7. Extensible Style Language

7.1. Extensible Style Language (XSL)

- XSL is a W3C Recommendation
- rather than rules of CSS like

```
CDcollection { ... }
CD           { ... }
```

XSL uses *template rules* like

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

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

- so XSL uses XML syntax, i.e., XSL is an XML vocabulary

- it uses the namespace `https://www.w3.org/1999/XSL/Transform`, usually with prefix `xsl:`
- we will only cover a subset of XSL, namely *XSLT* (XSL transformations)

7.2. XSLT example

- consider the CD collection XML file
- CSS can be used to provide a presentation
- now consider its presentation using XSL with an XSLT file
- note that
  - elements have been reordered: e.g., `publisher`
  - elements have been processed more than once: e.g., `soloist`
  - a heading has been added

7.3. XSLT rules

- unlike CSS, XSLT rules do not just apply styles to elements
- XSLT is a language for *transforming* an input document to an output document (e.g., XHTML)
- given an XSLT rule like

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

  - the value of the `match` attribute (e.g., `CD`) is a *pattern* for matching part of the input document
  - the contents of the `template` element (e.g., `...`) is a sequence of instructions for constructing part of the output document

7.4. XSLT Program Skeleton

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

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

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

  ...

</xsl:stylesheet>
```

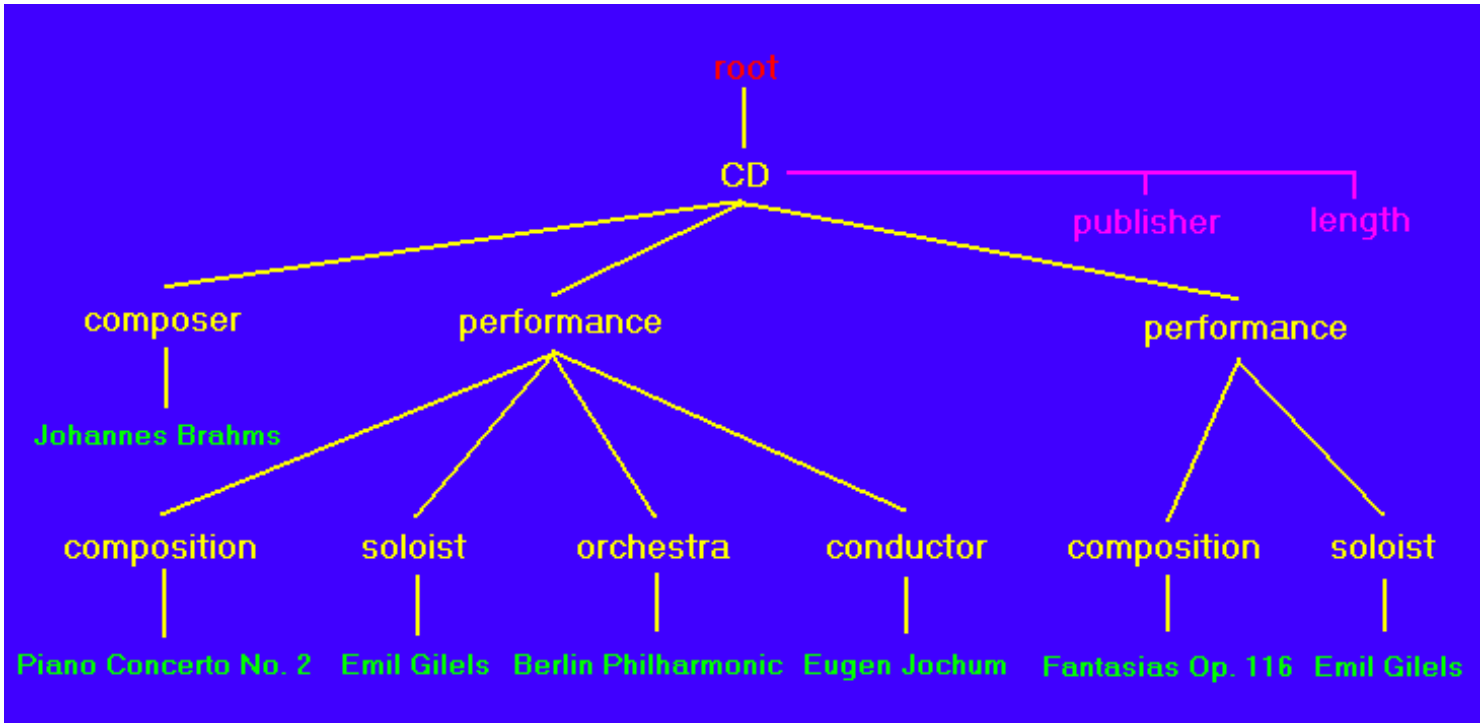
7.5. Data Model - example document

```
<CD publisher="Deutsche Grammophon"
    length="PT1H13M37S" >
  <composer>Johannes Brahms</composer>
  <performance>
    <composition>Piano Concerto No. 2</composition>
    <soloist>Emil Gilels</soloist>
    <orchestra>Berlin Philharmonic</orchestra>
    <conductor>Eugen Jochum</conductor>
  </performance>
  <performance>
    <composition>Fantasias Op. 116</composition>
    <soloist>Emil Gilels</soloist>
  </performance>
</CD>
```

- note that the CD element now has two attributes

7.6. Data Model - example tree

- document is viewed as a *tree* (hierarchy) of *nodes*



7.7. Data Model Description

- 6 types of *node*:
  - *root*, *element*, *attribute*, *text*, *comment*, *processing instruction*
- root of tree is different from (and parent of) root *element* of the document (CD in example)
- in the example slide
  - the special root node is red
  - element nodes are yellow
  - attribute nodes are pink
  - text nodes are green
- element nodes have associated set of attribute nodes
- attribute nodes are *not* children of element nodes
- order of child element nodes is *significant*

7.8. XSLT Processing Model

- processor (e.g., browser) reads an input XML document and an XSLT stylesheet
- input XML document viewed as a tree (the source tree)
- processing starts at root node of source tree
- a single node is processed by
  - finding the template rule with the best matching pattern
  - once found, executing the template instructions (often selecting more nodes to process) and creating a fragment of the output document (result tree)
  - if not found, proceeding with the list of child nodes
- a node list is processed by processing each node in order
- process continues recursively until no new source nodes are selected

7.9. RSS Example

- recall the RSS example, reproduced below

```
<rss>
  <channel>
    <title> ... </title>
      ...
    <item>
      <title> ... </title>
      <description> ... </description>
      <link> ... </link>
      <pubDate> ... </pubDate>
    </item>
      ...
    <item>
      <title> ... </title>
      <description> ... </description>
      <link> ... </link>
      <pubDate> ... </pubDate>
    </item>
  </channel>
</rss>
```

7.10. Example of an XSLT Rule

```
  <xsl:template match="channel">
    <html>
      <xsl:apply-templates select="item"/>
    </html>
  </xsl:template>
```

- template matches `channel` elements
- the value of the `match` attribute is an *XPath expression* in general (see later)
- the matched element is called the *context node*
- `<html>` and `</html>` are instructions to construct output element using *literals*
- `<xsl:apply-templates select="item"/>` is an instruction to apply templates to all `item` *children* of the context node
- the `select` attribute value is also an XPath expression
- patterns allowed in `match` are a *subset* of expressions allowed in `select`

7.11. Another Example of an XSLT Rule

```
<xsl:template match="item">
  <p>
    <xsl:value-of select="title"/>
  </p>
</xsl:template>
```

- template matches `item` elements
- `<p>` and `</p>` are literals constructing a result element named `p`
- `xsl:value-of` element is an instruction to output the value of what is selected by `select` attribute value

7.12. Example: RSS headlines

- an XSLT processor will take as input
  - the XML source `rss-fragment.xml`
  - the XML stylesheet `rss-headlines.xsl` comprising the two previous rules
- apply the stylesheet to the source to give
  - the HTML output

```
<html>
  <p>Policewoman shot during burglary</p>
  <p>Lebanon marks Hariri anniversary</p>
  <p>MPs to vote on full smoking ban</p>
</html>
```

(see `rss-fragment-headlines.html`)

7.13. Saxon and XT

- Saxon and XT are XSLT processors written in Java
- to run `xt` in the labs, you can use the batch file `xt.bat` in `n:\xmltools`
- e.g., running the following from the command line

```
n:\xmltools\xt rss-fragment.xml rss-headlines.xsl rss-fragment-headlines.html
```

- takes `rss-fragment.xml` and `rss-headlines.xsl` as input
- produces `rss-fragment-headlines.html` as output
- to use `Saxon` in the labs, you can use the batch file `saxon.bat` in `n:\SaxonHE`

```
n:\SaxonHE\saxon
   rss-fragment.xml rss-headlines.xsl rss-fragment-headlines.html
```

7.14. Using a stylesheet processing instruction

- web browsers have XSL processors built in to them
- can be invoked by including a stylesheet processing instruction in the XML source file
- processing instruction comes after the XML declaration and before the root element
- an example might be:

```
<?xml-stylesheet href="rss-headlines.xsl" type="text/xsl" ?>
```

  where the value of `href` is a URI and the value of `type` is a MIME type
- using the above stylesheet in our RSS fragment yields `rss-fragment-headlines.xml` (view the source to see the stylesheet processing instruction)

7.15. Example: RSS descriptions

applying the stylesheet rss.xsl comprising

```
<xsl:template match="channel">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <table border="1">
        <xsl:apply-templates select="item"/>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="item">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="description"/></td>
  </tr>
</xsl:template>
```

to rss-fragment.xml yields (rss-fragment.html) rss-fragment-xsl.xml as viewed in a browser with the correct stylesheet processing instruction

7.16. Example: RSS headlines (again)

- can use one rule instead of two:

```
<xsl:template match="channel">
  <html>
    <xsl:for-each select="item">
      <p>
        <xsl:value-of select="title"/>
      </p>
    </xsl:for-each>
  </html>
</xsl:template>
```

- `xsl:for-each` selects all `item` children of `channel`
- instructions given as contents of `xsl:for-each` element are applied to each `item` in turn
- note that `title` selects child elements of `item` named `title`

7.17. Some XPath expressions

- XPath is a general language for selecting nodes from an XML document tree, used in
  - `match` attribute of `xsl:template` element
  - `select` attribute of `xsl:apply-templates`, `xsl:value-of` and `xsl:for-each` elements
- we've seen the simplest kinds of expressions: simple element names like `channel`
- can build up paths of names:

  ```
  channel/title
  ```

  selects all `title` children of `channel` children of the current *context* node
- can select the *parent* of the context node: `..`
- can select the the context node itself: `.`
- can select the special extra *root node* of the tree: `/`
- can select *descendants* of the root node:

  ```
  //title
  ```

  selects all `title` children of descendants of the root (including itself)

7.18. XPath expressions

- an *XPath expression* is either
  - an *absolute expression* or
  - a *relative expression*
- an *absolute expression*
  - starts with `/`
  - is followed by a relative expression
  - and is evaluated starting at the root node
- a *relative expression* is
  - a sequence of *location steps*
  - each separated by `/`
- example (absolute expression comprising 2 steps):

```
/item/title
```

7.19. Relative expressions

- relative expression is evaluated with respect to an *initial context* (set of nodes)
- initial context is defined externally (e.g. by XSLT)

```
<xsl:template match="item">
  <xsl:value-of select="title"/>
</xsl:template>
```

  context for `title` given by `item`
- each location step
  - is evaluated with respect to some context
  - produces a set of nodes which
  - provides the context for the next location step

7.20. Simple subset of XPath

- subset uses *abbreviated syntax*
- a *location step* has one of 3 forms:
  - it is empty, i.e., `//`
  - `element-name predicates`
  - `@attribute-name predicates`
- an empty step means search all *descendants* of each node in the context
- `element-name` means find all *child* elements of each node in the context which have the given name
- `@attribute-name` means find the *attribute* node of each node in the context which has the given name
- optional `predicates` (each enclosed in `[` and `]`) filter out nodes

7.21. Example: using XPath (1)

- output all `title` elements from RSS feed
- first rule is

```
<xsl:template match="/">
  <html>
    <body>
        <xsl:apply-templates select="//title"/>
    </body>
  </html>
</xsl:template>
```

- rule matches only the root node of the document (`match="/"`)

- `select` attribute causes templates to be applied only to `title` descendents of the root node

7.22. Example: using XPath (2)

- other rules are

```
<xsl:template match="channel/title">
    <h1><xsl:value-of select="."/></h1>
</xsl:template>

<xsl:template match="image/title"/>

<xsl:template match="item/title">
  <p>
    <b><xsl:value-of select="."/></b><br />
    <xsl:value-of select="../description"/>
  </p>
</xsl:template>
```

- the first rule matches `title` elements that are children of `channel` elements
- the matched element (`title`) is selected using `.`
- the second rule matches `title` elements that are children of `image` elements and does nothing (we will see why later)
- the third rule matches `title` elements that are children of `item` elements
- the `description` element, which is a *sibling* of the matched `title` is selected using `../`
- the result of applying `rss-xpath.xsl` is `rss-fragment-xpath.xml` (`rss-fragment-xpath.html`)

7.23. Built-in template rules (1)

- if an *element* node is selected by a stylesheet but no rule matches it, the processor tries to find rules to match each of the node's children
- the following rule is effectively built in:

```
<xsl:template>
  <xsl:apply-templates/>
</xsl:template>
```

- `template` with no `match` attribute matches any node, but the above rule has the lowest priority
- `apply-templates` with no `select` attribute applies rules to all child nodes

7.24. Built-in template rules (2)

- if a *text* or *attribute* node is selected by a stylesheet but no rule matches it, the processor outputs the node's value
- the following rule is effectively built in:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

- `text()` matches text nodes
- `@` matches attribute nodes
- `*` matches any (attribute) name
- `|` matches either of its operands (`text()` or `@*`)

7.25. More XPath examples

- consider file `cd.xml`

- view results using
  - XPath Expression Testbed (available online)
  - XPath Tool (available online)
- `/CDlist/CD`: all child `CD` elements of the `CDlist` element that is the child of the root
- `//composer`: all `composer` elements that are descendants of the root
- `//performance/composer`: all `composer` child elements of `performance` elements which are descendants of the root
- `//performance[composer]`: all `performance` elements that have a `composer` element as a child
- `//CD[performance/date]`: all `CD` elements that have a `performance` element as a child that has a `date` element as a child
- `//performance[conductor][date]`: all `performance` elements that have both `conductor` and `date` elements as children

7.26. Predicates

- *predicates* filter out nodes from an ordered node-set *S*
- evaluate predicate on each node *x* in node-set *S* with
  - *x* as the *context node*
  - the size of *S* as the *context size*
  - the position of *x* in *S* as the *context position*
- predicate comprises
  - *Boolean expressions*: using `and`, `or`, `not`, `=`, ...
  - *numerical expressions*: using `+`, `-`, ...
  - *node-set expressions*: location paths filtered by predicates
  - *node-set functions*

7.27. Node-Set Functions

- `last()`: returns context size
- `position()`: returns context position
- `count(`*S*`)`: returns number of nodes in *S*
- `name(`*S*`)`: returns name of first node in *S*
- `id(`*S*`)`: returns nodes who have an `ID`-type attribute with a value in *S*
- e.g.
  - `position()=2`: true if node is 2nd in the context
  - `position()=last()`: true if node is last in the context

7.28. Examples

- `count(//performance)`: the number of `performance` elements
- `//performance[not(date)]`: `performance` elements that do not have a `date` element as a child
- all `CD` elements that have "Deutsche Grammophon" as `publisher` and have more than one `performance` element as child:

```
//CD [publisher="Deutsche Grammophon"
      and count(performance) > 1]
```

or

```
//CD [publisher="Deutsche Grammophon"]
     [count(performance) > 1]
```

or

```
//CD [count(performance) > 1]
     [publisher="Deutsche Grammophon"]
```

7.29. Some other XSLT instructions

```
<xsl:choose>
  <xsl:when test="...">
    ...
  </xsl:when>
  <xsl:otherwise>
    ...
  </xsl:otherwise>
</xsl:choose>
...
<xsl:if test="...">
  <xsl:copy-of select="..."/>
</xsl:if>
```

- use `choose` for conditional processing:
  - contents of `when` processed if result of `test` expression is true
  - contents of `otherwise` processed if result of *every* `test` expression is false
  - `if` element is used for conditional processing where there is no "else" part
  - `copy-of` copies the selected input (whole tree rooted at node) to output

7.30. Further XSLT elements

- `xsl:variable` element names a variable and assigns a value to it
- `xsl:element` element allows an element to be created with a computed name
- `xsl:attribute` element can be used to add attributes to result elements
- literal data characters may also be wrapped in an `xsl:text` element
- `xsl:comment` element is instantiated to create a comment node in the result tree
- sorting specified by adding `xsl:sort` elements as children of `xsl:apply-templates` or `xsl:for-each` element

7.31. Querying and transforming JSON

- W3C has extended its XQuery and XPath Data Model 3.1 (W3C Recommendation, 21 March 2017)
- this is the data model of Path 3.1, XSLT 3.0, and XQuery 3.1
- the model supports JSON by adding *maps* and *arrays*
- a *map* is just a collection of key/value pairs (JSON object)
- XSLT 3.0 provides two functions, `json-to-xml()` and `xml-to-json()`, to convert between JSON and XML
- XPath 3.1 (and XQuery 3.1) can query maps and arrays
- JSONiq is a query language for JSON, based on XQuery

7.32. Exercises

1. Consider an XML representation of information about students on an MSc programme. All information should be represented using elements rather than attributes. The root element of the document is `programme`. A `programme` has a `degree`, whose value might be "MSc", and a `year`, whose value might be "2018/2019". These elements are followed by the `results` for the programme. The `results` are partitioned into `distinction`, `merit`, `pass` and `fail`. Within each is a sequence of `name` elements, each containing the name of a person having achieved the corresponding `result` for the `programme`.

   Write an XSL template rule that, when matched against an XML document described above, produces an HTML document comprising a list of names of those students who obtained distinctions. The title of the document should be assembled from the contents of the degree and year elements, so that the answer when run on the document with the values suggested above would be "MSc (2018/2019)". There should be a level-2 heading "List of Distinctions", followed by an unnumbered list of names of students who

obtained a distinction.

2. Write an XSLT program which will transform an XML document of the form:

```
<teaches>
  <teaches-tuple course="IWT" lecturer="Peter Wood"/>
  <teaches-tuple course="CS" lecturer="Szabolcs Mikulas"/>
</teaches>
```

   into one of the form:

```
<teaches>
  <teaches-tuple>
    <course>IWT</course>
    <lecturer>Peter Wood</lecturer>
  </teaches-tuple>
  <teaches-tuple>
    <course>CS</course>
    <lecturer>Szabolcs Mikulas</lecturer>
  </teaches-tuple>
</teaches>
```

   You can assume that `teaches` is the root element, and that the `course` and `lecturer` attributes are required. Obviously your program should work for any number of occurrences of the `teaches-tuple` element.

3. For this exercise the source XML document is `booker.xml`. This file contains information about winners of the Booker prize. You should save a copy of this file in the directory where you intend to do the exercise. You will need to look at the document in order to see how the elements are structured.
   - Write an XSLT program to extract the *titles* of all books that have won the Booker prize. The output should be in HTML and each book title should be inserted inside double quote marks and should constitute a separate HTML paragraph.
   - Write an XSLT program to produce a table of winners of the Booker prize. The output should be in HTML, with a heading "Winners of the Booker Prize" (excluding the quotes). This should be followed by a table with column headings "Author", "Book title" and "Year" (excluding the quotes). Each row of the table should include the *author*, *title* and *year* of a Booker prize winner.

     Note that the rows in the table are ordered by author name. You can change this ordering by using the `xsl:sort` element. This empty element is placed as the contents of an `xsl:apply-templates` element or as the first child of an `xsl:for-each` element. Attributes include `order`, with values "ascending" (the default) and "descending", `data-type`, with values "text" (the default) and "number", and `select`, to order elements by the values of, for example, one of its child elements.

     Modify your program to order the table by descending year of award.

7.33. Links to more information

- The notes for the course are produced by applying
  - the stylesheet in `notes.xsl`
  - to the source XHTML, e.g. for this section `xsl.html`
  - giving the result `notes.html`
- `www.w3.org/Style/XSL/`
  W3C's XSL home page
- `www.w3.org/TR/xslt`
  W3C's XSLT page
- `hands-on-xsl.pdf`
  hands-on XSL: a simple exercise demonstrating the principles of XSLT (previously available from IBM

developerWorks)

- nwalsh.com/docs/tutorials/xsl/
  an XSL tutorial by Paul Grosso and Norman Walsh
- www.zvon.org/xxl/XSLTreference/Output/
  an XSLT reference using examples; links to other XML tutorials
- metalab.unc.edu/xml/books/bible/updates/14.html
  a chapter from the XML Bible on XSL Transformations (and XPath)
- saxon.sourceforge.net/
  SAXON, an XSLT implementation written in Java
- chris.photobooks.com/xml/default.htm
  an online resource for evaluating XPath expressions and XSLT stylesheets
- github.com/ghislainfourny/jsoniq-tutorial
  a tutorial on JSONiq and link to an online sandbox for testing

XSLT is covered in Chapter 6 and 7 of [Jacobs] and in Chapter 5 of [Moller and Schwartzbach].