Extreme Markup Languages 2002

Montréal, Québec
August 6-9, 2002

Comparing Transcoding Tools for Use with a Generic User Interface Format

Johan Plomp
VTT Electronics

Robbie Schaefer
C-LAB/Paderborn University

Wolfgang Mueller
C-LAB/Paderborn University

Abstract

This paper compares the use of three different approaches to transcoding of XML [Extensible Markup Language]-based user interface descriptions to other target formats. The source is the interface section of the XML-based markup language for user interfaces, UIML [User Interface Markup Language], which has been extended with a vocabulary for the description of generic user interfaces. Target formats used as examples for the comparison are HTML [Hypertext Markup Language], and VoiceXML. The compared means for transcoding are XSLT [Extensible Stylesheet Language Transformation], the UIML peers section with enhancements for transcoding, and RDL/TT [Rule Description Language for Tree Transformation], a Java-like transcoding language.

Comparing Transcoding Tools for Use with a Generic User Interface Format

Table of Contents

Comparing Transcoding Tools for Use with a Generic User Interface Format

Johan Plomp, Robbie Schaefer, and Wolfgang Mueller

§ 1 Introduction

The increasing use and growing variety of different devices to access information on the internet has induced both, the introduction of special purpose content presentation languages, like WAP [Wireless Application Protocol]/ WML [Wireless Markup Language] [1] and W3C [World Wide Web Consortium]/CompactHTML [2], as well as techniques for automatic conversion of traditional HTML content to these formats [3]. The abundant availability of mobile devices and increasing speed of mobile connections promise easy access to internet services. So far most of these services are maintained by a provider and contain mostly content. But in the near future these technologies will penetrate our daily environment and each private home may have dedicated servers that provide besides content also the ability to control appliances like entertainment and airconditioning systems.

The user interfaces of the home appliances may be provided in good old HTML, but there are several reasons why this format is not the most suitable one:

- HTML is mainly geared to presenting content. Forms provide an addition for gathering textual information from the user, but are limited to the implementation of the most simple user interfaces without advanced graphical objects or structure.
- HTML has been developed for sufficiently large displays with a pointing device; rendering on limited displays is cumbersome in most cases, mainly due to the use of positioning methods like frames and tables, and the use of image regions.
- HTML provides for graphical browsers in windowing environments and has no clues to support, e.g., speech applications.

As a consequence HTML has to be converted, i.e., transcoded, when it is to be used on limited devices. Transcoding from HTML does not even suffice when a renderer using alternative modalities like speech, is needed.

A search was done for suitable formats to describe a flexible user interface definition, supporting a variety of target formats including at least graphical and speech- driven interfaces. Our search was limited to XML-based formats and resulted in UIML [4][5] because of its versatility and extensibility as compared to e.g. XUL, which is limited to graphical user interfaces [15]. For our purposes, we developed a vocabulary (a set of classes or "widgets") to be used with UIML for the definition of generic user interfaces. While faced with the task of developing tools for the transformation of UIML to our chosen example target formats (HTML and VoiceXML [6]), we initially resorted to the transformation support intrinsic to UIML and represented by the so-called peers section.

As an alternative to (extended) UIML we have investigated RDL/TT[7] which was developed at the University of Paderborn to efficiently describe the definition of trancoding rules between two different XML-based formats. RDL/TT has been developed as a special purpose language for general transcoding between different formats especially dedicated to provide a (Java-oriented) programming language with functions for evaluating user and hardware profiles and executing DOM [Document Object Model] tree [8] manipulations.

In this paper, we compare the two previous approaches with the W3C standard XSLT [9]. XSLT is a template- oriented XML-based language for tree transformation. We investigate the three approaches in the context of User Interface transcoding or generation, respectively. In this context, we are interested to define User Interfaces by UIML descriptions, which are transcoded on demand to

VoiceXML, WML, or HTML. Our application scenario lies in the control of future home environments. By means of UIML, we specify a method to describe user interfaces of home appliances in order to control them via different interaction techniques. It requires that the UI [User Interface] description technique is truly application, target, and modality independent, and can be converted in real-time.

This paper is structured as follows. The next section gives a brief introduction to the basic structure of UIML. Section 3 gives the example UIML input and HTML and VoiceXML output on which the comparisons are based. Section 4 introduces the three different transcoding languages. Section 5 elaborates on the experiences with the transcoding systems and details the capabilities and shortcomings of the techniques. Section 6 concludes by discussing the results.

§ 2 A Generic UI Description Format

2.1 *UIML*

UIML stands for User Interface Markup Language. It is an initiative of Harmonia and aims to become a general platform independent standard for the formal definition of user interfaces. UIML is an XML application and thus inherits XML's portable and structural properties. UIML descriptions consist of

- an interface section, describing the structure, style and behaviour of the interface, and
- a peers section, describing how the used elements are translated to target format specific elements. The peers section also describes the API [Application Programming Interface] of the backend application, so that methods in this API can be invoked from within the code.

An interface section consists of parts, which can contain subparts. Properties are associated with the parts which can have an associated class. Behaviour is added by means of rules, which are invoked by events. Actions in a rule include setting property values, throwing events, or invoking (backend) methods. Thus, the interface reflects a simplified model of a general GUI [Graphical User Interface] developed with, e.g., Java. An example of the description of an UIML widget is shown below:

```
<part class="TextEntry" name="composer">
  <style>
    <property name="label">Composer:</property>
  </style>
  <behavior>
    <rule>
      <condition><event class="onChanged"/></condition>
      <action><call name="notifyValue">
          <param name="name">composer</param>
          <param name="value">
            <property name="value" part-name="composer"/>
          </param>
      </action>
    </rule>
  </behavior>
</part>
```

Since the resulting description is rather long, shorthands have been defined, providing simpler descriptions by using the class name as tag name for parts and allowing properties to be written as attributes. A simple transformation can produce the original format for processing. The above example would look in shorthand notation as follows:

```
<TextEntry name="composer" label="Composer: ">
  <behavior>
     ...(see above)
  </behavior>
</TextEntry>
```

UIML allows for switches to select the parts and properties that are required in a particular rendering. The UIML developers intended this to be used for describing interfaces geared towards different output formats. Thus, they would provide, e.g., an HTML section and a VXML section for generating HTML and VoiceXML output. For both sections they would use a dedicated vocabulary.

The peers section includes a description of API calls of the backend in the logic subsection and directions for the conversion of elements to the target format in the presentation subsection, of which there may be several. The peers section will be explained with the transcoding tools. API calls in the logic section contain the method name and the parameters. Optionally, a script may be added, in which case the API call may also be implemented locally.

```
     <logic>
     <d-component name="default" maps-to="#APPLICATION">
       <d-method name="notifyValue" maps-to="notifyValue"
return-type="Boolean">
         <d-param name="name" type="String"/>
       <d-param name="value" type="String"/>
       <d-param name="next" type="String"/>
     </d-method>
     </d-component>
     </logic>
```

2.2 A Vocabulary for Generic User Interfaces

We set out to define a set of widgets (or classes, i.e. the vocabulary) which would define as generic a user interface as possible. By selecting graphical and voice interfaces as our initial targets, we aimed to guarantee this wide support, since these interfaces differ quite a lot in their philosophy.

Graphical user interfaces adopt a "parallel" approach, i.e. the user may be presented with a number of selections simultaneously and may himself determine in what order he/she proceeds. Voice-based user interfaces inherently use a sequential approach, where the selections are mostly presented in a predefined order and the user provides the requested values in that order. Naturally barge-in and mixed initiative interfaces allow for more flexible input strategies, but most automatically generated user interfaces do not properly support this use.

For UIML interfaces, we defined a vocabulary, that is a widget set with their properties, to be used with UIML for the definition of generic user interfaces [10]. The widget set includes the following element types:

- Element - an abstract class defining common attributes like name, etc.
- Collection - a collection of elements which belong together semantically and should thus be rendered close to each other in time or space.
- Input elements - elements for the input of information, like text, numbers, lists, menus, and triggers (rendered, e.g., as buttons). These elements have been mostly defined due to their purpose, not due to their rendering.
- Output elements - elements for the output of information, like text, audio, or video.

The properties for the widgets include basic element specific attributes (value, label, etc.) and a number of additional properties for the support of rendering. These include for example next and previous properties for ordering the elements for sequential input, and positioning direction properties for graphical rendering. There are also properties like extended-label and vocabulary, which are used in voice applications.

§ 3 Transcoding Example

For comparing the different transcoding mechanisms we applied a simple example containing a trigger (button or link) and a list with three items. For simplicity only the trigger is provided with behaviour and the logic section in the peers (specifying the back-end API for the call) is omitted.

Taking this input, the desired output in HTML and VoiceXML was defined, and the transcoding rules were made such that the output was obtained, while observing a fair amount of generic functionality for later enhancement into a full transcoder.

3.1 *UIML Input*

The following example for transcoding input is written in shorthand UIML. For the XSLT transcoder, we used the proper UIML format (with DTD [Document Type Definition]) as input. That format is easily generated from the shorthand. The disadvantage of the shorthand is the need for a separate DTD for each application or used vocabulary.

```
     <uiml>
       <interface>
         <structure>
           <Frame name="Example" title="UIML Transcoding Test"
start="instrumentCollection">
             <InteractionCollection name="instrumentCollection"
start="instrumentList">

               <List name="instrumentList" label="Instruments"
extended-label="Select your
                instrument" value="piano" selections="4"
next="instrumentList">
                 <ListItem name="piano" label="Piano"
extended-label="piano"/>
                 <ListItem name="violin" label="Violin"
extended-label="violin"/>
                 <ListItem name="flute" label="Flute"
extended-label="flute"/>
               </List>

               <Trigger name="submitSelection" label="Submit"
vocabulary="submit">
                 <behavior>
                   <rule>
                     <condition><event class="onClick"/></condition>
                     <action><call name="notifyValue">
                         <param name="name">instrument</param>
                         <param name="value"><property name="value"
part-
                          name="instrumentList"/></param>
                         <param
name="next">FirstCollection</param></call>
                       </action>
                   </rule>
                 </behavior>
               </Trigger>

             </InteractionCollection>
           </Frame>
         </structure>

       </interface>
     </uiml>
```

3.2 *HTML Output*

The desired HTML output is rather straightforward and consists of a select tag with options and a button with an associated script. Since our environment will need some extra information for passing method names and parameters as well as a server URL [Uniform Resource Locator], some extra hidden inputs are added to hold that information. The script fills these fields before submitting the form.

```
     <html>
       <head>
         <title>UIML Transcoding Test</title>
       </head>
```

```
       <body>
         <form action="#HTTPSERVER" method="post"
name="instrumentCollection">
           <input name="params" type="hidden" value=""/>
           <input name="func" type="hidden" value="notifyValues"/>
           <input name="server" type="hidden" value="#SERVER"/>

           <label for="instrumentList">Instruments</label>
           <select name="instrumentList" size="4">
             <option value="piano" selected="true">Piano</option>
             <option value="violin">Violin</option>
             <option value="flute">Flute</option>
           </select>

           <input name="submitSelection"
onClick="submitSelection_onClick_handler()"
            type="button" value="Submit"/>
           <script type="text/javascript">
             function submitSelection_onClick_handler()
             {
               document.FirstCollection.func.value='notifyValue';
               document.FirstCollection.params.value='instrument,' +
                 document.FirstCollection.instrumentList.value;
               document.FirstCollection.submit();
             }
           </script>
         </form>
       </body>
     </html>
```

3.3 *VoiceXML Output*

The VoiceXML to be generated contains a link with an associated submit tag, implementing the behaviour, and a field with options for the list. Prompts are taken from the extended-label attributes of the corresponding elements. The control of the progress through the dialog needs some extra additions, like the form in the beginning jumping to the designated form (from the start attribute). Additional information for passing method names and parameters is kept in global variables.

```
     <vxml version="1.0">
       <var expr="'#SERVER'" name="server"/>
       <var expr="'notifyValues'" name="func"/>
       <var expr="''" name="params"/>
       <form>
         <block>
           <goto next="#instrumentCollection"/>
         </block>
       </form>
       <form id="instrumentCollection">
         <block>
           <goto nextitem="instrumentList"/>
         </block>

         <var expr="'piano'" name="temp_instrumentList"/>
         <field name="instrumentList">
           <prompt>Select your instrument</prompt>
           <prompt>Current selection is: <value
expr="temp_instrumentList"/></prompt>
           <filled>
            <assign expr="instrumentList" name="temp_instrumentList"/>
             <goto nextitem="instrumentList"/>
           </filled>
           <option value="piano">piano</option>
           <option value="violin">violin</option>
           <option value="flute">flute</option>
         </field>

         <link event="selected.submitSelection">
           <grammar>submit</grammar>
         </link>
         <catch event="selected.submitSelection">
```

```
          <assign expr="'notifyValue'" name="func"/>
          <var expr="'instrument,' + instrumentList" name="params"/>
         <submit caching="safe" expr="'#HTTPSERVER'" namelist="server
func params"/>
       </catch>
     </form>
   </vxml>
```

§ 4 Transcoding Languages

4.1 *XSLT*

The W3C standard XSL [eXtensible Stylesheet Language] [11] is a stylesheet language with strong influences from CSS [Cascading Style Sheet] [12]. The transformation language XSLT is part of XSL and designed for the purpose of transforming the XML-based input to textual, mostly XML-based, output. It can be used independently, or as part of an XSL stylesheet. At present, XSLT 1.0 has been accepted as a W3C standard and 2.0 is under evaluation and currently available as working draft since 30 April 2002.

The input of an XSLT program is a set of XML-based documents. The output can be XML, or plain text. With plain text output, an XSLT processor can generate languages different from XML such as LaTeX.

An XSLT definition, i.e., an XSL style sheet, defines a set of template rules which associate patterns with templates. Each rule consists of a matching pattern, optional mode and priority attributes, and a template. Matching pattern expressions are defined by a subset of the XPath language [13] and are evaluated with respect to a currently processed (matched) node or the root node. The matching process considers the node's name, attributes, location in the tree and position in the list and results in a set of nodes that can be used to provide parameters for the template or as a base for further matching. XPath supports the processing of node-sets and covers five additional basic types: booleans, numbers, strings, node sets, and tree fragments.

Processing usually starts at the root node. When a pattern is successfully matched, the pattern is associated with the template, the template (construction pattern) is recursively executed, mode is possibly changed, and matching is optionally continued from each matched node. For execution, XSLT provides variables and parameters which can be passed between template rules. For pattern processing, XSLT provides literals, constants, variables, and keys (for cross referencing) with conditions, list iterations, recursion, sorting, and numbering as control structures. For advanced processing, XSLT covers a powerful set of built-in string functions for creation, deletion, replacement, copying, and concatenation. Below an excerpt from the XSLT code for converting a list element to an HTML select tag.

```
    <xsl:template match="part">
      <xsl:param name="formname">form</xsl:param>
        <xsl:choose>
          <xsl:when test="@class='List'">
            <label for="{@name}">
              <xsl:value-of
select="style/property[@name='label']/text()"/>
            </label>
            <select name="{@name}">
              <xsl:attribute name="size">
                <xsl:value-of
select="style/property[@name='selections']/text()"/>
              </xsl:attribute>
              <xsl:apply-templates select="part"/>
              <xsl:apply-templates select="behavior/rule"/>
            </select>
          </xsl:when>
        </xsl:choose>
      </xsl:param>
    </xsl:template>
```

4.2 RDL/TT

RDL/TT was introduced to define tree transformations of XML-based documents. As it is a Java-oriented definition language RDL/TT is possible to be combined with Java, i.e., to apply native Java methods from within the RDL/TT rules.

In general, an RDL/TT definition is composed of the global settings and the definition of the transcoding rules. The global settings are for retrieving values from given hardware, software, and user profiles as properties, e.g., display width by

```
     resX  = getProperty("display.Width");
```

Retrieved values can be assigned to internal variables which can be used for parameters of transcoding functions. Additional variables can be declared for internal applications like counters etc. Variables are typed with respect to the expression on the right hand side: integer, boolean, string etc.

A transcoding rule definition consists of a search pattern followed by the transcoding rule description. A search pattern is an HTML or XML tag (e.g., <body>) which matches an HTML or XML element. Alternatively, a list of tags can be given as a shorthand definition, e.g., "<h1>, <h2>, <h3>, <h4>, <h5>". The transcoding descriptions given thereafter are then applied to each element in this list.

The functions given after the tag or tag list are executed on the DOM tree representation of the HTML or XML document. More precisely, they manipulate the subtree of the element given by the specified tag and they are applied on all nodes of the document tree that match the tag. Since attributes and text are represented as nodes in the DOM tree almost all of following functions can be applied to them as well. For manipulation functions, RDL supports deletion, insertion, copying, replacement, renaming of tags and attributes as well as replacement of content by a hyperlink and to outsource the content into a separate document.

Syntactically, RDL/TT rule definitions are based on Java, also inheriting concepts from XSL and XPath. An RDL statement can be either an assignment to a variable or a function call. RDL provides a built-in data type for pathnames denoting object locations in the given tree. Paths are specified by separating their components by the dot operator. Components are HTML elements or attribute identifiers or nodes relative to the current node: parent, selection of child elements, or content. For child selection aggregates like "*" for all children or numbers can be employed.

For basic control, RDL provides two control structures: "if/else"-branches and loops, where loops iterate over path patterns. The latter concept is inherited from UNIX C-Shell programming. For example, "foreach var in this.p" iterates over every paragraph which is a child of the current node and variable "var" takes each paragraph as value.

Combining those concepts, RDL definitions provide means to perform tree transformations dependent on properties from user and hardware profiles. Because of powerful built-in functions and its Java-based procedural language description, RDL typically provides very short transcoding definitions. Like the previous XSLT example the following code excerpt in RDL/TT shows the conversion from a list to an HTML select tag.

```
     <List>:
       rename(this.attribute.selections, "size");
       insertNodeBefore(this, "label");
       insertAttribute(this.parent.label, "for",
valueOf(this.attribute.name));
       remove(this.attribute.name);
       if(exists(this.attribute.label))
       {
         insertTextWithin(this.parent.label,
valueOf(this.attribute.label));
       }
       rename(this, "select");
```

```
<ListItem>:
  rename(this.attribute.name, "value");
  insertTextWithin(this, valueOf(this.attribute.label));
  keepAttribute(this.attribute.label);
  rename(this, "option");
```

4.3 *Extending UIML Peers Sections*

The peers presentation section of UIML includes mappings of classes (and components) and their properties to target format constructs. The logic section is also part of the peers section (see chapter 2), but describes mainly the backend interface. A presentation section usually has a name, which allows different presentation sections to be provided for different target formats. For example, there may be a presentation section for VoiceXML and for HTML.

Mappings of classes and their properties are not necessarily restricted to XML-based formats but may also be mapped to, e.g., Java constructs. In most cases one needs a dedicated renderer for the conversion of the user interface description in the interface section to the target format. Harmonia provides prototypes of such renderers on their web pages for interfaces specified in their proposed vocabularies (www.harmonia.com).

The mapping information included in the peers section is sufficient for mapping the UI description to an XML-based output format if the structure of the output format is similar to that of the input format. Mappings include mappings to tags, attributes, and text nodes, as well as definitions of set and get methods for properties (as scripts) and the events possible for a class.

An example of a basic mapping of, e.g., a text output to HTML could be:

```
<d-class name="TextOutput" maps-type="tag" maps-to="p">
 <d-property name="value" maps-type="attribute" maps-to="PCDATA"/>
</d-class>
```

For the purpose of transcoding UI descriptions using our generic vocabulary to XML-based output formats, we attempted to enhance the peers section with features to support more versatile transcoding functionality. We did not attempt to support the transcoding of behaviour, since that would have needed too big changes to the peers section. Our enhancements have mostly been added to existing attributes, some attributes were further added and the <d-param> tags under the <d-properties> (used for giving default values to the properties) have been enhanced as well. We see the following enhancements:

- The maps-to property of <d-class> and <d-property> tags may include a list of nodes (e.g. html/head/title), allowing the generation of a small tree from one instance.
- A maps-path attribute was added to <d-class> and <d-property> tags, allowing a path where the node should be added (e.g., "../form")
- A maps-presence attribute was added to <d-property> tags, defining whether or not the property should be rendered in the target format (omit/optional/required)
- There is a possibility to map properties or <d-param>'s to trees of tags with attributes. For this purpose the <d-tag> and <d-tag-attribute> elements were added. A arbitrary tree of nodes can be added to the output based on a property.
- The "#name" variable may be used to refer to the name of the part, where "#value" may be used to refer to the value of the property. Arbitrary variables may be defined externally (like "#APPLICATION" in one of the previous examples) and will be replaced by their value before processing.
- When names of <d-param> tags include a dot, the <d-param> will be rendered as an attribute of the enclosing property.

Of the above enhancements, the mapping to a tree (with tags or attributes referring to the value of a property) is a bit restrictive. This kind of mapping may result in a situation, where property values cannot be accessed by the renderer, and thus property set and get actions (by referring to the property value) may not be implemented properly.

§ 5 Comparison Of The Transcoding Tools

The considered means for transcoding were all able to generate satisfactory output for our application, i.e., for generation of HTML and VoiceXML. With a few minor differences, all generated results were similar. Hereby, we must note that the UIML Peers section used a dedicated transcoder (written in Java) to add the behaviour part to the output. The following subsections elaborate on some features of the transcoding mechanisms and the last subsection gives an overview of the observed features in a table.

5.1 *XSLT*

XSLT proved to be quite powerful with respect to functionality, not only for generating XML output, but also for generating the HTML JavaScript section from the call description.

The philosophy used by XSLT is that of templates, filled with information from the source. Information in the XML source can be accessed via a powerful path mechanism, but placement of the templates will always happen at the current location, i.e., one can only append to the output and not insert. This causes problems when the input is very differently structured from the output and may require some creativity in writing the XSLT code.

The support of variables in XSLT is very poor; only a kind of constants. The language would be more versatile with such support. Also the use of variables to define search patterns or names or attributes of newly created elements would be appreciated. Nevertheless, parameter passing to templates is supported.

In general, due to the principles of template definitions, XSLT leads to larger transformation descriptions so that for longer rule sets authoring tools are really needed for the management of these descriptions.

The aforementioned drawbacks caused a few minor workarounds in the XSLT code for the examples. Otherwise the generation of the examples was rather straightforward.

For XSLT compilation we used the Xalan implementation by Apache (xml.apache.org/xalan-j). With this implementation we recognized that the total processing speed, including the construction of the transcoder itself, was rather low. However, the actual processing time was only a fraction of the total time and the speed still fulfills our requirements within the given scope.

5.2 *RDL/TT*

In contrast to the other transcoding languages, RDL/TT has been designed as a Java-oriented programming language for general purpose transcoding descriptions and provides more features in the direction of a programming language, i.e., variables, method calls etc. RDL/TT thus leads to more concise definitions than the other two XML- based languages. Therefore, RDL/TT also combines seamlessly with Java programs. Because of these principles, more complex functions written in Java can be integrated and then executed within the transcoding rules. So, in addition to a couple of built-in functions, one can easily define libraries for specific applications.

Another main difference is the matching. RDL/TT basically provides only simple matching of tags. When matching a tag, the values of its individual subelements, like attributes, can be evaluated within conditions. This evaluation can trigger the execution of different commands on parts of the DOM tree, which can be freely selected with the path mechanism.

Finally, when comparing the runtime it has to be noted here that the current RDL/TT implementation is a prototype implementation and has not been heavily tuned for high performance yet.

5.3 *UIML Peers Section Extension*

Compared to the previous two transcoding means, the UIML peers section provides much less functionality. In spite of the fact that it was developed to be used with UIML, it does not come close to the transcoding performance of RDL/TT and XSLT, except for the total processing time which is significantly better than both other candidates. This is particularly obvious through the need for a dedicated Java-based module for adding the behaviour (i.e., this was not possible with the peers section format).

The UIML peers section follows the philosophy that the UIML input is processed in order, but the (XML) output can be placed at random. This is just in contrast to XSLT, and therefore, e.g., search templates are very simple, but path mechanisms are used for the placement of the output. The advantage can be seen in the processing speed.

The lack of conditional processing features, loops, and access of other elements besides the one being processed, causes the current UIML peers section to be inadequate for full-fledged transcoding - in spite of the enhancements - and dedicated transcoders must be written to achieve the desired transcoding behaviour. In many cases the target format for the UI may not be XML-, or even text-based. This is not necessarily a problem for UIML and constitutes one of the strengths of the approach. The UIML peers section was never intended to provide a full transcoding mechanism, but such a feature would certainly aid the adoption of UIML as an accepted standard for UI description.

Enhancements to the peers section were made such, that the original peers section format was changed as little as possible. Unfortunately this has resulted in a rather ad-hoc format, which does not allow for easy programming. If this approach is to be taken seriously, a better format for the peers section must be designed.

5.4 *Overview of Transcoder Features*

Table 1 Basic Language Feature Comparison

Feature	XSLT	RDL/TT	Ext. UIML Peers
Search pattern	Powerful pattern. No parameters allowed	Tags; Powerful expressions within rules	Exact match of classes and properties
Template	Any legal XML construct	No XML templates but context sensitive rules	Any legal XML construct
Control Flow in Specification	(Explicit continuation in) Recursive application of templates	Tree-oriented evaluation	Tree-oriented evaluation
Tree Modification/Creation	Generation of target tree	Modification of source tree	Generation of target tree and operations on target tree
Node creation/modification	Nodes and attributes can be dynamically constructed. Parameters cannot be used to define Node and Attribute names	Dynamic Construction of any DOM-Part. Parameters usable to apply rule depending on context	One-to-one creation easy, multiple nodes or attributes are complex
Result tree placement	Added to current node by default	Free (path mechanism)	Free (path mechanism)

Feature	XSLT	RDL/TT	Ext. UIML Peers
Access of node properties	Node name and attributes can easily be accessed and used	Any part of DOM can be accessed and used	Name of class and value of current property can easily be accessed
Access of secondary nodes	Path mechanismn and large number of built-in access functions	Path mechanismn and built-in access functions	Not possible
Loop processing	over node sets	over node sets	Not supported
Conditional processing	If and switch (choose) statements	If/Else statements	Not supported
Variables	Only constant definitions	Free typed Variables	Not supported
External values	Not supported	Supported	Supported
Data types	Boolean, number, text, nodelist, document fragment	Boolean, Integer, Float, String, Nodes (Subtrees)	Only character strings
Output format	XML & plain text	XML & plain text	XML, any output through dedicated renderers
Multiple document structure	Supported	Supported	Supported
Miscellaneous functions	Sorting, numbering, misc. functions	Several Built-in functions. Extensible with Java libraries. Pre selection of rulesets	UIML specific

Table 2 Performance with given Example

Feature	XSLT	RDL/TT	Ext. UIML Peers
Ease of definition HTML	Good	Good	Good
Ease of definition VXML	Good	Good	Fair
Addition of behavior [1]	Good	With basic functionset only, on Textlevel	Insufficient; supported through dedicated transcoders
Speed; HTML generation [2]	1480 ms (280 ms)	720 ms (590 ms)	380 ms (320 ms)
Speed; VXML generation [3]	1440 ms (240 ms)	690 ms (660 ms)	390 ms (310 ms)
Lines of code; HTML	122	85	73
Lines of code; VXML	121	81	123

§ 6 Discussion

In this paper, we compared three methods based on an example of transcoding an UIML-based user interface description to two inherently different XML-based output formats, i.e., XHTML and VoiceXML. All three methods were capable of generating satisfactory output in reasonable time.

One main difference can be identified with respect to the description style of the transcoding rules. RDL/TT, in contrast to the two other means, is a programming language with variables and methods especially dedicated and designed for tree-based transcoding operations, whereas XSLT and UIML peers provide XML-based means. All three methods are based on the principles of pattern matching. RDL/TT, however, uses a very simple mechanism by matching only the tag and selecting other properties by conditional rules.

Another difference lies in the methodologies and capabilities of the transcoding process, source tree evaluation, and the construction of the target tree. XSLT uses a simple append-to-file approach while allowing versatile access to the source. RDL/TT operates on the source tree and basically allows the use of hammer and chisel to shape the tree at wish. The UIML enhanced peers section again allows building a tree by adding branches in freely chosen places, but strictly follows the input structure for processing order. It appears that either on the source side, or on the target side, access should be completely free in order to achieve a sufficiently versatile transcoding mechanism. It is interesting to see that our three approaches differ in where this freedom was chosen, in all cases a trade-off was made between the versatility of the search-pattern and the expressiveness of the paths in the placement mechanism. Allowing both might be possible, but may also cause the language to produce too complex rules with unpredictable behaviour.

Construction of more than one target document from a single input was supported by all. This is a basic prerequisite for transcoding application for limited mobile devices since large content has to be split into small parts of different types (e.g., voice and text) which then can be easier captured. This was a major drawback of XSLT Version 1. Nevertheless, XSLT Version 2 now provides multi-document structures.

With respect to the supported output formats, the UIML peers section is the most versatile. While in this comparison only XML formatted output is used enhanced with scripts, UIML would be capable of rendering the UI also in, e.g., Java constructs with a suitable renderer. RDL/TT and XSLT both render only textual output.

Variables were not absolutely required for our definitions within the scope of our example, Nevertheless, in general they do help in creating the transcoding rules. Thus, we would recommend the enhancement of XSLT with real variables since the current concept of variables seems to be too limited. In our experiments, we saw that conditional and loop processing is a must. The lack of support in the UIML peers section was not yet refraining us from completing this example, but in our other studies, it has been a serious bottleneck.

In the UIML extension and in RDL/TT, the user explicitly defines operations with respect to the currently matching node in the DOM tree representation. XSLT and UIML (after enhancement) used templates, RDL/TT used manipulation commands. In application, XSLT is somehow different from the others since the user does not necessarily have to have the mental model of the tree representation since XSLT is based on principles of matching and replacing templates recursively applied. This can be seen as a disadvantage since XSLT results are only locally seen. It has to be noted here that editing of larger XSLT definitions without authoring tools is a little bit too cumbersome. In turn, the Java-based format of RDL/TT can on the one hand also be seen as a disadvantage, since for novice programmers it is more difficult to see the resulting output by just looking at the code. On the other hand, it is a great advantage for programmers with Java programming skills since it provides a couple of powerful features from the Java programming language.

The speed of the transcoding tools seems to relate to the amount of searches that need to be done in the code. The chosen examples are actually too small to draw a significant general conclusion. It can be expected that when scaling up the input, the searches will consume under some specific conditions significantly more time. The most extensive searches are done by XSLT's template matching mechanism when using backward searches, so that this can be expected to be a major bottleneck. So, large sets of highly context sensitive replacements with only little differences in the matching template might result in significantly higher differences in run time. This basically means that XSLT, under some circumstances, is less suited for defining a general transcoding of large amount of data between two different formats, which was the motivation behind the development of RDL/TT. Nevertheless, this is a first observation and we have to verify it by more exhaustive tests. In the context of the performance comparison it has to be noted here that the tools for UIML peers extensions and RDL/TT are currently only available as software prototypes in contrast to the applied XSLT transcoder.

For general conclusion, it can be noted that all three languages provide more or less sufficient means for the chosen example. Main differences seem to be more heavily due to the individual application.

XSLT provides a well defined tool for the general transformation of XML-based documents to textual output. Producing non-textual output is not supported. The template matching approach allowing for this genericity may cause performance problems with large documents.

The extended UIML peers section is on its best when variations of interactive user interfaces have to be generated, i.e., when the UI is the primary function and content is of secondary priority such as it is in user interface generation for home appliances. This is due to the fact that it is nicely embedded into UIML and therefore dedicated to device independent user interface description. Furthermore, the extension provides nice transcoding support when the source description (in the interface section of UIML) is sufficiently close and structurally similar to the target format. The future of the UIML peers section used as a transcoding mechanism should be reconsidered for further extensions. It needs to be enhanced with at least conditional processing and a more versatile access to secondary node information. In addition, the format should be reconsidered to allow easier programming. Adding behaviour transcoding also requires more powerful text processing tools.

RDL/TT in the presented form seems to be best suited for the description of general transcoding rules between two different formats for which purpose it was developed. Main advantages are, that the concept of hardware and user profile information processing is seamlessly integrated in RDL/TT. So general transcoding can be individually be tailored to the given user preferences and hardware limitations imposed by mobile devices. Since attributes of elements - like the number of characters in the content of a node - can be easily retrieved from within the language, a most context sensitive and profile specific transcoding can be specified. In this application, RDL/TT does not only have short specifications but also the underlying computation principles seem to be more efficient for highly context sensitive applications. In that direction, RDL/TT has already specified an additional extension so that the selection of different transcoding rules for one target format with respect to the individual profiles is better supported by specific language constructs [14]. Also advantageous is the possibility to extend the built-in functionality with dedicated libraries of Java methods.

The transcoding methods compared here were all applied to transcode a generic user interface format to two specific XML-based target formats. The conclusions presented must be seen in this context, and it is difficult to generalise them to a broader scope without additional tests. In summary, we may conclude that all three methods were able to perform the required transformations within a reasonable amount of time. XSLT is the best defined transcoding method with a clear philosophy, the UIML peers section finds its strength in its versatility of output formats, while RDL/TT wins in code efficiency and transcoding power when combined with Java. The reason for choosing one of these options may therefore be based on the purpose. When the rendering format should be something else than XML or text based, the UIML peers section is the only choice. When the transcoders must be

designed and maintained and accessed by a large group of people, a well documented and standardised solution like XSLT is the obvious choice. When efficiency and transcoding power are of prime importance, RDL/TT should be considered. It has to be noted here again that RDL/TT was not developed for small scale transcoding so that the given example did not provide a perfect application scenario for that language.

With respect to the unenhanced UIML peers section, we can conclude that it would only suffice for the most simple transcoding tasks and is definitely not sufficient for transcoding the generic UI used as a source format in this comparison. In fact, the usefulness of the presentation subsection is doubtful, since renderers should be made for each target format separately anyway.

Notes

1. That is; the addition of the event handling structure with call to the backend API
2. Without the time for opening files. The time within brackets is needed for transcoding only (i.e. after the transcoder is created). The measurements were done on a 600 MHz Pentium3 computer under Windows NT and using Sun's JDK 1.3.
3. Without the time for opening files. The time within brackets is needed for transcoding only (i.e. after the transcoder is created). The measurements were done on a 600 MHz Pentium3 computer under Windows NT and using Sun's JDK 1.3.

Acknowledgements

The work reported in this paper has been done in the ITEA-VHE (Middleware for Virtual Home Environments) project and was supported by ITEA and local funding organisations in Europe.

Bibliography

[1] WAP Forum. Wireless Markup Language Specification Version 2.0, June 2001. http://www.wapforum.org/tech/documents/WAP-238-WML-20010626-p.pdf

[10] Johan Plomp, Oscar Mayora-Ibarra & Heli Yli-Nikkola, Graphical and Speech-driven User Interface Generation from a single Source Format, proc. of the first annual VoiceXML Forum Users Group Meeting (AVIOS 2001), San Jose, April, 2001

[11] Sharon Adler et al. Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation. World Wide Web Consortium, October 2001. http://www.w3.org/TR/xsl/

[12] Bert Bos, Hakon Wium Lie, Chris Lilley & Ian Jacobs, Cascading Style Sheets, level 2 CSS2 Specification, W3C Recommendation. World Wide Web Consortium, May 1998. http://www.w3.org/TR/REC-CSS2/

[13] Anders Berglund et. al., XML Path Language (XPath) Version 2.0, W3C Working Draft. World Wide Web Consortium, April 2002. http://www.w3.org/TR/xpath

[14] Robbie Schaefer, Andreas Dangberg, Wolfgang Mueller, Fuzzy Rules for HTML Transcoding, proc. of the 35th Hawaii International Conference on System Sciences (HICSS-35), Hawaii, January 2002

[15] Tao Cheng, XUL - Creating Localizable XML GUI, proc. Of the 15th International Unicode Conference, San Jose, California August1999

[2] Tomihisa Kamada. Compact HTML for Small Information Appliances, W3CNote. World Wide Web Consortium, February 1998. http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/

[3] Korva, Jari, Johan Plomp, Petri Määttä & Maija Metso, On-line service adaptation for mobile and fixed terminal devices, proc. of the 2nd Int. Conf. on Mobile Data Management (MDM 2001), Hong Kong, 2001.

[4] Harmonia, Inc. User Interface Markup Language (UIML) Draft Specification. http://www.uiml.org/specs/docs/uiml20-17Jan00.pdf

[5] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E., UIML: an appliance-independent XML user interface language, Computer Networks 31, Elsevier Science, 1999, pp. 1695-1708.

[6] Scott McGlashan et al. Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Working Draft. World Wide Web Consortium, Oktober 2001. http://www.w3.org/TR/2001/WD-voicexml20-20011023/

[7] Robbie Schaefer, Andreas Dangberg, Wolfgang Mueller, A Generic Language for the Transcoding of HTML Pages, C-LAB Report 31/2001, Paderborn, Germany, February 2001. (In German)

[8] Arnaud Le Hors et al. Document Object Model (DOM) Level 3 Core Specification. Version 1.0, W3C Working Draft. World Wide Web Consortium, September 2001. http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/

[9] Michael Kay. XSL Transformations (XSLT) Version 2.0, W3C Working Draft. World Wide Web Consortium, April 2002. http://www.w3.org/TR/xslt20/

The Authors

Johan Plomp
VTT Electronics
Kaitoväylä 1
PL 1100
90571
Oulu
Finland
tel: (08)551 2305
fax: (08)551 2320
Johan.Plomp@vtt.fi
www.ele.vtt.fi

Johan Plomp received his Master's degree in Electrical Engineering from the University of Twente, the Netherlands in 1990. Between 1989 and 1995 he held several teaching, research and managerial positions at the University of Oulu, Finland, where he completed his Licentiate of Technology degree in 1996. Since that same year, he has been affiliated with VTT, the Technical Research Centre of Finland where he currently holds a position of senior researcher and vice group manager in the field of Advanced Interactive Systems.

Johan Plomp's research interests have included over the years e.g. VLSI chip design for computer vision systems, software platforms for image analysis, automatic help for PLC-logic based systems and telecommunication software, ubiquitous computing platforms and new human-system interaction technologies. His current interests include the development of a generic user interface description technology to support highly adaptive multimodal interaction.

Robbie Schaefer
C-LAB/Paderborn University
Fürstenallee 11
33102
Paderborn
Germany
tel: +49 5251 60-6107
fax: +49 5251 60-6065
robbie@c-lab.de
www.c-lab.de

Robbie Schaefer received his diploma in Computer science from Paderborn University in 2001 and works since then in the group of Visual Interactive Systems at C-LAB, a joint institute of Paderborn University and Siemens Business Services at Paderborn, Germany.

His current interests are the adaptability of user interfaces in general, and transcoding of web content in particular, where he authored two papers.

Wolfgang Mueller
C-LAB/Paderborn University
Füürstenallee 11
33102
Paderborn
Germany
tel: +49 5251 60-6134
fax: +49 5251 60-6065
wolfgang@c-lab.de
www.c-lab.de

Dr. Wolfgang Mueller received his diploma in Computer science from Paderborn University in 1989. In 1997 he received his doctorial degree at Paderborn University. Since then Dr. Mueller is heading the group of Visual Interactive Systems at C-LAB, a joint institute of Paderborn University and Siemens Business Services at Paderborn, Germany.

There Dr. Mueller is in charge of the coordination of multiple European and German projects. He authored more than 70 papers mainly in the areas of systems design languages, methodologies, and collaborative design environments. He is member of GI and ACM and works in programme and executive committess of various conferences and workshops like DATE, FDL, and Euromedia. Dr Mueller is a founder member of the VHE-Middleware consortium. His present interests are methodology and design of distributed systems, formal specification, and interactive graphical user interfaces.

Extreme Markup Languages 2002
Montréal, Québec, August 6-9, 2002
This paper was formatted from XML source via XSL
Mulberry Technologies, Inc., August 2002