XML Data Representation for UIML

Introduction

The goal of this document is to present some general ideas and approaches on how to extend UIML to support a complex data representation to be used in creating advanced user interface widgets. The widgets being considered in the examples are JTree and JTable, both of which have a clearly separate Model of data and a complex data representation. The ultimate goal is to have a single data representation mechanism that allows us to do the following:

- represent any data structure, including tables, trees, graphs, structures and recursive structures;
- represent data as part of a UIML Presentation section as to describe the data used to instantiate a user interface widget;
- represent data as an external source of information that can be loaded into a running program;
- define ways to set attributes and parameters for the different sub-parts of the advanced widgets even when there is no data in them (e.g. set the font of row 1 of a table even before there is any data loaded).

Approach

Our general approach has been to use existing technologies and notations when possible. To that end, two particular ideas have influenced our thinking. First, we are trying to use XPath as much as possible as a way to express references into portions of an XML representation. This would allow us a way to represent event listeners and properties for sub-parts of the widgets of interest. Second, we have borrowed ideas from Apple's Property List[1] as a way to capture a generic XML representation. We have modified how property lists are built to suit our purposes, but the general idea of a single data representation is very much in the spirit of our solution.

Our solution is not complete however. There are still a number of questions that depending on how they are answered, it might require us to change our approach. So, consider these as ideas, not as complete solutions. Any feedback, suggestions, praises, or complaints are welcome.

Example

For the following discussion, we will use a JTable example that is included in the Harmonia distribution of UIML. Note that this example creates a JTable inside of a JScrollPane inside of a JFrame. The example has four rows of data in the table with the first row intended as the headings and the next three rows containing values. The values are defined as part of the "content" property of the part "table".

[1] See http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/index.html

Our goal, restated more precisely in terms of this example, would be to extend (or replace) this <model> representation with a new representation such that:

- It will be independent of the type of model represented. In this example, the tags used are specific to the JTable example.
- It will allow us to set listeners in particular cells. For example, consider putting a listener in the heading row to sort the data by that column when clicked on it.
- It will allow us to load the data from an external file without restricting our ability to set listeners and other properties. Thus, we need the ability to express within the UIML program a path to a sub-part of the model without having the model completely expressed ahead of time.

Example: JTable example from Harmonia's UIML Distribution

```
<?xml version="1.0"?>
<uiml>
   <interface id="TableExample">
      <structure>
         <part id="frame" class="JFrame">
            <part id="scroll" class="JScrollPane">
               <part id="table" class = "JTable"/>
            </part>
         </part>
      </structure>

      <style>
         <property part-name="frame" name="title">Example</property>
         <property part-name="frame" name="size">400, 400</property>
         <property part-name="table" name="content">
            <model>
               <row>Color, Size, Material</row>
               <row>red, large, cotton</row>
               <row>blue, medium, nylon</row>
               <row>green, small, rayon</row>
            </model>
         </property>
      </style>
   </interface>
   <peers>
      <presentation base="Java_1.4_Harmonia_1.0"/>
   </peers>
</uiml>
```

The example above shows one approach that has been used with UIML to populate a JTable. The problem with this approach is that the language used (`<model>`, `<row>`) is defined by the JTable component. We would like to create a way to populate any user interface component with a language that is defined by UIML and not by the vocabularies or a particular rederer.

A second approach used in UIML is the `<constant>` notation used to expressed constant values. This is also limited in that it allows to represent single values.

Proposed solution

In general, a data model is defined by the following grammar (a DTD will be provided later).

```
model ::= <model> data-rep </model>
data-rep ::= {struct | data}*
struct ::= <struct [structtype=stype]> data-rep </struct>
data ::= <data datatype=dtype value=any-value/>
dtype ::= string | number | date | boolean
stype ::= array | list | graph | hash
```

As a way to present the notation, the model from the previous example is shown below. Note that this represents an array of strings, similar to what the example above showed. It would be up to the Swing component to interpret the commas as cell separators.

Example Model 1: first cut at a model representation

```
<model>
   <struct structtype=”array”>
      <data datatype=”string” value=”Color, Size, Material”/>
      <data datatype=”string” value=”red, large, cotton”/>
      <data datatype=”string” value=”blue, medium, nylon”/>
      <data datatype=”string” value=”green, small, rayon”/>
   </struct>
</model>
```

The datatype attribute might allow UIML to do some amount of error checking on data representations for complex widgets.

This next representation of the same example divides the strings into separate cells by defining an array of arrays.

Example Model 2: a matrix representation for the JTable example

```
<model>
   <struct structtype=”array”>
      <struct structtype=”array”>
         <data datatype=”string” value=”Color”/>
         <data datatype=”string” value=”Size”/>
         <data datatype=”string” value=”Material”/>
      </struct>

      <struct structtype=”array”>
         <data datatype=”string” value=”red”/>
         <data datatype=”string” value=”large”/>
         <data datatype=”string” value=”cotton”/>
      </struct>

      <struct structtype=”array”>
         <data datatype=”string” value=”blue”/>
         <data datatype=”string” value=”medium”/>
         <data datatype=”string” value=”nylon”/>
      </struct>

      <struct structtype=”array”>
         <data datatype=”string” value=”green”/>
         <data datatype=”string” value=”small”/>
         <data datatype=”string” value=”rayon”/>
      </struct>
   </struct>
</model>
```

Observations

- Note that the `structtype` used in the `<struct>` tag is really optional. The type specified there is intended to signal the processing code how to treat and access these values. From the UIML code, these are just attributes that have really no particular meaning or provide no particular added value.

- Also note the role of the three inner `<struct>` is to wrap each row as a single entry. This allows the resulting structure to represent a 4 row x 3 column matrix.

- We have decided to represent data inside of tags (as attributes) because it allows us the flexibility to set the `value` attribute via program calls. For example, below we show how the title for column one can be set dynamically by calling the routine `getHeadingName(cell)`.

Example: Setting value dynamically

```
<model>
   <struct structtype=”array”>
      <struct structtype=”array”>
         <data datatype=”string”>
            <call name="getHeadingName"/>
               <param name="cell">1</param>
            </call>
         </data>
         <data datatype=”string” value=”Size”/>
         <data datatype=”string” value=”Material”/>
      </struct>
      …
```

- A tree representation becomes a series of struct/data. For example, a simple tree representation showing a root node with a value of B and two children nodes with values A and C.

Example: Tree Representation

```
<model>
   <struct structtype=”tree”>
      <data datatype="string" value=”A”/>
      <struct structtype=”tree”>
         <data datatype="string" value=”B”/>
      </struct>
      <struct structtype=”tree”>
         <data datatype="string" value=”C”/>
      </struct>
   </struct>
</model
```

- A more interesting example of a tree representation is shown below. This is based on one of the JTree examples form the UIML distribution.

Example: Tree representation from Harmonia’s JTree example

```
<model>
   <struct structtype=”tree”>
      <data datatype="string" value=”A”/>
      <struct structtype=”tree”>
         <data datatype="string" value=”B”/>
         <struct structtype=”tree”>
            <data datatype="string" value=”C”/>
            <struct structtype=”tree”>
               <data datatype="string" value=”D”/>
            </struct>
            <struct structtype=”tree”>
               <data datatype="string" value=”E”/>
               <data datatype="string" value=”F”/>
            </struct>
        </struct>
   </struct>
   <struct structtype=”tree”>
      <data datatype="string" value=”G”/>
      <struct structtype=”tree”>
         <data datatype="string" value=”H”/>
      </struct>
   </struct>
</model>
```

Problems or Issues to address

Tree, array, and list representations are possible with this notation. What issues need to be solved before we can adopt this representation? The following sub-sections include some issues that still need to be addressed.

Add ID to <data> representation?

- Do we need an id attribute for the data representation? Clearly if we need to make a reference to a particular data item, then a unique id is required. But often times, we don’t need direct access to a data item. Should the id be optional?

Parts and sub-parts

- Related to the need for an id (see above), is the need to be able to make references to different parts of a UIML document. For example, we need to refer to a presentation part in the style section or behavior section. To that end, we need a way to refer unequivocably to the part. However, with complex UI widgets, we now have the need to refer to sub-parts of widgets. For example, how do we specify style and behavior for cell 1, 1 in the JTable example above? This is an issue that will require an extension to UIML as references to parts can appear through a UIML document.

- A solution is to use a part and sub-part representation. This will allow us to make references that are local without requiring unique global names in the UIML document for all sub parts. Each part can provide local names for its parts. An example is shown below.

```
<part id="check1" anchor=”name” sub-part="node" ... >
```

- The advantage of this is that it allows us to use XPath expressions as sub-part references. More on XPath below. The disadvantage is that it will require a modification to UIML to support part/sub-part references to elements.

Using XPath

We are considering using XPath (or a subset of XPath) as a way to make references to sub-parts in complex widgets or in any other data representation. XPath is defined by W3C as an expression language over a data model of XML documents. In this section we show some examples on how we are considering using XPath in UIML. A good tutorial on XPath can be found at http://www.w3schools.com/xpath/default.asp. Some examples are presented in the appendices.

One advantage of XPath is that it allows us to express a reference to an element in the XML document without requring that element to be present. For example, consider creating a JTable that will have a formula at the end of column 1 to total all the numbers read in. If we don't know how many rows there are in the data, we can't specify a formula for that particular cell. However, if we define the last cell with some tag (e.g. id=”last”), then we can make an XPath reference to a particular data item with tag id=”last”. This reference will allow us to then define properties for that cell (or the one that follows it), like formulas or formating.

Unique IDs

When parts are defined in terms of anchor and sub-parts, they don't automatically get a unique id. Is this a problem? That is, how is a unique id generated if we use xpath to create subparts? Consider the following two examples.

- In this example, a JCheckbox is added to all top level `<struct>` subparts of the JTree data model. Note that the `/struct` is an XPath expression that says “match all tags struct at the top level (/)”. The highlighted line is bound to create multiple checkboxes all with “check1” as id (non unique).

```
<part id="tree" class="JTree"/>
   <!-- add a checkbox on all top level nodes -->
   <part id="check1" anchor="tree" sub-part="/struct" class="JCheckbox"/>
</part>
```

- In this second example, the `anchor` attribute is obmitted and thus the id of the parent node (enclosing) is used as default. The result is the same as above.

```
<part id="tree" class="JTree"/>
   <!-- or using the default (tree) for anchor -->
   <part id="check1" sub-part="/struct" class="JCheckbox"/>
</part>
```

Using the xpath in a rule

We have seen how objects created with an xpath notation in the structure section might not have id's for the sub-parts. Therefore, we must use the same way to express a reference to the part in the rules. However, because XPath expressions can match multiple nodes in an XML document, then we have introduce another problems in the rules.

Consider this code sample. It is ment to define a behavior rule attached to all top level struct nodes of a part named tree. However, the problem occurs when the rule fires because the <action> portion won't know which UIML part actually received the event.

```
<condition>
   <event anchor-name="tree" sub-part="/struct" class="g:actionperformed" />
</condition>
<action>
   ... do something when a check is pressed
</action>
```

Consider this more concrete example. The intent is to setup a rule that changes a clicked cell's content to "clicked". However, the `subpart` shown in the `<action>` portion of the rule is still an XPath expression. This matches all top level `struct` nodes. While that is the clear intent of the use of `/struct` in `<condition>` it is not the intent in the `<action>` section. What we want/need is the analogous to the "`this`" pointer in `C++`, that is, we need a way to refer to the actual widget that is participating in the event.

```
<rule>
   <condition>
      <event part-name="tree" subpart="/struct" class="g:actionperformed"/>
   </condition>
   <action>
      <property part-name="table" subpart="/struct" name="content">
         clicked
      </property>
   </action>
</rule>
```

(end)

Appendices

Property Lists

- Property lists “types” of arrays, dictionaries, and primitive allow arbitrary composition of structures. For more information see http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/index.html

Example: Using P-lists as a way to capture structure

```
<plist version="1.0">
   <array>
      <array>
         <string>Color</string>
         <string>Size</string>
         <string>Material</string>
      </array>
      <array>
         <string>red</string>
         <string>blue</string>
         <string>green</string>
      </array>
      <array>
         <string>large</string>
         <string>medium</string>
         <string>small</string>
      </array>
      <array>
         <string>cotton</string>
         <string>nylon</string>
         <string>rayon</string>
      </array>
   </array>
</plist>
```

XPath examples

The following examples show a model representation followed by XPath expressions that would select different parts of the XML model. The XPath expressions appear in bold and red.

Row Major example 1

```
<model>
    <row>Color, Size, Material</row>
    <row>red, large, cotton</row>
    <row>blue, medium, nylon</row>
    <row>green, small, rayon</row>
</model>
```

- **model** - selects all children of model
- **model/row** = selects all row elements of model
- **//row** = selects all row elements no matter where they appear
- **//@attr** = selects all attributes named "attr"
- **/model/row[1]** = selects first row element child of model
- **/model/row[last()]** = selects last row element child of model
- **/model/row[position()>1]** = selects all rows but the first

Other XPath notations: Wildcards

- ***** match any element node
- **@*** match any attribute
- **node()** match any node of any kind

More examples

```
<model>
    <row>
        <column>Color</column>
        <column>Size</column>
        <column>Material</column>
    </row>
    <row>
        <column>red</column>
        <column>large</column>
        <column>cotton</column>
    </row>
    <row>
        <column>blue</column>
        <column>medium</column>
        <column>nylon</column>
    </row>
    <row>
        <column>green</column>
        <column>small</column>
        <column>rayon</column>
    </row>
</model>
```

- **model** - selects all children of model
- **model/row** = selects all row elements of model
- **model//column** = selects all columns, no matter where they are

- **model/row[1]/column[1]** = selects the Color cell
- **model/row[1]/column[position()>2]** = selects the third column

Example matching attribute values

```
<model>
    <row>
        <cell value="Color"/>
        <cell value="Size"/>
        <cell value="Material"/>
    </row>
    <row>
        <cell value="red"/>
        <cell value="large"/>
        <cell value="cotton"/>
    </row>
    <row>
        <cell value="blue"/>
        <cell value="medium"/>
        <cell value="nylon"/>
    </row>
    <row>
        <cell value="green"/>
        <cell value="small"/>
        <cell value="rayon"/>
    </row>
</model>
```

- **//cell** = selects all cells no matter where they appear
- **//@value** = selects all attributes named "value"
- **/model/row/cell[@value="green"]** = selects the "green" cell
- **/model/row[1]** = selects first row element child of model
- **/model/row[last()]** = selects last row element child of model
- **/model/row[position()>1]** = selects all rows but the first