6 XSL: Extensible Stylesheet Language

- An advanced style language for XML documents:
  1. Language for transforming XML documents: XSLT
  2. XML vocabulary (of formatting objects) for specifying formatting semantics:
     XSL version 1.0, W3C Candidate Rec. (21-Nov-2001)

- Introduction and Overview
- Using XSL Formatting Objects

What is it?

- An XSL style sheet specifies the presentation of a class of XML documents
  - by describing an XSLT transformation of the XML document into an XML document that uses the formatting vocabulary
  - XSL FO: a markup language to describe formatting

- XSL builds on CSS2 and DSSSL
  - DSSSL a standardised but mainly unimplemented SGML style language

Example of XSL syntax

- Formatting paragraph elements (p):
  - NB: An incomplete style sheet!

```xml
<xml version='1.0'/>
<xsl:stylesheet version='1.0'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format" >
  <xsl:template match="p">
    <fo:block>
      <fo:initial-property-set
        font-variant="small-caps"/>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
</xml:stylesheet>
```

6.1 Overview of XSL Formatting

- A style sheet processor accepts an XML document and an XSL style sheet, and produces a formatted presentation

- Two steps:
  - tree transformation: XML source tree → result tree (using XSLT)
  - formatting
    - by interpreting the result tree to produce formatted presentation

Transformation & Formatting

- Tree transformation adds information needed to format the result tree
- Formatting semantics expressed using a formatting vocabulary, of
  - formatting objects (FOs), nodes of the result tree
    - for typographic abstractions like page, block, line, page reference, ...
    - XSL 1.0 defines 56 formatting object classes
  - formatting properties control the presentation of formatting objects
    - indents, spacing, fonts, ...
    - XSL 1.0 defines 246 formatting properties
Formatting

- Formatting-object tree interpreted to produce the representation
- Each FO specifies a part of pagination, layout and styling applied to its content
- Properties control the formatting of a FO
  - some directly, e.g., color
  - some through constraints, e.g., space-before, minimum
  - rendered form not uniquely defined by XSL

Area tree

- Formatting generates an area tree consisting of geometric areas
  - areas may nest: glyphs within lines, lines within blocks, blocks within a page
- Rendering causes the area tree to appear on a medium
  - areas printed on a sequence of sheets (or displayed on a single scroll in a browser)

Generating the Area Tree

- Formatting a gradual and complex process
- Element and attribute tree
  - target of transformation, source of formatting
  - consists of element, attribute, and text nodes
  - transformed into a...
- Formatting object tree
  - consists of formatting objects with properties
  - more detailed: each character its own object

Generating the Area Tree (2)

- Properties of the formatting object tree refined into traits
  - e.g., by propagating inherited properties, and computing absolute values for relative properties
  - font-size="12pt", start-indent="2em"
  - become traits
  - font-size="12pt", start-indent="24pt"
  - traits control generation of areas out of formatting objects
  - some traits only available as a result of formatting, e.g., page numbers

Generating the Area Tree (3)

Benefits of XSL

- a comprehensive model and vocabulary for expressing XML stylesheets
- pagination and layout model extend existing ones
  - area model a superset of the CSS2 box model
- support of non-western-language directions
  - distances may be specified in terms of before, after, start and end, relative to writing-mode
- source selection and manipulation (XPath/XSLT) powerful
Each area tree node (except root) is an area, and is associated to a rectangular portion of the output medium.

Areas are generated by formatting objects.

An area has a content-rectangle:
- portion for child areas.
- optionally surrounded by a border and padding.

Content, Padding and Border

XSL contains (for compatibility) also CSS-like margins; normally margins of blocks controlled through space-before, space-after, start-indent and end-indent.

Two area types

- **block-areas**
  - generated in block-progression-direction (normally top-to-bottom).
  - paragraphs normally rendered using `fo:block`, which creates block areas.
  - line-area a special case: no borders or padding.

- **inline-areas**
  - generated in inline-progression-direction (normally left-to-right).
  - characters rendered using `fo:character`, which generates glyph-area inline-areas.
  - no child areas, a single glyph image as content.

Formatting objects and properties

- XSL 1.0 defines 56 formatting objects...
- `page-sequence`, `simple-page-master`, `block`, `inline`, `list-block`, `list-item`, `list-item-label`, `list-item-body`, `external-graphic`, `basic-link`, `float`, `footnote`, ...
- and 246 properties
- `background-color`, `font-family`, `font-size`, `space-before`, `end-indent`, `text-align`, `text-indent`, ...

Some central formatting objects

- **fo:root**
  - top node of the formatting object tree.

- **fo:simple-page-master**
  - guides the creation of pages.
  - specifies the geometry of pages:
    - `region-body` for page content.
    - `region-before`, `region-after`, `region-start` and `region-end` for header, footer, and left and right sidebar.

- **fo:page-sequence**
  - specifies the creation of page sequences.
  - possibly different `page-sequence` (and `page-master`) for, say, each chapter.

More formatting objects

- **fo:flow**
  - child objects of `page-sequences`.
  - flows attached to regions of a page-master.
  - content of flows distributes to regions of pages.

- **NB**: No `page`-formatting objects.
  - pages created by the formatter.

- **fo:block**
  - commonly used for paragraphs, titles, ...

- **fo:inline**
  - to change properties (e.g., `font-style`) of inline text.
“Hello world” result tree as XSL document

```xml
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-name="page"> <!-- use "page" master-->
    <fo:block>Hello World</fo:block>
  </fo:page-sequence>
</fo:root>
```

Implementations?

- W3C XSL WG considers Candidate Recommendation stable, and encourages implementation
- www.xmlsoftware.com/xslfo mentions 9 products/prototypes, e.g.,
  - XEP by RenderX
    - commercial (~ $5000, April 2001)
    - Java-based XSL-FO to PS/PDF processor
  - Apache FOP

Apache FOP

- FOP (Formatting object to PDF) by J. Tauber
  - “fop: a man who pays too much attention to his appearance”
  - donated to XML Apache project (http://xml.apache.org/fop/)
- open-source freeware
- Java-based XML/XSL-FO to PDF (or MIF/PCL/TXT) processor
- Implements a subset of XSL 1.0 Candidate Rec; Version 0.18.1:
  - 39 formatting objects (out of 56)
  - 110 formatting properties (out of 246)

An XSL-FO Example

- From J. David Eisenberg: Using XSL Formatting Objects. XML.com, January 17, 2001
- XSL FOs for a CD-ROM version of a handbook of Spanish
  - No one would mark-up document instances using XSL formatting objects by hand
  - Instead, think of this as the result of an XSLT transformation
- Overall structure of `fo:root`: specification of
  - page masters, followed by
  - the content of the pages

Example: Page dimensions and margins

```xml
<fo:layout-master-set>
  <fo:simple-page-master master-name="cover"
    page-height="12cm"
    page-width="12cm"
    margin-top="0.5cm"
    margin-bottom="0.5cm"
    margin-left="1cm"
    margin-right="0.5cm">
  </fo:simple-page-master>
  <fo:simple-page-master master-name="rightPage" (identical)
  </fo:simple-page-master>
  <fo:simple-page-master master-name="leftPage" (left and right margins switched)
  </fo:simple-page-master>
</fo:layout-master-set>
```

Example: Page regions

- A simple page can contain 1-5 regions, specified by child elements of the `simple-page-master`
- Let us refine the page masters by specifying regions

Example: Page regions

- A simple page can contain 1-5 regions, specified by child elements of the `simple-page-master`
- Let us refine the page masters by specifying regions

Example: Page regions

- A simple page can contain 1-5 regions, specified by child elements of the `simple-page-master`
- Let us refine the page masters by specifying regions
Example: Region dimensions

```
<fo:simple-page-master master-name="cover"... dimensions and margins as above...>
  <fo:region-body margin-top="3"/>
</fo:simple-page-master>
<fo:simple-page-master master-name="leftPage"...>
  <fo:region-before extent="1cm"/>
  <fo:region-body margin-top="1.1cm" margin-bottom="1.1cm"/>
</fo:simple-page-master>
```

Regions of rightPage identical to leftPage

Note: body has no extent; it gets the remaining area – margins of body have to accommodate other regions.

Example: Page Sequences

Next: masters for sequences of content pages, using the defined simple-page-masters – repeatedly alternate masters for left and right pages:

```
<fo:page-sequence-master master-name="contents">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference master-name="leftPage" odd-or-even="even"/>
    <fo:conditional-page-master-reference master-name="rightPage" odd-or-even="odd"/>
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

Page Sequences

- Other attributes of conditional-page-master-reference to select the page master to be used:
  - page-position="first" or "last", or "rest" (neither first or last), or "any"
  - blank-or-not-blank="true" (or "false")
- For example, to generate a blank page to force chapters to end at odd numbered pages

Next: Specifying the sequences of content pages – by naming masters to be used, and attaching flows to regions

Example: Contents of the Cover Page

```
<fo:page-sequence master-name="cover">
  <fo:flow flow-name="xsl-region-body">
    <fo:block font-family="Helvetica" font-size="18pt" text-align="end">
      Spanish Review Handbook
    </fo:block>
    <fo:block font-family="Helvetica" font-size="12pt" text-align="end" space-after="36pt">
      Copyright © 2001 J. David Eisenberg
    </fo:block>
    <fo:block>
      A Catcode Production
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```

Example: Cover Page Formatted

Formatting the first page-sequence produces page:

Example: Content Pages

Finally, we specify a page-sequence for the content pages – with static-content for the header and footer, and a flow for page bodies:

```
<fo:page-sequence master-name="contents" initial-page-number="2">
  <!-- Content of the header: -->
  <fo:static-content flow-name="xsl-region-before">
    <fo:block font-family="Helvetica" font-size="10pt" text-align="center">
      Spanish Review Handbook
    </fo:block>
  </fo:static-content>
  <!-- Content of the footer: -->
  <fo:static-content flow-name="xsl-region-after">
    <fo:block font-family="Helvetica" font-size="10pt" text-align="center">
      Copyright © 2001 J. David Eisenberg
    </fo:block>
  </fo:static-content>
</fo:page-sequence>
```
Example: Content Pages Continue

- Specify content for the footer:
  <!-- static-content is repeated on every page -->
  <fo:static-content flow-name="xsl-region-after">
    <fo:block font-family="Helvetica" font-size="10pt" text-align="center">
      Página <fo:page-number /></fo:block>
  </fo:static-content>

- Finally, specify the content of page body:

Example: Content Pages Continue

- Assign a flow of blocks to region-body:
  <fo:flow flow-name="xsl-region-body">
    <fo:block font-size="14pt">
      Watch this space!
    </fo:block>
  </fo:flow>
</fo:page-sequence>

Formatting and rendering this gives …

Example: Content Pages Formatted

How About Practise?

- As said, no one would write XSL FO document instances by hand
- Instead, XSLT style rules are used to create appropriate XSL formatting objects
  - page masters for match="/"
  - page-sequences with flows for major structural elements (like chapters, or the entire document)
  - content elements would be mapped to blocks, inlines, list-blocks, tables, … as appropriate

Examples of mapping content elements

- For example, headers:
  <xsl:template match="header">
    <fo:block font-size="14pt" font-family="sans-serif" font-weight="bold" color="green" space-before="6pt" space-after="6pt">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
  </xsl:template>

Examples of mapping content elements

- Formatting in-line emphasis:
  <xsl:template match="emph">
    <fo:inline font-style="italic">
      <xsl:apply-templates/>
    </fo:inline>
  </xsl:template>
  </xsl:template>

- More examples in the exercises

Examples of mapping content elements

- Formatting in-line emphasis:
  <xsl:template match="emph">
    <fo:inline font-style="italic">
      <xsl:apply-templates/>
    </fo:inline>
  </xsl:template>
  </xsl:template>

- More examples in the exercises
Summary

- XSL is a powerful (and complex) style language for XML documents
  - allows arbitrary transformations of input documents
  - allows fine-tuned specification of formatted representation

- It is a standard!
  - well, almost: a W3C Candidate Recommendation
  - emerging implementations seem promising