XSLT Constructors

Martin Holmes
XSLT Constructors

- Normally, to create a result element in your output, you just type it literally:

  `<div class="chapter"> [...] </div>`

- However, there are some circumstances when you can't do that. For instance, the name of the element you want to create may change based on the XML input. For instance:
  - If you're processing a TEI `<list>` element, you may need to produce either `<ul>` or `<ol>` in your output, depending on whether it's an unordered or an ordered list:

    ```xml
    <list type="bulleted"> # <ul>
    <list type="ordered"> # <ol>
    ```

- We can handle this with the XSLT *element constructor*.

The element constructor

```xml
<xsl:element name="h2">
    [...]contents of the h2 element...]
</xsl:element>
```

An example element constructor

```xml
<xsl:element name="(if (@type='ordered') then 'ol' else 'ul')">
    [...]  
</xsl:element>
```

- The XPath logic is contained in {curly braces}.
- Be careful with nested quotes!
The attribute constructor

Just as you can create an element with a constructor, you can also create an attribute:

```xml
<xsl:attribute name="class">
    [... value of the class attribute ...]
</xsl:attribute>
```

You might do this if you need to generate the attribute value dynamically based on the content.

Like `<xsl:variable>`, the value can be given as the value of a `@select` attribute, or as the content.

An example attribute constructor

```xml
<xsl:attribute name="class">
    <xsl:choose>
        <xsl:when test="local-name() = 'p'">paragraph</xsl:when>
        <xsl:when test="local-name() = 'ab'">anonBlock</xsl:when>
        <xsl:otherwise>paraLike</xsl:otherwise>
    </xsl:choose>
</xsl:attribute>
```

XSL constructors: Task 1

Write a constructor for an element called `<div>` with an attribute called "class", whose value is "chapter".
**XSL constructors: Task 1 answer**

A constructor for an element `<div>` with a `@class` attribute whose value is "chapter":

```xml
<xsl:element name="div">
   <xsl:attribute name="class">chapter</xsl:attribute>
</xsl:element>
```

**XSL constructors: Task 1 discussion**

A constructor for an element `<div>` with a `@class` attribute whose value is "chapter":

```xml
<xsl:element name="div">
   [What can go in this location?]
   <xsl:attribute name="class">chapter</xsl:attribute>
   [What can go in this location?]
</xsl:element>
```
Copy constructors: `<xsl:copy>` and `<xsl:copy-of>`

- Sometimes, you simply want to copy some of the XML in your tree to the output document.
- This is often the case when you’re using XSLT to make small modifications to your XML files.
- You can use `<xsl:copy>` and `<xsl:copy-of>` to do this.
- Although they look similar, they have completely different effects.

```xml
<xsl:copy-of />
```

- `<xsl:copy-of />` is the simplest of the two copy constructors.
- It makes a complete copy of the node, including its attributes and descendants, in the output tree.
- For instance, if your input is this:

```xml
<p xml:id="para_1">
  This is my first paragraph.
</p>
```

and your template is this:
XSLT Constructors

```
<xsl:template match="p[@xml:id='para_1']">
    <xsl:copy-of select="." />
</xsl:template>
```

then your output will be this:

```
<p xml:id="para_1">
    This is my first paragraph.
</p>
```

```
<xsl:copy>

• <xsl:copy> copies only the node itself, and not its descendants or attributes to the output tree.
• For instance, if your input is this:

```
<p xml:id="para_2">
    This is my second paragraph.
</p>
```

and your template is this:

```
<xsl:template match="p[@xml:id='para_2']">
    <xsl:copy />
</xsl:template>
```

then your output will be this:

```
<p/>
```

• Only the node (<p>) is copied. Its @xml:id attribute and text content are ignored.
Why copy constructors are useful: the "identity transform"

- You're probably wondering why copy constructors would be useful. After all, usually we're converting our XML into something else such as XHTML, so we can't just copy TEI nodes to the output document.
- However, we also use XSLT to edit, amend or improve existing TEI documents, transforming TEI into TEI, with minor changes.
- This is done through the so-called identity transform(ation). One version of it looks like this:

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

Can you figure out what this is doing?

An example use of the identity transform

- Imagine you have an old TEI file which has a lot of `<div0>` and `<div1>` elements in it:

```xml
[...]
<text>
  <body>
    <div0>
      <head>Title...</head>
      <div1><p>Some stuff...</p> [...]<div1>
      <div1><p>Some more stuff...</p> [...]<div1>
    </div0>
  </body>
</text>
[...]
```

- You have, quite sensibly, decided that both `<div0>` and `<div1>` should be turned into plain `<div>` elements. We can do this with an identity transform.
Changing `<div0>` and `<div1>` to `<div>` with an identity transform

This is the identity transform we need:

```xml
<xsl:template match="node()|@*" priority="-1">
  <xsl:copy>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="div0 | div1">
  <div>
    <xsl:apply-templates select="node()|@*"/>
  </div>
</xsl:template>
```

### XSL constructors: Task 2

Write an identity transform to manipulate the `ham.xml` file, making the following change:

The `<role>` element in the cast list should include the total number of lines spoken by that character. For instance:

```xml
<role xml:id="Claudius">Claudius</role>
```

becomes

```xml
<role xml:id="Claudius">Claudius (Lines: 528)</role>
```

Try to use what you've learned about XPath axes, functions and predicates, as well as variables, to solve this problem.
XSLT Constructors

XSL constructors: Task 2 answer

This is a complete working stylesheet. You'll find it at http://www.wwp.brown.edu/outreach/seminars/uvic_xslt_2013/demos/xslt_intro/mdh_line_count.xsl. Look at each line, and figure out what it's doing.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xpath-default-namespace="http://www.tei-c.org/ns/1.0">

<xsl:template match="node()|@*" priority="-1">
    <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
</xsl:template>

<xsl:template match="role">
    <xsl:copy>
        <xsl:apply-templates select="@* | node()"/>
        <xsl:variable name="currentRoleId" select="@xml:id"/>
        (Lines: <xsl:value-of select="count(()//l[parent::sp[contains(@who,$currentRoleId)])]"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

XSL constructors: Task 2 discussion

- Q: Why did we include xpath-default-namespace="http://www.tei-c.org/ns/1.0"?
  - A: So we don't have to include the TEI namespace or a prefix every time we reference a TEI element.
- Q: Why did we use a variable to store the @xml:id of the <role>?
  - A: Because if we didn't, it would be difficult to retrieve the @xml:id during the subsequent line count, because the line-count XPath places us in a different context (the context of the <l> elements we're counting).
• Q: Why did we use `contains(@who, $currentRoleID)` instead of `@who = $currentRoleID`?
• A: Because some lines are spoken by more than one speaker (Cornelius1 and Vlademar, for instance).