1

XML-based multimedia

XML is becoming a standard for the encoding of multimedia data. An important advantage of XML-based encodings is that standard XML tools, such as XSLT stylesheet-based processing, are available. Another advantage is that the interchange of data becomes more easy. Examples of XML-based media formats include SMIL, X3D, Speech ML, Voice XML.

In fact, to my mind, we should have a course on XML-based multimedia. Zhisheng Huang, who developed the STEP language (and its XML-encoding) which is described in the next section, has compiled a list of topics that you should know about XML-based multimedia.

XML-based multimedia

- *introduction*: Extensible Markup Language (XML). Extensibility and profiling of web-based multimedia. Streaming. Model of timing and synchronization of web-based multimedia.
- *processing XML*: XSLT stylesheets, Java-based XML Processing, SAX, DOM, Java XSL object APIs
- *SMIL*: (Synchronized Multimedia Integration Language) SMIL modules: animation, content control, layout, linking, media object, metainformation, timing, and profiles.
- *X3D*: (XML-based VRML) Extensible 3D: architecture and based components, profile reference, translation between VRML and X3D. X3D examples: case studies.
- *VHML*: (Virtual Human Markup Language) Virtual Human Markup Language, Humanoid, H-anim specification, Speech Synthesis Markup Language Specification for the Speech Interface Framework (Speech ML), Voice Extensible Markup Language (VoiceXML). Text to Speech Technology.
- *STEP*: Scripting Technology for Embodied Persona and XSTEP, the XML-encoding of STEP and its processing tools. Embodied agents and multimedia presentation: theory, model, and practice.

The course should emphasize practice and experience. An example assignment is the development of an information system, including multimedia data in the form

of images, 3D objects and audio recordings. The content should be organized according conceptual criteria, in an XML format to be designed by the student. Additional processing tools should then be written, using XSLT, to create a web site and to generate presentations in which the material is displayed from a particular perspective, for example a historic timeline, in one or more of the available presentation formats. See appendix **??** for a (more or less) concrete example.

As noted in the *research directions* of section **??**, XML comes with a set of related technologies. For processing XML we have XSLT, the transformation language which allows us to generate arbitrary text (including XML) from the information content of XML-encoded information. In the following, we will look at the use of XSLT to generate VRML-code from XML-encoded slides, using the collection of PROTOs developed in appendix **??**.

3D slides in XML

To refresh your memory, a slide set is a collection of slides that may contain lines of text and possibly 3D objects. Writing slides in VRML would be rather tedious. Besides, slides written in VRML could not be used in, say, HTML pages.

So the solution I came up with is to isolate particular pieces in a text as slides and to process these slides to create a presentation. In effect, both dynamic HTML-based and VRML-based presentations are supported. As a notation, an XML-based encoding seems to be the most natural, since it very close already to HTML, thus reducing the amount of processing needed to convert text containing slides to HTML. Now, how should the conversion to VRML take place. The answer is, simply, by using XSLT.

Let's first look at the XML-encoding of the example slides of appendix **??**.

slides in XML

```
<slideset>
<slide id="1">
<text>
<line>What about the slide format?</line>
<break/>
<line string="yeh, what about it">?</line>
</text>
<vrml>Sphere { radius 0.5 }</vrml>
</slide>
<slide id="2">
<vrml>Sphere { radius 0.5 }</vrml>
</slide>
</slideset>
```

One difference is that we introduced an *id* attribute in the *slide* tag, to allow for cross-referencing. These *id* attributes are, however, ignored in the conversion to VRML. Also, a *string* attribute has been introduced for the *line* tag. This is, however, just to illustrate how attributes are dealt with in processing XML files.

Before looking at the stylesheet used for the conversion to VRML, let me briefly say something about XSLT. The XSLT transformation language is a declarative language. It allows for processing an XML-encoded text by templates matching particular tags. In addition, the values of attributes of tags may be used when generating output.

The first part of our XSLT stylesheet looks as follows.

XSLT *stylesheet*

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

Apart from the obligatory declaration that the stylesheet itself is written in XML, there is also the indication that the file is a stylesheet written according to the rules and conventions that can be found in a file dating from 1999, as given in the url. Since we do want to generate VRML (and not XML), we need to indicate that our output method is *text*, to avoid having an XML header at the start of the output.

Now we are ready to define our first template.

slideset

```
<xsl:template match="/slideset">

... load (extern) proto(s)

DEF slides slideset {
slides [
<xsl:apply-templates/>
] # slides
}

... include timer or user interface

</xsl:template>
```

Everything that is not part of a tag containing the *xsl* prefix is literally copied to output. In this fragment, I have not included the full PROTO declarations nor the timer or user interface needed to traverse the slides. In the middle of the fragment we see the *xsl* tag *apply-templates*. This results in further processing the content that is contained between the *slideset* begin and end tag, using the template definitions given below.

The template for the *slide* tag is simple.

slide

```
<xsl:template match="*/slide">
slide { children [
<xsl:apply-templates/>
] }
</xsl:template>
```

You will recognize the structure, which is in agreement with the way we encoded slides in VRML, as illustrated in appendix **??**.

The template for the *text* is equally simple.

text

```
<xsl:template match="*/text">
text { lines [
<xsl:apply-templates/>
] }
</xsl:template>
```

For the *line* tag we need to do a bit more. Namely, we have to ask for the value of the *string* attribute, to obtain the complete result.

line

```
<xsl:template match="*/line">
line { string [ "<xsl:value-of select="@string"/>
<xsl:apply-templates/> " ] }
</xsl:template>
```

Note that, as mentioned above, the *string* attribute was just introduced to illustrate how to process attributes and is in itself superfluous. Actually, this way the *line* tag can be used as a closed tag, containing only the attribute and no contents, or an open tag with contents and possibly attributes.

Then, we are almost done.

etcetera

```
<xsl:template match="*/break">
line { string [ "<xsl:apply-templates/>" ] }
</xsl:template>

<xsl:template match="*/vrml">
<xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

We need to define a template for the *break* tag and a template for the *vrml* tag, which does nothing but copy what is beteen the *vrml* begin and end tag.

And that's it. Check the online version for the resulting slides obtained by processing this specification with the XSLT stylesheet given above.

You may have wondered why no mention was made of a DTD or *schema*. Simply, because we do not need such a thing when processing an XML-file using XSLT stylesheets.

When you want to use XSLT to process your own XML-encoded information, you will probably want to know more about XSLT. That is a good idea. Consult [XSLT] or one of the online tutorials.