Presentation of XML Documents

Patryk Czarnik

Institute of Informatics University of Warsaw

XML and Modern Techniques of Content Management – 2010/11
1. **Stylesheets**
   - Separating content and presentation
   - Associating style with document

2. **CSS**
   - Idea and applications
   - CSS sheet structure
   - Formatting

3. **XSL**
   - Presentation by transformation
   - XSLT — Introduction
   - XSL-FO
1. **Stylesheets**
   - Separating content and presentation
   - Associating style with document

2. **CSS**
   - Idea and applications
   - CSS sheet structure
   - Formatting

3. **XSL**
   - Presentation by transformation
   - XSLT — Introduction
   - XSL-FO
According to XML best practices documents contain:
- content/data
- markup for structure and meaning (semantical markup)
- no formatting

How to present?
- “hard-coded” interpretation of known document types
- external style sheets
According to XML best practices documents contain:
- content/data
- markup for structure and meaning (semantical markup)
- no formatting

How to present?
- “hard-coded” interpretation of known document types
- external style sheets
Separating content and presentation

- According to XML best practices documents contain:
  - content/data
  - markup for structure and meaning (semantical markup)
  - no formatting

- How to present?
  - “hard-coded” interpretation of known document types
  - external style sheets
<person position="starszy referent">
  <name>Dawid</name>
  <surname>Paszkiewicz</surname>
  <phone type="office">+48223213203</phone>
  <phone type="mobile">+48501502503</phone>
  <email>paszkiewicz@superfirma.pl</email>
</person>

* yellow background
* blue font and border
* italic font for name
* typewriter font for email
Stylesheets idea

Dawid Paszkiewicz
+48223213203 +48501502503
paszkiewicz@superfirma.pl
Stylesheets idea

- yellow background
- blue font and border
- italic font for name
- typewritter font for email
- font 'Times 10pt'
- 12pt for name
- abbreviation before phone number
* yellow background
* blue font and border
* italic font for name
* typewriter font for email

Dawid Paszkiewicz
+48223213203 +48501502503
paszkiewicz@superfirma.pl

Arkadiusz Gierasimczyk
+48223213213 +48501502503
gieras@superfirma.pl

* font 'Times 10pt'
* 12pt for name
* abbreviation before phone number

starszy referent
Dawid Paszkiewicz
tel. +48223213203
tel kom. +48501502503
paszkiewicz@superfirma.pl

starszy referent
Arkadiusz Gierasimczyk
tel. +48223213213
tel kom. +48501502503
gieras@superfirma.pl
Advantages of separating content and formatting

- General advantages of semantical markup
  - better content understanding and easier analysis

- Ability to show again
  - the same document after modification
  - another document of the same structure

- Formatting managed in one place
  - easy to change look of whole class of documents

- Ability to define many stylesheets for a document, depending on:
  - purpose
  - medium (screen, paper, voice)
  - detail level (full report, summary)
  - reader preferences (font size, colors, . . . )
Advantages of separating content and formatting

- General advantages of semantical markup
  - better content understanding and easier analysis
- Ability to show again
  - the same document after modification
  - another document of the same structure
- Formatting managed in one place
  - easy to change look of whole class of documents
- Ability to define many stylesheets for a document, depending on:
  - purpose
  - medium (screen, paper, voice)
  - detail level (full report, summary)
  - reader preferences (font size, colors, . . . )
Advantages of separating content and formatting

- General advantages of semantical markup
  - better content understanding and easier analysis

- Ability to show again
  - the same document after modification
  - another document of the same structure

- Formatting managed in one place
  - easy to change look of whole class of documents

- Ability to define many stylesheets for a document, depending on:
  - purpose
  - medium (screen, paper, voice)
  - detail level (full report, summary)
  - reader preferences (font size, colors, ...)

Advantages of separating content and formatting

- General advantages of semantical markup
  - better content understanding and easier analysis
- Ability to show again
  - the same document after modification
  - another document of the same structure
- Formatting managed in one place
  - easy to change look of whole class of documents
- Ability to define many stylesheets for a document, depending on:
  - purpose
  - medium (screen, paper, voice)
  - detail level (full report, summary)
  - reader preferences (font size, colors, . . . )
Standards related to XML presentation

- General:
  - *Associating Style Sheets with XML documents*

- Stylesheet languages:
  - DSSSL (deprecated, applied to SGML)
    *Document Style Semantics and Specification Language*
  - CSS
    *Cascading Style Sheets*
  - XSL
    *Extensible Stylesheet Language*
Standards related to XML presentation

- General:
  - *Associating Style Sheets with XML documents*

- Stylesheet languages:
  - DSSSL (deprecated, applied to SGML)
    *Document Style Semantics and Specification Language*
  - CSS
    *Cascading Style Sheets*
  - XSL
    *Extensible Stylesheet Language*
Standards related to XML presentation

- **General:**
  - *Associating Style Sheets with XML documents*

- **Stylesheet languages:**
  - DSSSL (deprecated, applied to SGML)
    - *Document Style Semantics and Specification Language*
  - CSS
    - *Cascading Style Sheets*
  - XSL
    - *Extensible Stylesheet Language*
Standards related to XML presentation

- General:
  - *Associating Style Sheets with XML documents*

- Stylesheet languages:
  - DSSSL (deprecated, applied to SGML)
    - *Document Style Semantics and Specification Language*
  - CSS
    - *Cascading Style Sheets*
  - XSL
    - *Extensible Stylesheet Language*
Associating Style Sheets with XML documents

- W3C Recommendation
- Processing instruction `xml-stylesheet`

One stylesheet

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

Alternative stylesheets

```xml
<?xml-stylesheet title="Blue" type="text/css" href="blue.css" ?>

<?xml-stylesheet title="Yellow" alternate="yes" type="text/css" href="yellow.css" ?>

<?xml-stylesheet title="Big fonts" alternate="yes" type="text/css" href="big.css" ?>
```
Associating Style Sheets with XML documents

- W3C Recommendation
- Processing instruction `xml-stylesheet`

**One stylesheet**

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

**Alternative stylesheets**

```xml
<?xml-stylesheet title="Blue" type="text/css" href="blue.css" ?>

<?xml-stylesheet title="Yellow" alternate="yes" type="text/css" href="yellow.css" ?>

<?xml-stylesheet title="Big fonts" alternate="yes" type="text/css" href="big.css" ?>
```
1. Stylesheets
   - Separating content and presentation
   - Associating style with document

2. CSS
   - Idea and applications
   - CSS sheet structure
   - Formatting

3. XSL
   - Presentation by transformation
   - XSLT — Introduction
   - XSL-FO
Cascading Style Sheets

- Style sheets idea beginnings: 1970s
  - motivation — different printer languages and properties

- CSS beginnings: 1994

- CSS Level 1 Recommendation: December 1996
- CSS Level 2 Recommendation: May 1998
  - implemented (more or less completely) in present-day internet browsers

- CSS Level 3 — work still in progress:
  - modularisation,
  - new features.

- CSS 2.1 — Candidate Recommendation:
  - CSS 2 corrected and made less ambiguous
  - also, a bit less ambitious (unsupported features thrown out)
Cascading Style Sheets

- **Style sheets idea beginnings**: 1970s
  - motivation — different printer languages and properties

- **CSS beginnings**: 1994
  - CSS Level 1 Recommendation: December 1996
  - CSS Level 2 Recommendation: May 1998
    - implemented (more or less completely) in present-day internet browsers
  - CSS Level 3 — work still in progress:
    - modularisation,
    - new features.
  - **CSS 2.1 — Candidate Recommendation**:
    - CSS 2 corrected and made less ambiguous
    - also, a bit less ambitious (unsupported features thrown out)
Cascading Style Sheets

- Style sheets idea beginnings: 1970s
  - motivation — different printer languages and properties

- CSS beginnings: 1994

- CSS Level 1 Recommendation: December 1996

- CSS Level 2 Recommendation: May 1998
  - implemented (more or less completely) in present-day internet browsers

- CSS Level 3 — work still in progress:
  - modularisation,
  - new features.

- CSS 2.1 — Candidate Recommendation:
  - CSS 2 corrected and made less ambiguous
  - also, a bit less ambitious (unsupported features thrown out)
Cascading Style Sheets

- Style sheets idea beginnings: 1970s
  - motivation — different printer languages and properties
- CSS beginnings: 1994
- CSS Level 1 Recommendation: December 1996
- CSS Level 2 Recommendation: May 1998
  - implemented (more or less completely) in present-day internet browsers
- CSS Level 3 — work still in progress:
  - modularisation,
  - new features.
- CSS 2.1 — Candidate Recommendation:
  - CSS 2 corrected and made less ambiguous
  - also, a bit less ambitious (unsupported features thrown out)
Cascading Style Sheets

- Style sheets idea beginnings: 1970s
  - motivation — different printer languages and properties
- CSS beginnings: 1994
- CSS Level 1 Recommendation: December 1996
- CSS Level 2 Recommendation: May 1998
  - implemented (more or less completely) in present-day internet browsers
- CSS Level 3 — work still in progress:
  - modularisation,
  - new features.
- CSS 2.1 — Candidate Recommendation:
  - CSS 2 corrected and made less ambiguous
  - also, a bit less ambitious (unsupported features thrown out)
Cascading Style Sheets

- Style sheets idea beginnings: 1970s
  - motivation — different printer languages and properties
- CSS beginnings: 1994
- CSS Level 1 Recommendation: December 1996
- CSS Level 2 Recommendation: May 1998
  - implemented (more or less completely) in present-day internet browsers
- CSS Level 3 — work still in progress:
  - modularisation,
  - new features.
- CSS 2.1 — Candidate Recommendation:
  - CSS 2 corrected and made less ambiguous
  - also, a bit less ambitious (unsupported features thrown out)
CSS applications

- First and main application: style for WWW sites
- Separation of content and formatting for HTML
- Simple stylesheets for XML

Ideologically important

- especially from Level 2
- according to web accessibility

- support for alternative presentation methods (e.g. voice)
- enabling reader to provide her/his own style (e.g. larger font and contrast colors for people with eyesight disability)
CSS applications

- First and main application: style for WWW sites
- Separation of content and formatting for HTML
- Simple stylesheets for XML

Ideologically important

- especially from Level 2
- according to web accessibility

- support for alternative presentation methods (e.g. voice)
- enabling reader to provide her/his own style (e.g. larger font and contrast colors for people with eyesight disability)
Example XML

```xml
<department id="D07">
  <name>Accountancy</name>

  <person position="accountant" id="102103">
    <name>Dawid</name>
    <surname>Paszkiewicz</surname>
    <phone type="office">+48223213203</phone>
    <phone type="mobile">+48501502503</phone>
    <email>paszkiewicz@superfirma.pll</email>
  </person>

  <person position="manager" id="102104">
    <name>Monika</name>
    <surname>Domżałowicz</surname>
    <phone type="office">+48223213200</phone>
    <phone type="mobile">+48501502513</phone>
    <email>mdom@superfirma.pll</email>
  </person>

  ...
</department>
```
Example stylesheet

```css
person {
    display: block;
    margin: 10px auto 10px 30px;
    padding: 0.75em 1em;
    width: 200px;
    border-style: solid;
    border-width: 2px;
    border-color: #002288;
    background-color: #FFFFFF;
}

person[position='manager'] {
    background-color: #DDFFDD;
}

name, surname {
    display: inline;
    font-size: larger;
}

person[position='manager'] name {
    font-weight: bold;
}
```
Result of formatting

Księgowość

starszy referent
Dawid Paszkiewicz
tel. +48223213203
tel kom. +48501502503
paszkiewicz@superfirma.pl

kierownik
Monika Domżałowicz
tel. +48223213200
tel kom. +48501502513
mdom@superfirma.pl
Selectors

- **person** — element name
- name, surname — two elements
- department name — descendant
- department > name — child
- A + B — element B directly following A
- name:first-child — ‘name’ being first child
- person[position] — test of attribute existence
- person[position='manager'] — attribute value test
- person[position~='manager'] — value is element of list
- person#k12 — ID-type attribute value
- ol.staff — equivalent to ol[class~='staff'] (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
  - `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- `person` — *element name*
- `name, surname` — *two elements*
- `department name` — *descendant*
- `department > name` — *child*
- `A + B` — *element B directly following A*
- `name:first-child` — ‘name’ being first child
- `person[position]` — *test of attribute existence*
- `person[position='manager']` — *attribute value test*
- `person[position~='manager']` — *value is element of list*
- `person#k12` — *ID-type attribute value*
- `ol.staff` — *equivalent to ol[class~='staff']* (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- **person** — element name
- **name, surname** — two elements
- **department name** — descendant
- **department > name** — child
- **A + B** — element B directly following A
- **name:first-child** — ‘name’ being first child
- **person[position]** — test of attribute existence
  - **person[position=’manager’]** — attribute value test
  - **person[position~=’manager’]** — value is element of list
- **person#k12** — ID-type attribute value
- **ol.staff** — equivalent to **ol[class~=’staff’]**
  (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
## Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — ‘name’ being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Selectors

- `person` — element name
- `name, surname` — two elements
- `department name` — descendant
- `department > name` — child
- `A + B` — element B directly following A
- `name:first-child` — 'name' being first child
- `person[position]` — test of attribute existence
- `person[position='manager']` — attribute value test
- `person[position~='manager']` — value is element of list
- `person#k12` — ID-type attribute value
- `ol.staff` — equivalent to `ol[class~='staff']` (HTML specific).
Style depending on media type

Example

```css
@media print {
    person {
        background-color: white;
        font-family: serif;
    }
}

@media screen {
    person {
        background-color: yellow;
        font-family: sans-serif;
    }
}

@media all {
    person {
        border-style: solid;
    }
}
```
display property

- Kind of visual object representing an element
- **Allowed values**: inline, block, list-item, run-in, inline-block, table, table-cell, ..., none
- Basic property in case of XML visualisation
- Rare explicit use for HTML (browsers know default values for HTML elements)
Boxes and positions

- Blocks nested according to nesting of elements in document
- Blocks layout handled by CSS
- Custom positioning possible
- `margin, padding` — external and internal margins
- `border-style, border-color, border-width` — border properties
- `position (static | relative | absolute | fixed)` — positioning policy
- `left, right, top, bottom` — position coordinates
- `width, height, min-width, max-height, ...` — size
Boxes, margins, borders

cite: W3C, CSS Level 2 Recommendation
person {
    display: block;
    margin: 10px auto 10px 30px;
    padding: 0.75em 1em;
    width: 200px;
    border-style: solid;
    border-width: 2px;
    border-color: #002288;
    background-color: #FFFFFF;
}
Text and font properties

- color, background-color, background-image — color and background,
- font-family — name or generic family of font
- font-size
- font-style, font-weight
- text-decoration
- text-align
Fonts and colors example

class company {
    display: block;
    background-color: #EEFFFF;
    color: rgb(0, 0, 33%);
    font-family: 'Bookman', 'Times New Roman', serif;
    font-size: 14pt;
}

class company > name {
    font-size: 1.5em;
    font-family: 'Verdana', 'Arial', sans-serif;
    font-weight: bold;
    text-align: center;
}

class department > name {
    font-size: 1.3em;
    font-family: 'Verdana', 'Arial', sans-serif;
    font-weight: bold;
    font-style: italic;
}
Generated content

- Ability to show text not occurring in document text content
- Access to attribute values
- Automatic numbering

Example

```css
person:before {
  content: attr(position);
}

phone[type='office']::before {
  content: 'tel. ';
}

phone[type='mobile']::before {
  content: 'kom. ';
}
```
Visualisation once again
CSS capabilities and advantages

- Great visual formatting capabilities
  - especially for single elements
- Elements distinguished basing on
  - name
  - placement within document tree
  - existence and value of attributes
- Support
  - internet browsers
  - authoring tools
- Easy to write simple stylesheets :(
CSS capabilities and advantages

- Great visual formatting capabilities
  - especially for single elements
- Elements distinguished basing on
  - name
  - placement within document tree
  - existence and value of attributes

- Support
  - internet browsers
  - authoring tools
- Easy to write simple stylesheets :)
CSS capabilities and advantages

- Great visual formatting capabilities
  - especially for single elements
- Elements distinguished basing on
  - name
  - placement within document tree
  - existence and value of attributes
- Support
  - internet browsers
  - authoring tools
- Easy to write simple stylesheets :(
CSS capabilities and advantages

- Great visual formatting capabilities
  - especially for single elements
- Elements distinguished basing on
  - name
  - placement within document tree
  - existence and value of attributes
- Support
  - internet browsers
  - authoring tools
- Easy to write simple stylesheets :)}
CSS lacks and drawbacks

- Only visualisation
  - no conversion to other formats
- Some visual formatting issues
  - e.g. confusing block size computing (moreover, incorrectly performed in the most popular Web browser)
- Not possible (CSS Level 2):
  - repeating the same content several times
  - distinguish elements basing on their contents
  - complex logical conditions
  - data processing (e.g. summing numeric values)
  - access to many documents at once
- Some things require more work than they should, e.g.:
  - to show attribute values
  - to change elements order
CSS lacks and drawbacks

- Only visualisation
  - no conversion to other formats

- Some visual formatting issues
  - e.g. confusing block size computing (moreover, incorrectly performed in the most popular Web browser)

- Not possible (CSS Level 2):
  - repeating the same content several times
  - distinguish elements basing on their contents
  - complex logical conditions
  - data processing (e.g. summing numeric values)
  - access to many documents at once

- Some things require more work than they should, e.g.:
  - to show attribute values
  - to change elements order
CSS lacks and drawbacks

- Only visualisation
  - no conversion to other formats

- Some visual formatting issues
  - e.g. confusing block size computing (moreover, incorrectly performed in the most popular Web browser)

- Not possible (CSS Level 2):
  - repeating the same content several times
  - distinguish elements basing on their contents
  - complex logical conditions
  - data processing (e.g. summing numeric values)
  - access to many documents at once

- Some things require more work than they should, e.g.:
  - to show attribute values
  - to change elements order
CSS lacks and drawbacks

- Only visualisation
  - no conversion to other formats

- Some visual formatting issues
  - e.g. confusing block size computing (moreover, incorrectly performed in the most popular Web browser)

- Not possible (CSS Level 2):
  - repeating the same content several times
  - distinguish elements basing on their contents
  - complex logical conditions
  - data processing (e.g. summing numeric values)
  - access to many documents at once

- Some things require more work than they should, e.g.:
  - to show attribute values
  - to change elements order
1 Stylesheets
- Separating content and presentation
- Associating style with document

2 CSS
- Idea and applications
- CSS sheet structure
- Formatting

3 XSL
- Presentation by transformation
- XSLT — Introduction
- XSL-FO
In order to present a document, we can transform it into a format, that we know how to present.

Examples of “presentable” XML formats:
- XHTML
- XSL Formatting Objects
Extensible Stylesheet Language (XSL)

- First generation of recommendations (1999 and 2001):
  - XSL (general framework, XSL Formatting Objects language)
  - XSLT (transformation language)
  - XPath (expression language, in particular paths addressing document fragments)
- XSL stylesheet — transformation from XML to XSL-FO
  - usually for a class of documents (XML application)
  - in practice, transformation to other formats, often (X)HTML
Extensible Stylesheet Language (XSL)

- First generation of recommendations (1999 and 2001):
  - **XSL** (general framework, XSL Formatting Objects language)
  - **XSLT** (transformation language)
  - **XPath** (expression language, in particular paths addressing document fragments)

- XSL stylesheet — transformation from XML to XSL-FO
  - usually for a class of documents (XML application)
  - in practice, transformation to other formats, often (X)HTML
Extensible Stylesheet Language (XSL)

- First generation of recommendations (1999 and 2001):
  - **XSL** (general framework, XSL Formatting Objects language)
  - **XSLT** (transformation language)
  - **XPath** (expression language, in particular paths addressing document fragments)

- XSL stylesheet — transformation from XML to XSL-FO
  - usually for a class of documents (XML application)
  - in practice, transformation to other formats, often (X)HTML
Extensible Stylesheet Language (XSL)

- First generation of recommendations (1999 and 2001):
  - **XSL** (general framework, XSL Formatting Objects language)
  - **XSLT** (transformation language)
  - **XPath** (expression language, in particular paths addressing document fragments)

- XSL stylesheet — transformation from XML to XSL-FO
  - usually for a class of documents (XML application)
  - in practice, transformation to other formats, often (X)HTML
XSL Idea

XSL Transform

Source Tree

Result Tree
(element and attribute nodes)

Result XML tree is the result of XSLT processing.

cite: W3C, XSL 1.0
XSLT — status

- Created within XSL
- Applications beyond XML visualisation
- Version 1.0:
  - October 1999, connected with XPath 1.0
  - good support in tools and libraries
- Version 2.0:
  - January 2007, connected with XPath 2.0 and XQuery 1.0,
  - new features
  - less software support
XSLT — status

- Created within XSL
- Applications beyond XML visualisation
  - Version 1.0:
    - October 1999, connected with XPath 1.0
    - good support in tools and libraries
  - Version 2.0:
    - January 2007, connected with XPath 2.0 and XQuery 1.0,
    - new features
    - less software support
XSLT — status

- Created within XSL
- Applications beyond XML visualisation
- Version 1.0:
  - October 1999, connected with XPath 1.0
  - good support in tools and libraries
- Version 2.0:
  - January 2007, connected with XPath 2.0 and XQuery 1.0,
  - new features
  - less software support
XSLT — status

- Created within XSL
- Applications beyond XML visualisation
  - Version 1.0:
    - October 1999, connected with XPath 1.0
    - good support in tools and libraries
  - Version 2.0:
    - January 2007, connected with XPath 2.0 and XQuery 1.0,
    - new features
    - less software support
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ... 

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- XSLT 2.0 processors:
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- XSLT 1.0 processors:
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...

- Authoring tools
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- **XSLT 1.0 processors:**
  - internet browsers
    - Xalan (Java and C++ libraries)
    - xsltproc, part of libxml (C, basically for Linux)
    - XML extensions of database engines
    - ...

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen)
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - `xsltproc`, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)
- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen)
**XSLT — support**

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)

- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...

- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)
- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - …
- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
XSLT — support

- **XSLT 2.0 processors:**
  - Saxon
    - Java and .NET libraries, command-line applications
    - free (Open Source) basic version
    - commercial *schema aware* version
  - XML Spy (commercial windows application)
- **XSLT 1.0 processors:**
  - internet browsers
  - Xalan (Java and C++ libraries)
  - xsltproc, part of libxml (C, basically for Linux)
  - XML extensions of database engines
  - ...
- **Authoring tools**
  - raw text editors and programmer environments (e.g. Eclipse)
  - specialised tools — rather paid (e.g. XML Spy, oXygen).
<?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" encoding="utf-8"/>

    <xsl:template match="/">
        <html>
            <head>
                <title>Staff list</title>
            </head>
            <body>
                <h1>Staff list</h1>
                <ul>
                    <xsl:apply-templates
                        select="//person"/>
                </ul>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
XSL — sheet example

...  

```xml
<xsl:template match="person">
  <li>
    <xsl:value-of select="name"/>
    <xsl:value-of select="surname"/>
    (<xsl:value-of select="phone[type='mobile']"/>)
  </li>
</xsl:template>
</xsl:stylesheet>
```
XSLT — stylesheet structure

- **Stylesheet** *(arkusz)* consists of templates.
- **Template** *(szablon)* specifies how to transform a source node into result tree fragment.
- Inside templates:
  - text and elements not from XSLT namespace → copied to result
  - XSLT instructions → affects processing
  - XPath in instructions → access to source document, arithmetic, conditions testing, . . .
- XSLT can be considered a programming language specialised for XML documents transformation.
Stylesheet \textit{(arkusz)} consists of templates.

Template \textit{(szablon)} specifies how to transform a source node into result tree fragment

Inside templates:
- text and elements not from XSLT namespace $\rightarrow$ copied to result
- XSLT instructions $\rightarrow$ affects processing
- XPath in instructions $\rightarrow$ access to source document, arithmetic, conditions testing, \ldots

XSLT can be considered a programming language specialised for XML documents transformation.
**XSLT — stylesheet structure**

- **Stylesheet** (*arkusz*) consists of templates.
- **Template** (*szablon*) specifies how to transform a source node into result tree fragment
- Inside templates:
  - text and elements not from XSLT namespace → copied to result
  - XSLT instructions → affects processing
  - XPath in instructions → access to source document, arithmetic, conditions testing, ...
- XSLT can be considered a programming language specialised for XML documents transformation.
XSLT — stylesheet structure

- **Stylesheet** (*arkusz*) consists of templates.
- **Template** (*szablon*) specifies how to transform a source node into result tree fragment
- Inside templates:
  - text and elements not from XSLT namespace → copied to result
  - XSLT instructions → affects processing
  - XPath in instructions → access to source document, arithmetic, conditions testing, . . .
- XSLT can be considered a programming language specialised for XML documents transformation.
XSLT — stylesheet structure

- **Stylesheet** *(arkusz)* consists of templates.
- **Template** *(szablon)* specifies how to transform a source node into result tree fragment
- Inside templates:
  - text and elements not from XSLT namespace → copied to result
  - XSLT instructions → affects processing
  - XPath in instructions → access to source document, arithmetic, conditions testing, ...

- XSLT can be considered a programming language specialised for XML documents transformation.
XSLT — stylesheet structure

- **Stylesheet** (*arkusz*) consists of templates.
- **Template** (*szablon*) specifies how to transform a source node into result tree fragment.
- Inside templates:
  - text and elements not from XSLT namespace → copied to result
  - XSLT instructions → affects processing
  - XPath in instructions → access to source document, arithmetic, conditions testing, . . .
- XSLT can be considered a programming language specialised for XML documents transformation.
XSLT — operation overview

- Transformation on tree level
  - Template for document node (root) started first
    - template exists even if we had not written it
  - `apply-templates` within a template — another templates called for other nodes, usually for children
  - Templates matched according to node kind and name, location within document tree, etc.
  - Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, . . .
XSLT — operation overview

- Transformation on tree level
- Template for document node (root) started first
  - template exists even if we had not written it
- `apply-templates` within a template — another templates called for other nodes, usually for children
- Templates matched according to node kind and name, location within document tree, etc.
- Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, ...
XSLT — operation overview

- Transformation on tree level
- Template for document node (root) started first
  - template exists even if we had not written it
- `apply-templates` within a template — another templates called for other nodes, usually for children
- Templates matched according to node kind and name, location within document tree, etc.
- Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, ...
XSLT — operation overview

- Transformation on tree level
- Template for document node (root) started first
  - template exists even if we had not written it
- `apply-templates` within a template — another templates called for other nodes, usually for children
- Templates matched according to node kind and name, location within document tree, etc.
- Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, ...
XSLT — operation overview

- Transformation on tree level
- Template for document node (root) started first
  - template exists even if we had not written it
- `apply-templates` within a template — another templates called for other nodes, usually for children
- Templates matched according to node kind and name, location within document tree, etc.
- Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, ...
XSLT — operation overview

- Transformation on tree level
- Template for document node (root) started first
  - template exists even if we had not written it
- `apply-templates` within a template — another templates called for other nodes, usually for children
- Templates matched according to node kind and name, location within document tree, etc.
- Additional features: conditional processing, copying values and nodes from source, generating new nodes, grouping, sorting, . . .
XPath in XSLT

- XPath expressions used for:
  - accessing nodes and values from source document
  - computing values
  - checking logical conditions

- XPath constructions:
  - paths (navigating through document tree)
  - arithmetic and logical operators
  - functions for numbers, strings, etc.
  - more in version 2.0

- XSLT 1.0 uses XPath 1.0.
- XSLT 2.0 uses XPath 2.0.
XPath in XSLT

- XPath expressions used for:
  - accessing nodes and values from source document
  - computing values
  - checking logical conditions

- XPath constructions:
  - paths (navigating through document tree)
  - arithmetic and logical operators
  - functions for numbers, strings, etc.
  - more in version 2.0

- XSLT 1.0 uses XPath 1.0.
- XSLT 2.0 uses XPath 2.0.
XPath in XSLT

- XPath expressions used for:
  - accessing nodes and values from source document
  - computing values
  - checking logical conditions

- XPath constructions:
  - paths (navigating through document tree)
  - arithmetic and logical operators
  - functions for numbers, strings, etc.
  - more in version 2.0

- XSLT 1.0 uses XPath 1.0.
- XSLT 2.0 uses XPath 2.0.
XPath in XSLT stylesheet — example

...  
  <xsl:template match="person">
    <li>
      <xsl:value-of select="name"/>
      <xsl:value-of select="./surname"/>
      (<xsl:value-of select="phone[type='mobile']"/>)
    </li>
  </xsl:template>
</xsl:stylesheet>
XSLT — transformation result

- **XSL Formatting Objects:**
  - in line with XSL idea
  - usable e.g. for printout

- **HTML and XHTML:**
  - most popular
  - usable for Web publication and on-screen view

- **Arbitrary XML:**
  - translating to another (or new) format
  - picking up or processing data (alternative to XQuery)
  - XSLT as result of XSLT

- **Plain text**
  - CSV and other text data formats
  - scripts or configuration files
  - plain representation of text documents
XSLT — transformation result

- XSL Formatting Objects:
  - in line with XSL idea
  - usable e.g. for printout
- HTML and XHTML:
  - most popular
  - usable for Web publication and on-screen view
- Arbitrary XML:
  - translating to another (or new) format
  - picking up or processing data (alternative to XQuery)
  - XSLT as result of XSLT
- Plain text
  - CSV and other text data formats
  - scripts or configuration files
  - plain representation of text documents
XSLT — transformation result

- XSL Formatting Objects:
  - in line with XSL idea
  - usable e.g. for printout
- HTML and XHTML:
  - most popular
  - usable for Web publication and on-screen view
- Arbitrary XML:
  - translating to another (or new) format
  - picking up or processing data (alternative to XQuery)
  - XSLT as result of XSLT
- Plain text
  - CSV and other text data formats
  - scripts or configuration files
  - plain representation of text documents
XSLT — transformation result

- XSL Formatting Objects:
  - in line with XSL idea
  - usable e.g. for printout

- HTML and XHTML:
  - most popular
  - usable for Web publication and on-screen view

- Arbitrary XML:
  - translating to another (or new) format
  - picking up or processing data (alternative to XQuery)
  - XSLT as result of XSLT

- Plain text
  - CSV and other text data formats
  - scripts or configuration files
  - plain representation of text documents
Transformations vs styles

cite: Szymon Zioło, XML i nowoczesne technologie zarządzania treścią
XSL Formatting Objects

- XML application designed for visual presentation
- Focused on printed media
  - page templates ("masters")
  - automatic pagination
- Normally, XSL-FO documents are not written by hand but generated via XSLT.
XSL Formatting Objects

- XML application designed for visual presentation
- Focused on printed media
  - page templates ("masters")
  - automatic pagination
- Normally, XSL-FO documents are not written by hand but generated via XSLT.
XSL Formatting Objects

- XML application designed for visual presentation
- Focused on printed media
  - page templates ("masters")
  - automatic pagination
- Normally, XSL-FO documents are not written by hand but generated via XSLT.
XSL-FO document structure

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="my-page">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="my-page">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Hello World!</fo:block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
Transformation to XSL-FO (1/2)

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:output method="xml" encoding="utf-8"/>

    <xsl:template match="/">
        <fo:root>

            <fo:layout-master-set>
                <fo:simple-page-master master-name="A4" ...>
                ...
                </fo:simple-page-master>
            </fo:layout-master-set>

            <fo:page-sequence master-reference="A4">
                <fo:flow flow-name="xsl-region-body">
                    <xsl:apply-templates />
                </fo:flow>
            </fo:page-sequence>
        </fo:root>
    </xsl:template>
</xsl:stylesheet>
<xsl:template match="person">
  <fo:block
      border-width="1.5pt"
      border-style="solid"
      border-color="#664400"
      background-color="#FFFFEE">
    ...
    <fo:block>
      <xsl:apply-templates select="phone"/>
    </fo:block>
  </fo:block>
</xsl:template>

<xsl:template match="phone">
  <fo:block>
    <xsl:choose>
      <xsl:when test="@type='mobile'">kom. </xsl:when>
      <xsl:otherwise>tel. </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
Example visualisation

Ksiągowy

Dawid Paszkiewicz
tel. +48223213203
kom. +48501502503
paszkiewicz@superfirma.pl

Monika Domągowska

tel. +48223213200
kom. +48501502513
mdom@superfirma.pl
**page-master — page template**

- Single page layout
- A document may be split in many such pages.

Example

```xml
<fo:simple-page-master master-name="A4"
page-width="297mm" page-height="210mm"
margin-top="1cm" margin-bottom="1cm"
margin-left="1cm" margin-right="1cm">
  <fo:region-body margin="3cm"/>
  <fo:region-before extent="2cm"/>
  <fo:region-after extent="2cm"/>  
  <fo:region-start extent="2cm"/>  
  <fo:region-end extent="2cm"/>
</fo:simple-page-master>
```
Content of pages

`page-sequence` results in a number of pages

`flow` content split into pages

`static-content` content repeated on all pages

`flow-name` page region reference

Stylesheet fragment

```xml
<fo:page-sequence master-reference="A4">
  <fo:static-content flow-name="xsl-region-before">
    Company <xsl:value-of select="name" /> staff
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <xsl:apply-templates />
  </fo:flow>
</fo:page-sequence>
```
Content of pages

page-sequence results in a number of pages
flow content split into pages
static-content content repeated on all pages
flow-name page region reference

Stylesheet fragment

```xml
<fo:page-sequence master-reference="A4">
  <fo:static-content flow-name="xsl-region-before">
    Company <xsl:value-of select="name" /> staff
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <xsl:apply-templates />
  </fo:flow>
</fo:page-sequence>
```
Content of pages

`page-sequence` results in a number of pages
`flow` content split into pages
`static-content` content repeated on all pages

flow-name page region reference

Stylesheet fragment

```xml
<fo:page-sequence master-reference="A4">
  <fo:static-content flow-name="xsl-region-before">
    Company <xsl:value-of select="name" /> staff
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <xsl:apply-templates />
  </fo:flow>
</fo:page-sequence>
```
Content of pages

page-sequence results in a number of pages
flow content split into pages
static-content content repeated on all pages
flow-name page region reference

Stylesheet fragment

```xml
<fo:page-sequence master-reference="A4">
  <fo:static-content flow-name="xsl-region-before">
    Company <xsl:value-of select="name" /> staff
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <xsl:apply-templates />
  </fo:flow>
</fo:page-sequence>
```
# XSL-FO tree elements

## Block level
- block
- list-block,
  - list-item,
  - list-item-label
- table, table-row, table-cell,...

## Inline level
- inline, character
- external-graphics

## Special features
- basic-link, bookmark
- footnote
- flow
- ...

---

Patryk Czarnik
06 – Presentation
XML 2010/11
Style of visual elements

- CSS roots of visual formatting model
- Attributes for style properties:
  - margin, padding, border-style...
  - background-color, background-image...
  - font-family, font-weight, font-style, font-size...
  - text-align, text-align-last, text-indent, start-indent, end-indent, wrap-option, break-before...
Lists — example

<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>
      <fo:block>First name: </fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:block margin-left="15em">Dawid</fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label>
      <fo:block>Surname: </fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:block margin-left="15em">Paszkiewicz</fo:block>
    </fo:list-item-body>
  </fo:list-item>
</fo:list-block>
Example visualisation — lists

**Księgowo**

* Stanowisko: starszy referent
  Imię: Dawid
  Nazwisko: Paszkiewicz
  tel.: +48223213203
  kom.: +48501502503
  Email: paszkiewicz@superfirma.pl

* Stanowisko: kierownik
  Imię: Monika
  Nazwisko: Domagowski
  tel.: +48223213200
  kom.: +48501502513
  Email: mdom@superfirma.pl
Tables — example

<fo:table border="solid 2pt black">
  <fo:table-header>
    <fo:table-row>
      <fo:table-cell><fo:block font-weight="bold">Surname</fo:block></fo:table-cell>
      <fo:table-cell><fo:block font-weight="bold">First name</fo:block></fo:table-cell>
    </fo:table-row>
  </fo:table-header>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell><fo:block>Paszkiewicz</fo:block></fo:table-cell>
      <fo:table-cell><fo:block>Dawid</fo:block></fo:table-cell>
    </fo:table-row>
    ...
  </fo:table-body>
</fo:table>
## Example visualisation — table

| Księgowość  |  |  |  |  |
|-------------|--------------|------------|--------------|
| Nazwisko    | Imię         | Telefons   | Email        |
| Paszkiewicz | Dawid        | +48223213203 | paszkiewicz@superfirma.com |
|             |              | +48501502503 |              |
| Domżałowicz | Monika       | +48223213200 | mdom@superfirma.pl |
|             |              | +48501502513 |              |
| Kącki       | Marek        | +48223213212 | kacki@superfirma.pl |
|             |              | +48501502524 |              |
| Woźniak     | Tomasz       | +48223213234 | wozniak@superfirma.pl |
|             |              | +48501502544 |              |
| Chmielnicki | Maciej       | +48223213236 | chmiel@superfirma.pl |
|             |              | +48501502527 |              |
| Ślusarek    | Patrycja     | +48223213216 | paszkiewicz@superfirma.com |
|             |              | +48501502594 |              |
| Koper       | Robert       | +48501502111 | paszkiewicz@superfirma.com |
XSL-FO — support

- **Commercial software:**
  - Antenna House XSL Formatter
  - RenderX
  - Ecrion
  - Lunasil LTD Xinc
  - …

- **Open software:**
  - Apache FOP
  - xmlroff
Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
- lack of pagination feedback, cannot say “if these two elements occur on the same page...”
- hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
- lack of pagination feedback, cannot say “if these two elements occur on the same page...”
- hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
- lack of pagination feedback, cannot say “if these two elements occur on the same page...”
- hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
  - lack of pagination feedback, cannot say “if these two elements occur on the same page…”
  - hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
  - lack of pagination feedback, cannot say “if these two elements occur on the same page…”
  - hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
  - lack of pagination feedback, cannot say “if these two elements occur on the same page…”
  - hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)
XSL-FO — critique

Main XSL-FO advantages

- “in line” with XSL
- easy and direct way to obtain printout (e.g. PDF) from XML data
- general advantages of stylesheets over “hard-coded” formatting

Main XSL-FO disadvantages

- too complex for simple needs
- too limited for advanced needs
  - lack of pagination feedback, cannot say “if these two elements occur on the same page…”
  - hard to format particular elements in a very special way (thus this is a general drawback of using stylesheets)