog_to_html.xsl"?>  
<catalog>  
  ...  
</catalog>
```

Structure of an XSLT file

```
<?xml version="1.0" encoding="UTF-8" ?>  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  
  <!-- TEMPLATES TO APPLY AT THE ORIGINAL FILE -->  
  
</xsl:stylesheet>
```

But... what's a “template to apply at the original file”??

```
<xsl:template match="cd">  
  <b> <xsl:value-of select="title"/> </b>  
  <br/>  
</xsl:template>
```

A complete example... (1)

```
<?xml version="1.0" encoding="UTF-8" ?>  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">  
  <html>  
    <body>  
      <h1> Uma bela colecção de CDs!</h1>  
  
    </body>  
  </html>  
</xsl:template>
```

Template 1

```
<xsl:template match="cd">  
  <b>  
    <xsl:value-of select="title"/>  
  </b>  
  <br/>  
</xsl:template>
```

Template 2

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

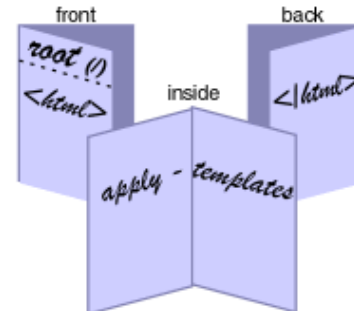
A complete example... (2)

■ But...

- If matches are done using the first template, how are the others applied?
- And, how can all matches be applied in an hierarchical way?

■ **Solution:** `<xsl:apply-templates/>`

- Guaranties the correct sequencing of all the XML structures and substructures. This happens because it applies all templates that match recursively.
- What this means is that every time an `<xsl:apply-templates/>` is seen, the system tries to re-apply the templates over that match.



A complete example... (3)

```
<?xml version="1.0" encoding="UTF-8" ?>  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">  
  <html>  
    <body>  
      <h1> Uma bela colecção de CDs!</h1>  
  
      <xsl:apply-templates/>  
  
    </body>  
  </html>  
</xsl:template>
```

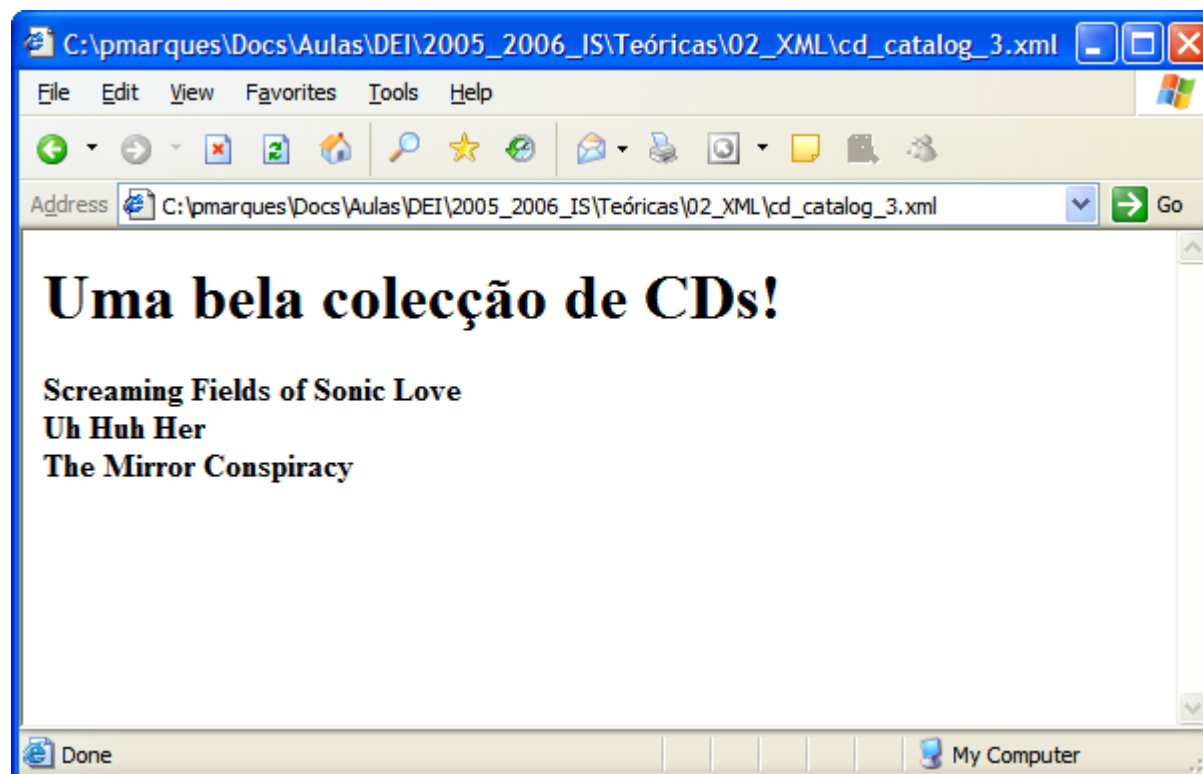
Template 1

```
<xsl:template match="cd">  
  <b>  
    <xsl:value-of select="title"/>  
  </b>  
  <br/>  
</xsl:template>
```

Template 2

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


Result



A more realistic example

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head> <title> CDs </title> </head>
      <body>
        <h1> Current CDs in Catalog </h1>

        <table border="1">
          <tr>
            <td> <b> Name  </b> </td>
            <td> <b> Artist  </b> </td>
            <td> <b> Year   </b> </td>
          </tr>

          <xsl:apply-templates/>

        </table>
      </body>
    </html>
  </xsl:template>

  <!-- (...) next slide -->
```

A more realistic example... (2)

<!-- (...) from the previous slide -->

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

```
  <tr>
```

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

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

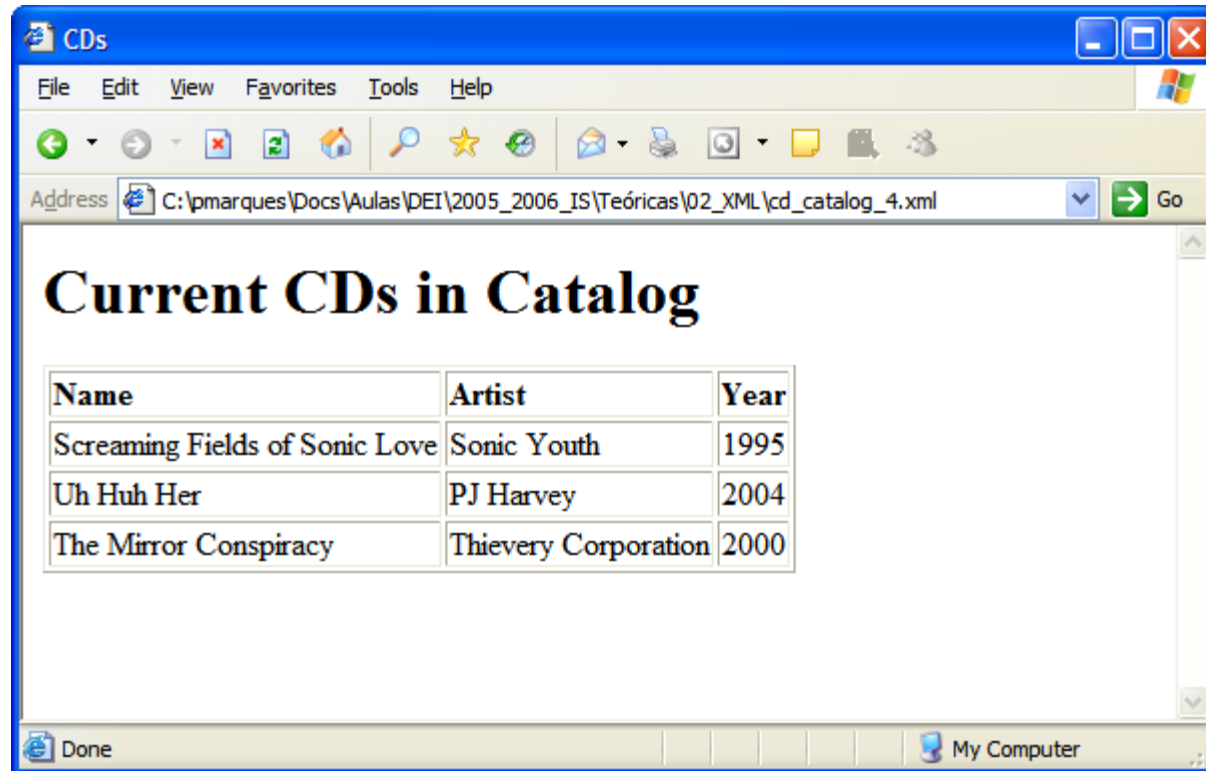
```
    <td> <xsl:value-of select="year"/> </td>
```

```
  </tr>
```

```
</xsl:template>
```

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

Result...



Some considerations...

- `<xsl:template match="???">`
 - Associates a template to a set of nodes in the XML tree
- `<xsl:value-of select="???">`
 - Extracts the value of an XML element according to the XPATH expression being specified. The value is included in the resulting file.

Note that you don't always have to use *templates*!

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><title> CDs </title></head> <body>
    <h1> Current CDs in Catalog</h1>
    <table border="1">
      <tr>
        <td><b> Name </b></td>
        <td><b> Artist </b></td>
        <td><b> Year </b></td>
      <tr>

        <xsl:for-each select="//cd">
          <tr>
            <td> <xsl:value-of select="title"/> </td>
            <td> <xsl:value-of select="artist"/> </td>
            <td> <xsl:value-of select="year"/> </td>
          </tr>
        </xsl:for-each>

      </table>
    </body>
  </html>
```

Some important expressions...

- `<xsl:for-each select="???">`
 - Allows you to iterate over all nodes specified using an XPATH expression
 - Obviously, the values can be extracted using `<xsl:value-of select=".">` or any other XPATH expressions. For instance: `<xsl:for-each select="//cd[artist='Sonic Youth']">`
- How would you order 'Sonic Youth's' cds according to year?

```
<xsl:for-each select="//cd[artist='Sonic Youth']">
  <xsl:sort select="year">

    <tr>
      <td> <xsl:value-of select="title"/>    </td>
      <td> <xsl:value-of select="artist"/>    </td>
      <td> <xsl:value-of select="year"/>    </td>
    </tr>
  </xsl:sort>
</xsl:for-each>
```

Some important expressions...

- Conditional expressions... `xsl:if`
 - Show only cds with one artist sold after 1995:

```
<xsl:for-each select="//cd">
  <xsl:if test="count(artist)=1 and year>1995">
    <tr>
      <td> <xsl:value-of select="title"/>    </td>
      <td> <xsl:value-of select="artist"/>    </td>
      <td> <xsl:value-of select="year"/>    </td>
    </tr>
  </xsl:if>
</xsl:for-each>
```


Some important expressions...

■ Conditional Expressions... xsl:choose

- Equivalent to a normal "switch" statement
- Formats the output according to the number of artists:

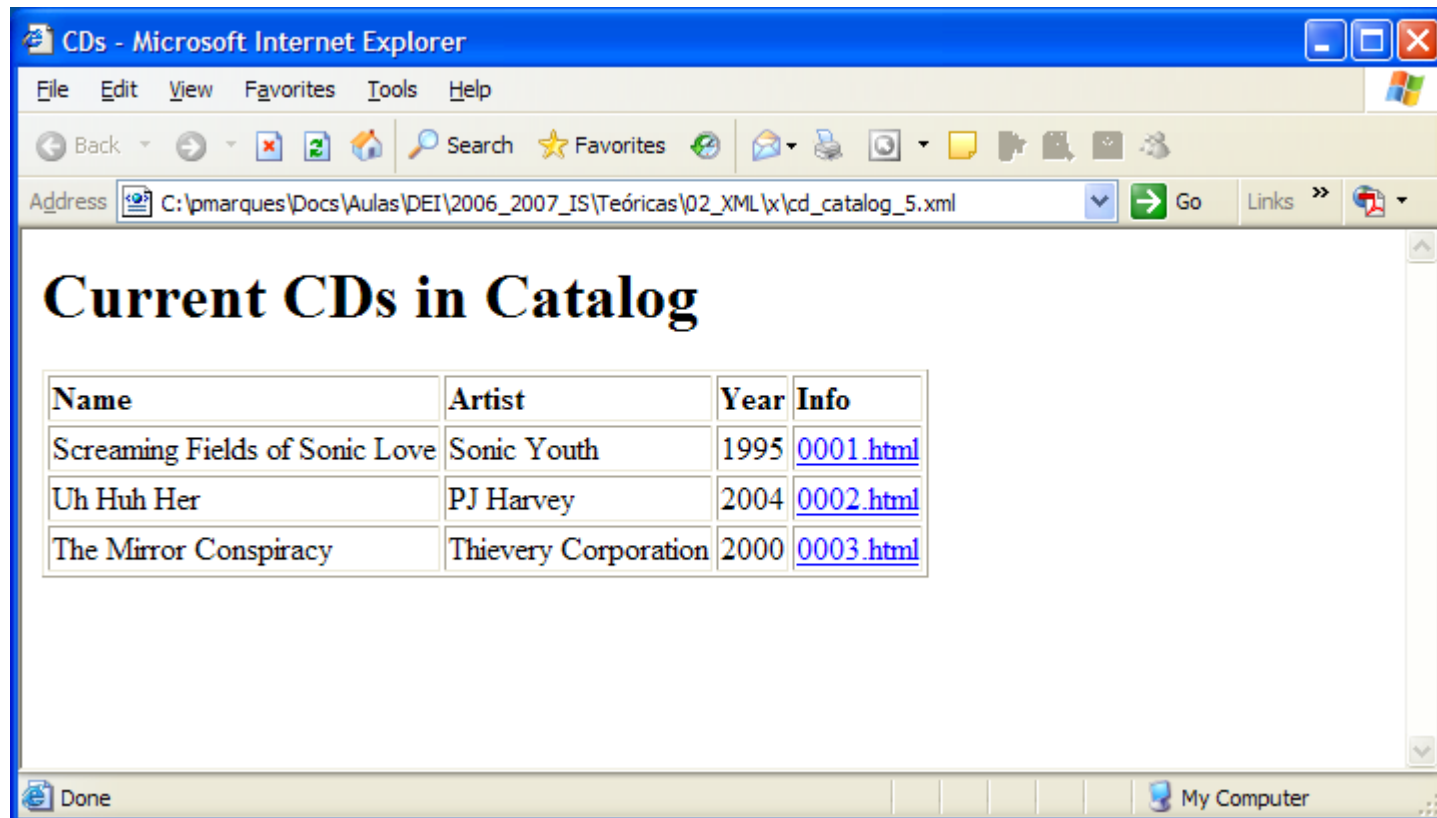
```
<xsl:for-each select="//cd">
  <td> <xsl:value-of select="title"/>      </td>
  <td> <xsl:value-of select="year"/>      </td>
  <td>

    <xsl:choose>
      <xsl:when test="count(artist)=1">
        <xsl:value-of select="artist"/>
      </xsl:when >
      <xsl:when test="count(artist)=2">
        <xsl:value-of select="artist[1]"/> AND <xsl:value-of select="artist[2]"/>
      </xsl:when >
      <xsl:otherwise>
        <xsl:value-of select="artist[1]"/> AND FRIENDS (MANY!)
      </xsl:otherwise>
    </xsl: choose >

  </td>
</xsl:for-each>
```

Consider the following problem...

- Suppose that you need to create a *link* for a web page with information about a music cd. The name of the web page with the information is the id of the cd having an “.html” extension.



But...

- This doesn't work!!!

```
<xsl:template match="cd">
  <tr>
    <td> <xsl:value-of select="title"/> </td>
    <td> <xsl:value-of select="artist"/> </td>
    <td> <xsl:value-of select="year"/> </td>

    <td>
      <a href="<xsl:value-of select="@id"/>.html">
        <xsl:value-of select="@id"/>.html
      </a>
    </td>
  </tr>
</xsl:template>
</xsl:stylesheet>
```

This is text, not an
XSLT instruction!

The solution...

```
<xsl:template match="cd">
  <tr>
    <td> <xsl:value-of select="title"/> </td>
    <td> <xsl:value-of select="artist"/> </td>
    <td> <xsl:value-of select="year"/> </td>

    <td>
      <a href="{@id}.html">
        <xsl:value-of select="@id"/>.html
      </a>
    </td>

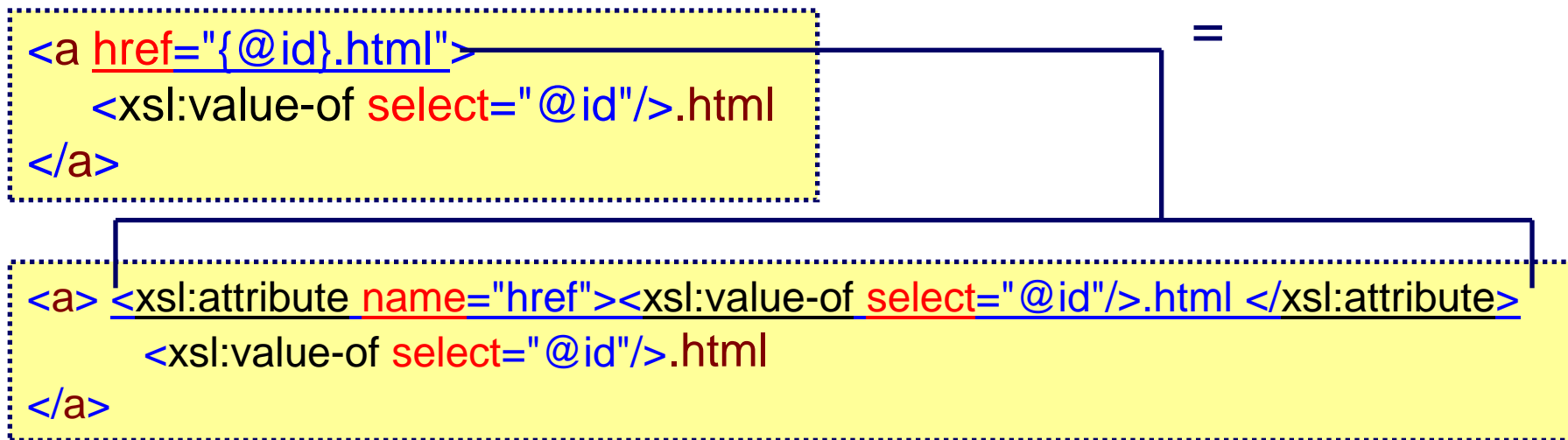
  </tr>
</xsl:template>

</xsl:stylesheet>
```

- Whenever you include an XPATH expression inside of {XXX}, its value is included in the result as text.

But, how can I generate attributes in Elements?

- As the previous as shown, some times it's necessary to determine the value of certain attributes at runtime.
 - And, some times it's even necessary to change the tags to be generated (elements) according to different information. This is especially relevant if elements are generated based on certain mathematical operations.
- So, the general solution is...



And the Elements themselves?

- Sometimes you need to generate the elements themselves based on the input XML. E.g.
 - Consider that in the original XML file there's an element named <style> which can either be "rock" or "pop". How can you programmatically generate tags which correspond to such elements?

```
<xsl:template match="cd">
  <xsl:element name="{style}">
    <artist><xsl:value-of select="title"></artist>
    <artist><xsl:value-of select="artist"></artist>
    <artist><xsl:value-of select="year"></artist>
  </xsl:element>
</xsl:template>
```

And the Elements themselves?

- Sometimes you need to generate the elements themselves

based on

```
<cd>  
  <title>Screaming Fields of Sonic Love</title>  
  <artist>Sonic Youth</artist>  
  <year>1995</year>  
  <style>rock</style>  
</cd>
```

element named
can you
to such



```
<rock>  
  <title>Screaming Fields of Sonic Love</title>  
  <artist>Sonic Youth</artist>  
  <year>1995</year>  
</rock>
```

```
</xsl:element>  
</xsl:template>
```

Including other documents... (Enriching)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
    <price>15</price>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <price>15</price>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <price>20</price>
  </cd>
</catalog>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase number="34243">
  <name>Carlos Manuel</name>
  <address>Coimbra</address>
  <items>
    <cd id="0001" quantity="2"/>
    <cd id="0003" quantity="5"/>
  </items>
</purchase>
```


The resulting document...

```
<?xml version="1.0" encoding="UTF-8"?>
<invoice id="34243">
  <name>Carlos Manuel</name>
  <address>Coimbra</address>
  <items>
    <cd id="0001" quantity="2">
      <title>Screaming Fields of Sonic Love</title>
      <artist>Sonic Youth</artist>
      <year>1995</year>
      <partial_price>30</partial_price>
    </cd>
    <cd id="0003" quantity="5">
      <title>The Mirror Conspiracy</title>
      <artist>Thievery Corporation</artist>
      <year>2000</year>
      <partial_price>100</partial_price>
    </cd>
  </items>
</invoice>
```

Including other documents...

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="cds" select="'cd_catalog.xml'"/>

<xsl:template match="/">
  <invoice id="{purchase/@number}">
    <xsl:apply-templates/>
  </invoice>
</xsl:template>

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

<xsl:template match="items">
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Including other documents...

```
<xsl:template match="cd">
  <cd id="{@id}" quantity="{@quantity}">
    <xsl:copy-of select="document($cds)//cd[@id = current()/@id]/artist"/>
    <xsl:copy-of select="document($cds)//cd[@id = current()/@id]/title"/>
    <xsl:copy-of select="document($cds)//cd[@id = current()/@id]/year"/>
    <partial_price>
      <xsl:value-of
select="(document($cds)//cd[@id = current()/@id]/price) * @quantity"/>
    </partial_price>
  </cd>
</xsl:template>
```

- **xsl:variable**
 - Defines a new variable, accessible by *\$name*
- **xsl:copy**
 - Copies the current node
- **xsl:copy-of**
 - Copies the current node and its children
- **document(string)**
 - Uses a different XML document as source

Regarding variables...

- In XSLT **variables cannot change value!** This implies that certain expressions are extremely hard to write in XSLT.
 - e.g. try to calculate the final price of a purchase order based on several items by using just XSLT. It's possible, but hard!
- Even so, variables can be the result of any "direct" expression, as long as it's non-iterative.

```
<xsl:variable name="book_stock" select="count(document($cds)//cd)"/>
```

```
<h1>The number of books in stock: <xsl:value-of select="$books_stock"/> </h1>
```

- If you need something more powerful, you should try XQUERY!

Enterprise Application Integration

2. XML Transformations

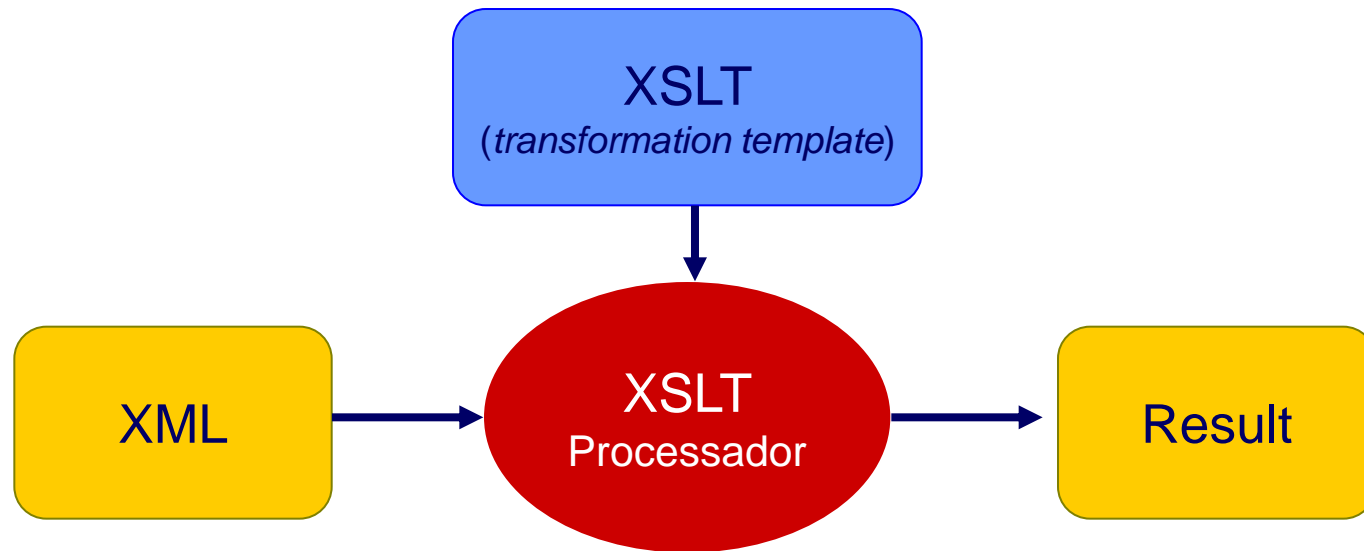
2.2. Programming Interfaces



Paulo Marques
Informatics Engineering Department
University of Coimbra
pmarques@dei.uc.pt

How to use XSLT (and XPATH)

- An XSLT processor can be:
 - A stand-alone application
 - A processing engine which is part of another program (e.g. Web Browser)
 - A library (e.g. Jaxen, Apache Xalan API)



XPATH, using... JDOM + JAXEN

```
import org.jdom.input.*;
import org.jdom.xpath.*;
import org.jdom.*;
import java.util.*;

public class CatalogXPath
{
    public static void main(String[] args)
    {
        try
        {
            // Parse our XML file into a Document object (uses SAX)
            SAXBuilder builder = new SAXBuilder();
            Document myXML = builder.build(args[0]);

            // Build an XPATH expression that returns all cds
            XPath cdsPath = XPath.newInstance("//cd");

            // Obtain the cd nodes from the document
            List cds = cdsPath.selectNodes(myXML);
        }
    }
}
```

XPATH, using... JDOM + JAXEN

```
// Iterate along the cds printing them out
Iterator it = cds.iterator();
while (it.hasNext())
{
    // Get the details of the current CD and print them out
    Element cd = (Element) it.next();

    String id = cd.getAttributeValue("id");
    String title = cd.getChild("title").getValue();
    String artist = cd.getChild("artist").getValue();
    String year = cd.getChild("year").getValue();

    System.out.println();
    System.out.println("CD #" + id + ":");
    System.out.println("-----");
    System.out.println("TITLE: \t" + title);
    System.out.println("ARTIST: \t" + artist);
    System.out.println("YEAR: \t" + year);
}
}
```


Programming XSLT with JDOM

```
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.transform.*;

(...)

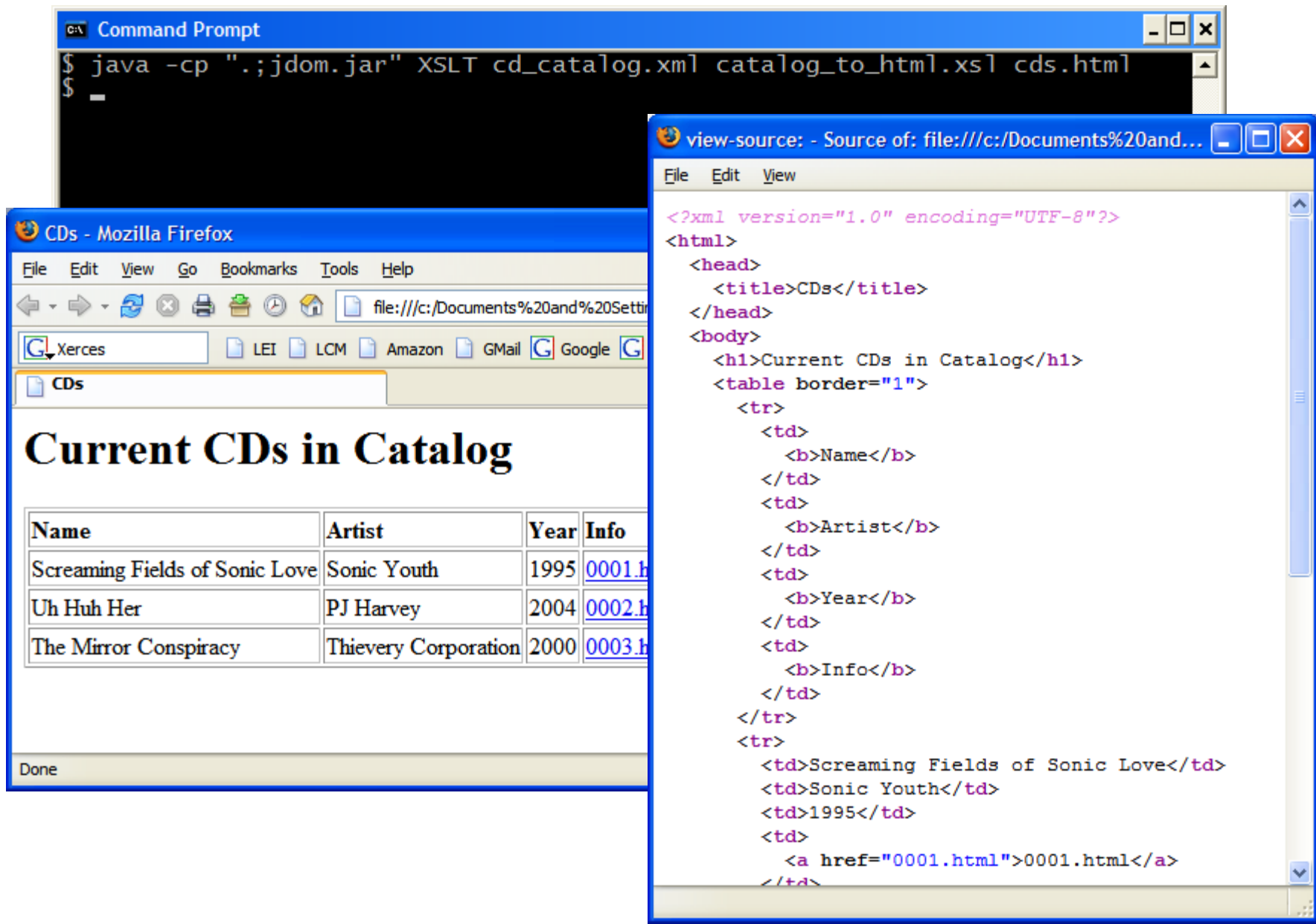
// Build a representation for the input and output documents
SAXBuilder builder = new SAXBuilder();
Document inDoc = builder.build(args[0]);

// Create an XSLT Transformer
XSLTransformer transformer = new XSLTransformer(args[1]);

// Transform the input document into the output document
Document outDoc = transformer.transform(inDoc);

// Write out the result
XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());
out.output(outDoc, new FileWriter(args[2]));
```

Programming XSLT with JDOM



Command Prompt

```
$ java -cp ".;jdom.jar" XSLT cd_catalog.xml catalog_to_html.xsl cds.html
```

view-source: - Source of: file:///c:/Documents%20and%20Settings/.../CDs

File Edit View

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>CDs</title>
  </head>
  <body>
    <h1>Current CDs in Catalog</h1>
    <table border="1">
      <tr>
        <td>
          <b>Name</b>
        </td>
        <td>
          <b>Artist</b>
        </td>
        <td>
          <b>Year</b>
        </td>
        <td>
          <b>Info</b>
        </td>
      </tr>
      <tr>
        <td>Screaming Fields of Sonic Love</td>
        <td>Sonic Youth</td>
        <td>1995</td>
        <td>
          <a href="0001.html">0001.html</a>
        </td>
      </tr>
    </table>
  </body>
</html>
```

CDs - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///c:/Documents%20and%20Settings/.../CDs

Xerces

LEI LCM Amazon GMail Google

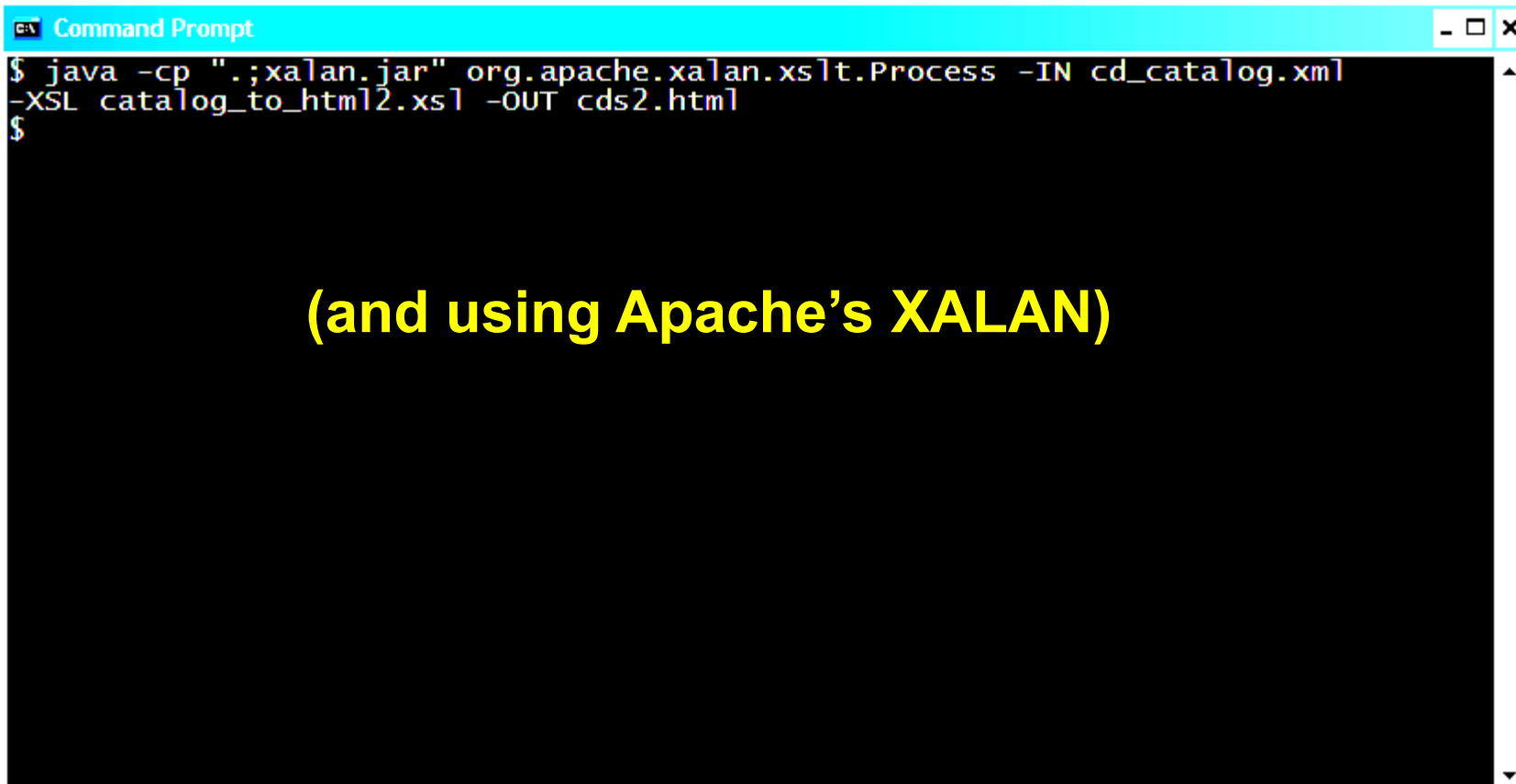
Current CDs in Catalog

Name	Artist	Year	Info
Screaming Fields of Sonic Love	Sonic Youth	1995	0001.html
Uh Huh Her	PJ Harvey	2004	0002.html
The Mirror Conspiracy	Thievery Corporation	2000	0003.html

Done

Controlling the output type

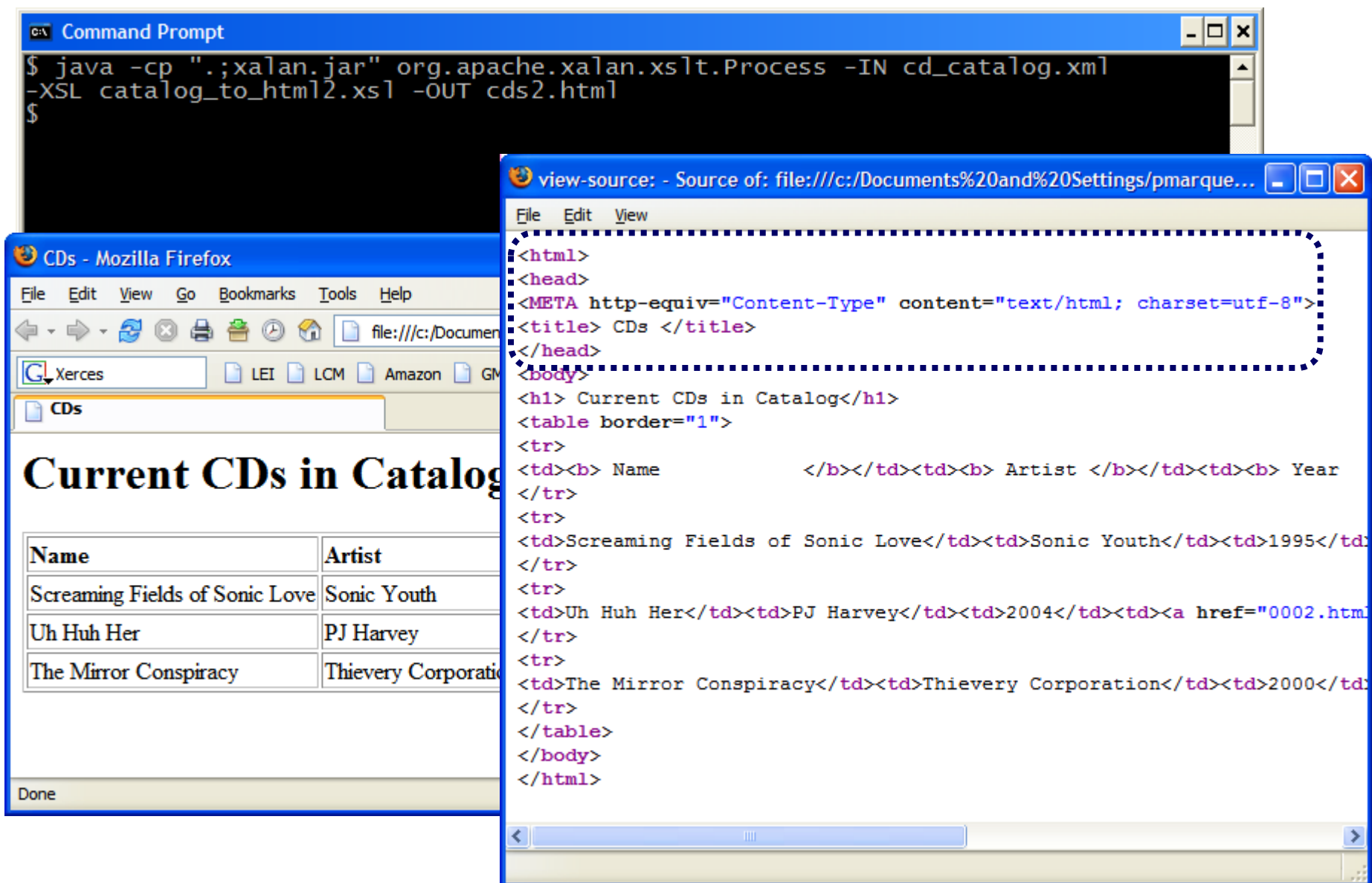
```
<?xml version="1.0" encoding="UTF-8" ?>  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
<xsl:output method="html" omit-xml-declaration="yes" encoding="utf-8"/>  
  
<!-- (...) -->  
</xsl:stylesheet>
```



```
Command Prompt  
$ java -cp ".;xalan.jar" org.apache.xalan.xslt.Process -IN cd_catalog.xml  
-XSL catalog_to_html2.xsl -OUT cds2.html  
$
```

(and using Apache's XALAN)

Controlling the output type



The image shows a Java command prompt window and a Mozilla Firefox browser window. The command prompt window displays the command to run an XSLT transformation using the xalan.jar file. The browser window shows the resulting HTML output, which includes a title "Current CDs in Catalog" and a table of CDs. The table has columns for Name, Artist, and Year. The table contains four rows of data: Screaming Fields of Sonic Love by Sonic Youth (1995), Uh Huh Her by PJ Harvey (2004), and The Mirror Conspiracy by Thievery Corporation (2000). The browser window also shows the source code of the HTML document, which is highlighted with a dashed blue box.

```
$ java -cp ".;xalan.jar" org.apache.xalan.xslt.Process -IN cd_catalog.xml -XSL catalog_to_html2.xsl -OUT cds2.html
```

view-source: - Source of: file:///c:/Documents%20and%20Settings/pmarque...

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> CDs </title>
</head>
<body>
<h1> Current CDs in Catalog</h1>
<table border="1">
<tr>
<td><b> Name </b></td><td><b> Artist </b></td><td><b> Year
```

Name	Artist	Year
Screaming Fields of Sonic Love	Sonic Youth	1995
Uh Huh Her	PJ Harvey	2004
The Mirror Conspiracy	Thievery Corporation	2000

```
</tr>
<tr>
<td>Screaming Fields of Sonic Love</td><td>Sonic Youth</td><td>1995</td>
</tr>
<tr>
<td>Uh Huh Her</td><td>PJ Harvey</td><td>2004</td><td><a href="0002.html
</tr>
<tr>
<td>The Mirror Conspiracy</td><td>Thievery Corporation</td><td>2000</td>
</tr>
</table>
</body>
</html>
```

Why use XSLT programmatically?

- The major (if not single) reason for using XSLT in this way is to be able to have fine control over the XML tree while generating the output.
 - It may be necessary to pre-process some nodes
 - It may be necessary to post-process some nodes
 - Some nodes may have to be programmatically generated
 - You may not need to generate a physical file but only an XML tree in memory

Why use XSLT programmatically?

```
<?xml version="1.0" encoding="UTF-8"?>
<invoice id="34243">
  <name>Carlos Manuel</name>
  <address>Coimbra</address>
  <items>
    <cd id="0001" quantity="2">
      <title>Screaming Fields of Sonic Love</title>
      <artist>Sonic Youth</artist>
      <year>1995</year>
      <partial_price>40</partial_price>
    </cd>
    <cd id="0003" quantity="5">
      <title>The Mirror Conspiracy</title>
      <artist>Thievery Corporation</artist>
      <year>2000</year>
      <partial_price>75</partial_price>
    </cd>
  </items>
  <total_price>115</total_price>
</invoice>
```

Do you remember the purchase example?

Manipulating the final XML tree...

```
// Transform the original document using XSLT
```

```
SAXBuilder builder = new SAXBuilder();
```

```
Document inDoc = builder.build(args[0]);
```

```
XSLTransformer transformer = new XSLTransformer(args[1]);
```

```
Document outDoc = transformer.transform(inDoc);
```

```
// Get the partial prices from the generated document and calculate the final price
```

```
XPath price = XPath.newInstance("//partial_price");
```

```
List prices = price.selectNodes(outDoc);
```

```
Iterator pricesIt = prices.iterator();
```

```
int total = 0;
```

```
while (pricesIt.hasNext())
```

```
    total+= Integer.parseInt(((Element) pricesIt.next()).getValue());
```

```
// Add the total_price node to it
```

```
Element totalPrice = new Element("total_price");
```

```
totalPrice.setText("" + total);
```

```
outDoc.getRootElement().addContent(totalPrice);
```

```
// Write out the result
```

```
XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());
```

```
out.output(outDoc, new FileWriter(args[2]));
```

Bibliography

- XPATH/XSLT – W3Schools
 - XPATH: <http://www.w3schools.com/xpath>
 - XSLT: <http://www.w3schools.com/xls>
- Online XPATH Test Tool (COOL 😊)
<http://www.activsoftware.com/xml/xpath/>
- David Jacobs, “Rescuing XSLT from Niche Status – A Gentle Introduction to XSLT through HTML Templates”,
<http://www.xfront.com/rescuing-xslt.html>
- XML Bible (2nd Edition)
by [Elliote Rusty Harold](#)
Wiley, 2001, ISBN 0764547607
- XSLT and Java
 - “Chapter 7: Extensible Stylesheet Language Transformations”, in J2EE 1.4 Tutorial



the J2EE 1.4
TUTORIAL

IMPORTANT NOTICE

YOU ARE FREE TO USE THIS MATERIAL FOR YOUR PERSONAL LEARNING OR REFERENCE, DISTRIBUTE IT AMONG COLLEAGUES OR EVEN USE IT FOR TEACHING CLASSES. YOU MAY EVEN MODIFY IT, INCLUDING MORE INFORMATION OR CORRECTING STANDING ERRORS.

THIS RIGHT IS GIVEN TO YOU AS LONG AS YOU KEEP THIS NOTICE AND GIVE PROPER CREDIT TO THE AUTHOR. YOU CANNOT REMOVE THE REFERENCES TO THE AUTHOR OR TO THE INFORMATICS ENGINEERING DEPARTMENT OF THE UNIVERSITY OF COIMBRA.

(c) 2009 – Paulo Marques, pmarques@dei.uc.pt