Data Transformation

Faculty of Computer Science
Workflow Systems and Technologies

Interoperability SS 2023

Amolkirat Singh Mangat
Matthias Ehrendorfer
Florian Stertz

Data Transformation

- Transform data from one format into another format
    - Between different data types (e.g. JSON to XML, YAML to XML)
    - Between different data arrangements based on some schema of the same data type (e.g. XML to XML, JSON to JSON)
- Two *standardised* transformation technologies by W3C include *XSLT and XQuery*
- **XQuery** - Querying language initially designed for (flexible) XML
    - Composed of expressions
    - Since Version 3.1 support for JSON was added
- **XSL[T] -** Extensible Stylesheet Language [Transformations]
    - Declarative pattern matching
    - Version 3.0 supports conversion of JSON into XML

Minimal Valid Examples: XSLT 1.0 vs. XQUERY 1.0

XSLT Stylesheet

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

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

</xsl:stylesheet>
```

XQUERY Expression

```
<foo/>
```

Minimal Valid Examples: XSLT 1.0 vs. XQUERY 1.0

XSLT Stylesheet

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

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

</xsl:stylesheet>
```

XQUERY Expression

```
<foo/>
```

Output

```
<foo/>
```

XQUERY: Fundamental Building Blocks I

Expression

```
<my-new-xml>
  { for $node in doc('abc.xml')/abc/a
    where $node/@foo = "g"
    return $node                      }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XQUERY: Fundamental Building Blocks I

Expression

```
<my-new-xml>
  { for $node in doc('abc.xml')/abc/a
    where $node/@foo = "g"
    return $node                      }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

/abc/a, $node/@foo = "g" and $node represent XPath Expressions!

$ is a mandatory variable prefix.

{} marks expressions the XQuery processor should evaluate. (Yes, expressions can be mixed with XML!)

XQUERY: Fundamental Building Blocks I

Expression

```
<my-new-xml>
  { for $node in doc('abc.xml')/abc/a
    where $node/@foo = "g"
    return $node                      }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`for … in …` construct iterates over XML nodes.

`doc()` loads a XML document from a file.

`return`s XML after evaluating an expression. Use `{}` for expressions if mixed with XML.

XQUERY: Fundamental Building Blocks II

Expression

```
<my-new-xml>
  { for $node in doc('abc.xml')/abc/a
    let $variable := $node
    order by $node/@foo
    return $variable                  }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XQUERY: Fundamental Building Blocks II

Expression

```
<my-new-xml>
  { for $node in doc('abc.xml')/abc/a
    let $variable := $node
    order by $node/@foo
    return $variable                  }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`let` declares a new variable `$` that can be assigned the result of an expression.
`order by` changes the order in which the nodes returned by `for` are output.

XQUERY: Fundamental Building Blocks III

Expression

```
<my-new-xml>
  { for $parent in doc('abc.xml')/abc/a
    return
      <a>
      { for $child in $parent/*
        return element
                 { $parent/@foo }
                 { attribute
                     {"type"}
                     {$child/name()},
                   $child/text()      } }
      </a>                                }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a>
    <g type="b">X</g>
    <g type="c">Y</g>
  </a>
  <a>
    <f type="b">M</f>
    <f type="c">N</f>
  </a>
</my-new-xml>
```

XQUERY: Fundamental Building Blocks III

Expression

```
<my-new-xml>
  { for $parent in doc('abc.xml')/abc/a
    return
      <a>
      { for $child in $parent/*
        return element
                 { $parent/@foo }
                 { attribute
                     {"type"}
                     {$child/name()},
                   $child/text()      } }
      </a>                              }
</my-new-xml>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a>
    <g type="b">X</g>
    <g type="c">Y</g>
  </a>
  <a>
    <f type="b">M</f>
    <f type="c">N</f>
  </a>
</my-new-xml>
```

`for` constructs (or expressions in general) can be nested!

XSLT: Fundamental Building Blocks I

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:if test="./@foo = 'g'">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XSLT: Fundamental Building Blocks I

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:if test="./@foo = 'g'">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`/`, `./@foo = 'g'` and `.` represent **XPath Expressions**!
`select` and `match` return a set of nodes.

XSLT: Fundamental Building Blocks I

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:if test="./@foo = 'g'">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`for-each` iterates over the set of nodes obtained from the `select` expression. To access a node in focus use `.`.
`if` enables to do conditional processing based on the outcome of the `test` expression.
`copy-of` returns a copy of the node and its children.

XSLT: Fundamental Building Blocks I

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:if test="./@foo = 'g'">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`xsl` refers to the namespace definition which has been excluded here due to the lack of space.
`template` provides access to a subset of the XML document tree. By convention the default template matches the root of the document tree with `template match="/"` and serves as an entry point for processing.

XSLT: Fundamental Building Blocks II

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:sort select="@foo"/>
    <xsl:variable name="var" select="."/>
    <xsl:copy-of select="$var"/>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XSLT: Fundamental Building Blocks II

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="abc/a">
    <xsl:sort select="@foo"/>
    <xsl:variable name="var" select="."/>
    <xsl:copy-of select="$var"/>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`variable` declares a variable using a XPath expression.
`$` followed by the value under `name` as suffix grants access to the node referenced by a `variable`.
`sort` orders nodes by. Note that the order in which declarations are defined can matter!

XSLT: Fundamental Building Blocks III

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="./abc/a">
   <xsl:variable name="parent" select="."/>
   <a>
    <xsl:for-each select="$parent/*">
     <xsl:variable name="child" select="."/>
     <xsl:element name="{$parent/@foo}">
      <xsl:attribute name="type">
       <xsl:value-of select="$child/name()"/>
      </xsl:attribute>
      <xsl:value-of select="$child"/>
     </xsl:element>
    </xsl:for-each>
   </a>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a>
    <g type="b">X</g>
    <g type="c">Y</g>
  </a>
  <a>
    <f type="b">M</f>
    <f type="c">N</f>
  </a>
</my-new-xml>
```

XSLT: Fundamental Building Blocks III

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="./abc/a">
   <xsl:variable name="parent" select="."/>
   <a>
    <xsl:for-each select="$parent/*">
     <xsl:variable name="child" select="."/>
     <xsl:element name="{$parent/@foo}">
      <xsl:attribute name="type">
       <xsl:value-of select="$child/name()"/>
      </xsl:attribute>
      <xsl:value-of select="$child"/>
     </xsl:element>
    </xsl:for-each>
   </a>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a>
    <g type="b">X</g>
    <g type="c">Y</g>
  </a>
  <a>
    <f type="b">M</f>
    <f type="c">N</f>
  </a>
</my-new-xml>
```

`value-of` extracts the content of a node.
`{}` evaluates an expression.
`element` and `attribute` allow dynamic node creation.

Is there an alternative way to structure XSLT Stylesheets?

XSLT: Fundamental Building Blocks IV

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
   <xsl:apply-templates/>
 </my-new-xml>
</xsl:template>

<xsl:template match="a">
 <xsl:if test="./@foo = 'g'">
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XSLT: Fundamental Building Blocks IV

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
   <xsl:apply-templates/>
 </my-new-xml>
</xsl:template>

<xsl:template match="a">
 <xsl:if test="./@foo = 'g'">
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`apply-templates` applies matching template to the current context node and to its child nodes.

Q: What happens if a matched node also has child nodes that match the pattern?

XSLT: Fundamental Building Blocks V

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
   <xsl:apply-templates select="//a"/>
 </my-new-xml>
</xsl:template>

<xsl:template match="a">
 <xsl:if test="./@foo = 'g'">
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`apply-templates` can also be limited to a specific set of nodes using `select`.

XSLT: Fundamental Building Blocks VI

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
   <xsl:apply-templates/>
 </my-new-xml>
</xsl:template>

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

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>

    X




    M


</my-new-xml>
```

Note: If no template matches, built-in template rules are applied to allow the recursive processing to continue. E.g. see the fragment `RULE_TEXT_ATTRIBUTE` which is the built-in template rule for text and attribute nodes if no matching pattern is found.

Q: What is the reason for the white space in the output?

XSLT: Fundamental Building Blocks VI

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
   <xsl:apply-templates/>
 </my-new-xml>
</xsl:template>

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

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml><o>
  </o><o>
    </o><o>X</o><o>
    </o><o>
  </o><o>
  </o><o>
    </o><o>M</o><o>
    </o><o>
  </o><o>
</o></my-new-xml>
```

Note: If no template matches, built-in template rules are applied to allow the recursive processing to continue. E.g. see the fragment `RULE_TEXT_ATTRIBUTE` which is the built-in template rule for text and attribute nodes if no matching pattern is found.

Q: What is the reason for the white space in the output?

XSLT: Fundamental Building Blocks VII

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="//a">
   <xsl:call-template name="a"/>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>

<xsl:template name="a">
 <xsl:if test="./@foo = 'g'">
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

XSLT: Fundamental Building Blocks VII

Stylesheet

```
<xsl:template match="/">
 <my-new-xml>
  <xsl:for-each select="//a">
   <xsl:call-template name="a"/>
  </xsl:for-each>
 </my-new-xml>
</xsl:template>

<xsl:template name="a">
 <xsl:if test="./@foo = 'g'">
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<my-new-xml>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
</my-new-xml>
```

`call-template` enables to explicitly apply a named `template` to a context node.

XSLT: Fun Exercise - Identity “Transformation”

Stylesheet

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

Input

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Output

```
<abc>
  <a foo="g">
    <b>X</b>
    <c>Y</c>
  </a>
  <a foo="f">
    <b>M</b>
    <c>N</c>
  </a>
</abc>
```

Resources

https://www.w3.org/TR/xquery-31/

https://www.w3.org/Style/XSL/

https://www.w3.org/TR/xslt/

https://developer.mozilla.org/en-US/docs/Web/XSLT