OASIS
User Interface Markup Language (UIML)
Technical Committee (TC)
Minutes

Logistics

<table>
<thead>
<tr>
<th>Meeting Date</th>
<th>May 15, 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting Time</td>
<td>12:00 PM EST</td>
</tr>
<tr>
<td>Location</td>
<td>Meeting held via Teleconference hosted by Harmonia, Inc.</td>
</tr>
<tr>
<td>Duration</td>
<td>1 Hour</td>
</tr>
<tr>
<td>Chair</td>
<td>Jim Helms</td>
</tr>
<tr>
<td>Recording Secretary</td>
<td>Jim Helms</td>
</tr>
</tbody>
</table>

Attending

<table>
<thead>
<tr>
<th>Name</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mr. Jim Helms</td>
<td>Harmonia, Inc.</td>
</tr>
<tr>
<td>Mr. Brendan Berry</td>
<td>Harmonia, Inc.</td>
</tr>
<tr>
<td>Dr. Jean Vanderdonckt</td>
<td>Universite Catholique de Louvain</td>
</tr>
<tr>
<td>Dr. Marc Abrams</td>
<td>Virginia Tech University</td>
</tr>
<tr>
<td>Mr. Robbie Schaefer</td>
<td>C-Lab, Paderborn University</td>
</tr>
<tr>
<td>Mr. Jo Vermeulen</td>
<td>Hasselt University</td>
</tr>
<tr>
<td>Dr. Kris Luyten</td>
<td>Hasselt University</td>
</tr>
</tbody>
</table>

Business in Order

- Discussion of Open Issues in the UIML Specification
- Discussion of <variable> tags and Arithmetic:
Discussion at the May meeting of the TC continued the topic of incorporating variable and arithmetic into the UIML Specification. Mr. Schaefer provided documentation and a number of examples of variable usage for inclusion in the discussion. His materials are included in the Supplemental Material Section of these minutes. At the last meeting, it was suggested that <op> tags be used as statements. One potential problem that was raised was that Boolean expressions do not typically have meaning in the action section. Thus, using <op>s for statements may not make sense. Instead a counter suggestion was made to introduce a separate tag.

While it is true that Boolean expressions do not typically have meaning in the action section, there are some cases where this would make sense. For one, it may be that the action is setting the Boolean value of a variable, or even passing the Boolean result of an <op> to a <call>. Given this, using <op> for statements will probably suffice.

This conversation also asked whether statements where valid in <condition> tags. The TC concluded that statements should be allowed in conditions to check for more complex conditions such as if A + B = C then perform operation X.

Along with the inclusion of arithmetic comes a need for the specification to define the way arithmetic operations are to be performed? Precedence for such definition exists in the Java specification. The W3C’s XPATH specification is another example of a specification that defined how certain operations will be performed and has resulted in different processors and engines remaining consistent. The TC will examine how these specifications define their arithmetic operations.

The syntax of variables was discussed in detail as well. The TC felt that setting variables in the same way as properties feels a bit unnatural to a developer, but provides a hierarchical way of expressing the relationship between variables and their values. Such syntax also provides explicit ordering and is consistent with standard forms used in the language already.

The TC also discussed how types can be defined for variables and decided to use the types defined by the XML Schema specification. These types should then be usable everywhere that we specify type (<call>, <d-method>, etc...).

Discussion of Template Parameterization

Mr. Vermeulen and a student he is working with at Hasselt University suggested a scheme for allowing templates to take parameters from the sourcing document. They suggested introducing a <template-parameter> tag that could be placed under any elements that can source a template. These parameters are then matched by id to template parameters defined in the template file.

In their initial scheme, the ids used are reserved from regular use. This simplified the implementation and allowed them to include it in their uiml.net implementation quickly. In general, a case would need to add a language mechanism to specify this. Mr. Vermeulen will supply documentation for template parameterization before the next meeting.

The scheme suggested still need to be modified to accommodate adding multiple sourcing as we discussed last month.

Discussion of How to Accelerate the TC’s Progress to Specification

In order to more quickly produce a UIML specification and draft Committee Standard, the TC has decided to use an OASIS-provided Wiki for a central repository of language discussions and move to a bi-weekly or semi monthly meeting schedule.
**Action Items**

- Jim Helms will update the Open Issue document to reflect the accepted solution proposals
- Jim Helms will update the Open Issues document to include the discussion materials put forth by members of the TC.
- Jim Helms will create examples of using the export attribute to address issue 4.1
- Dr. Luyten will create an API template for the connect functionality.
- Dr. Marc Abrams will send VoiceXML examples in UIML and multimodal white paper to the TC.
- Dr. Kris Luyten will send old examples of channels for multi-modal HCIs.
- Dr. Vanderdonckt will send a language example of when it would be useful to dynamically choose a duplicate template form instead of a single template.
- Dr. Vanderdonckt will prepare a UIML example of how selector is used in UIML and compare this to a CSS example.
- Mr. Schaefer will document how the UIML language will change to support variables.
- Mr. Vermeulen will write up template parameterization as a proposal for the UIML language.
- Mr. Helms will distribute a proposal for changes to action elements.
- Mr. Helms will add a blanket statement to the specification that defines the standard by which arithmetic operations will be performed.

**Adjournment**

The meeting ended at 1:15 PM EST to reconvene on June 19, 2006.

---

**Supplemental Materials**

**Usage of variables in UIML - some issues from/since April Meeting**

*Usage of <op>-Elements for statements:*

Instead of introducing a new <statement>-Element, assignments can be included in <op>-Elements, which means the name-attribute may contain additional values for the assignments, such as `<op name="add">`.

When <op>-Elements are used for assignments, they may also occur in the <action>-part, so we have to look at two cases:

- In the <condition>-part, a Boolean result is required, so a result from an arithmetic operation has to be interpreted as a Boolean value. It must be clearly defined which results should be interpreted as true or false. Or, we do not permit the usage of assignments in conditions, which means that the set of allowed name values differs between action and condition.
• In the <action>-part, the <op>-Element could be used for both, assignment of Boolean values (e.g. to set a state) or for arithmetic operations. However it must be defined for which data types which operations are allowed. (add, sub, mul, div would not make much sense for Booleans). Also the legal possibilities for casting must be defined.

This requires an update of section 5.7.6.1 to explain the semantic of the additional values and an update of section 5.7.7 to allow <op> as a child of <action>.

**Types of statements:**

Proposal: only basic arithmetic (add, sub, mul, div). Is this sufficient? In principle, an implementation could add further types of assignments but then we would end up with incompatible UIML-variants.

**Assignments:**

Currently, there are two ways, to assign values to variables. The implicit way uses the child element as the value to be set. For example set a=b looks:

```xml
<variable variable-name="a"/> <variable variable-name="b"/> <variable/>
```

This corresponds to the way properties are set in UIML. Another way is to use an explicit “set” operator:

```xml
<op name="set">
  <variable variable-name="a"/>
  <variable variable-name="b"/>
</op>
```

While the second looks more structured and is also in line with other operations, the first allows a compacter notation and adheres to common practice in UIML. Should both be allowed, or is it better to decide for one option, probably option one?

**Lifetime:**

To define the lifetime of a variable, it was discussed that the creation of the variable takes place when the renderer arrives at the first occurrence and the maximum lifetime will be until the renderer ends. Marc proposed to use the <restructure>-tag to delete variables explicitly. This is relevant when for example a sub interface is deleted and therefore the variables belonging to it are a) useless and b) should be in an initialized state when the sub-interface is instantiated again at a later time.

**Typing:**

Since UIML itself works only with Strings and has no notion of data types yet, it must be defined which types variables may support, how operations on these types are performed and how implicit casting between types can be achieved. Since defining types properly can be quite time consuming, we could use a subset of the primitive types of the XML-Schema definition (http://www.w3.org/TR/xmlschema11-2/).
This example shows a part of a copier to set the machine state WRT brightness and the page mode. There are two buttons, one to set the brightness (dark, normal and bright) and one to switch between single and double sided. Pushing the buttons cycles through the different states.

The statespace is \{dark, bright, normal\} \times \{single, double\} and the inputspace is \{setBrightness, setPageMode\}. Since the brightness and page mode settings are independent from each other, 5 rules are sufficient to describe 9 transitions.

```xml
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 3.1 Draft//EN" "UIML3_1a.dtd"/>

<!-- This example shows a part of a copier to set the machine state with respect to brightness and the page mode. There are two buttons, one to set the brightness (dark, normal and bright) and one to switch between single and double sided. Pushing the buttons cycles through the different states. -->

<uiml>
  <interface>
    <structure>
      <part id="setBrightness" class="G:Button"/>
      <part id="setPageMode" class="G:Button"/>
    </structure>
    <style>
      <property part-name="setBrightness" name="g:text"> brightness
      <property part-name="setPageMode" name="g:text"> page mode
    </style>
  </interface>
</uiml>
```
<behavior>
  <variable id="brightness" type="string">normal</variable>
  <variable id="pagemode" type="string">single</variable>
  <!-- constants -->
  <variable id="bright" constant="true" type="string"> bright</variable>
  <variable id="normal" constant="true" type="string"> normal</variable>
  <variable id="dark" constant="true" type="string"> dark</variable>
  <variable id="single" constant="true" type="string"> single</variable>
  <variable id="double" constant="true" type="string"> double</variable>

  <!-- If brightness == dark and setBrightness pressed then brightness = normal -->
  <rule id="cycleBrightness1">
    <condition>
      <op name="and">
        <event part-name="setBrightness" class="g:actionperformed"/>
        <op name="equals">
          <variable variable-name="brightness"/>
          <variable variable-name="dark"/>
        </op>
      </op>
    </condition>
    <action>
      <op name="set">
        <variable variable-name="brightness"/>
        <variable variable-name="normal"/>
      </op>
    </action>
  </rule>

  <!-- If brightness == normal and setBrightness pressed then brightness = bright -->
  <rule id="cycleBrightness2">
    <condition>
      <op name="and">
        <event part-name="setBrightness" class="g:actionperformed"/>
        <op name="equals">
          <variable variable-name="brightness"/>
          <variable variable-name="normal"/>
        </op>
      </op>
    </condition>
    <action>
      <op name="set">
        <variable variable-name="brightness"/>
        <variable variable-name="bright"/>
      </op>
    </action>
  </rule>
</behavior>
<!-- If brightness == bright and setBrightness pressed then brightness = dark -->
<rule id="cycleBrightness1">
  <condition>
    <op name="and">
      <event part-name="setBrightness" class="g:actionperformed">
        <op name="equals">
          <variable variable-name="brightness"/>
          <variable variable-name="bright"/>
        </op>
      </op>
    </op>
  </condition>
  <action>
    <op name="set">
      <variable variable-name="brightness"/>
      <variable variable-name="dark"/>
    </op>
  </action>
</rule>

<!-- If pagemode == single and setPageMode pressed then pagemode = double -->
<rule id="togglePageMode1">
  <condition>
    <op name="and">
      <event part-name="setPageMode" class="g:actionperformed">
        <op name="equals">
          <variable variable-name="pagemode"/>
          <variable variable-name="single"/>
        </op>
      </op>
    </op>
  </condition>
  <action>
    <op name="set">
      <variable variable-name="pagemode"/>
      <variable variable-name="double"/>
    </op>
  </action>
</rule>

<!-- If pagemode == double and setPageMode pressed then pagemode = single -->
<rule id="togglePageMode2">
  <condition>
    <op name="and">
      <event part-name="setPageMode" class="g:actionperformed">
        <op name="equals">
          <variable variable-name="pagemode"/>
          <variable variable-name="double"/>
        </op>
      </op>
    </op>
  </condition>
  <action>
    <op name="set">
      <variable variable-name="pagemode"/>
    </op>
  </action>
</rule>
Simple Toggle Button

Most simple example of a push-button, switching the state between on and off each time the button is pressed:

statespace = \{ON, OFF\}
inputspace = \{push\}

<!-- Most simple example of a push-button, switching the state between on and off each time the button is pressed. The statespace is \{on, off\} and the inputspace consists only of a push-button. -->

<uiml>
  <interface>
    <structure>
      <part id="button" class="G:Button"/>
    </structure>
    <style>
      <property part-name="button" name="g:text">ON/OFF</property>
    </style>
  </interface>
</uiml>
<behavior>
  <variable id="OnOffState" type="Boolean">false</variable>
  <variable id="TrueValue" constant="true" type="Boolean">true</variable>
  <variable id="FalseValue" constant="true" type="Boolean">false</variable>

  <!-- If state == true and button pressed then state = false -->
  <rule id="buttonPushedEven">
    <condition>
      <op nam="and">
        <event part-name="button" class="g:actionperformed">
          <op name="equals">
            <variable variable-name="OnOffState"/>
            <variable variable-name="TrueValue"/>
          </op>
        </op>
      </condition>
      <action>
        <op name="set">
          <variable variable-name="OnOffState"/>
          <variable variable-name="FalseValue"/>
        </op>
      </action>
    </rule>

  <!-- If state == false and button pressed then state = true -->
  <rule id="buttonPushedOdd">
    <condition>
      <op nam="and">
        <event part-name="button" class="g:actionperformed">
          <op name="equals">
            <variable variable-name="OnOffState"/>
            <variable variable-name="FalseValue"/>
          </op>
        </op>
      </condition>
      <action>
        <op name="set">
          <variable variable-name="OnOffState"/>
          <variable variable-name="TrueValue"/>
        </op>
      </action>
    </rule>

  </behavior>

  </interface>

  <peers>
    <presentation base="Generic_1.2_Harmonia_1.0"/>
  </peers>

</uiml>