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

Logistics

<table>
<thead>
<tr>
<th>Meeting Date</th>
<th>April 24, 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>
</tbody>
</table>

Business in Order

<table>
<thead>
<tr>
<th>Discussion of Open Issues in the UIML Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>In April, the UIML TC continued its discussion of Open Issues in the UIML 3.1 specification. We focused this month on the potential need to source multiple templates from a single element. Such a capability would allow users of UIML to combine styles without having to create awkward intermediate files.</td>
</tr>
</tbody>
</table>
Mr. Brendan Berry of Harmonia participated in the call to present a proposed method for realizing multiple sources. His proposal is included in the “Document on Multiple Sources” section in the Supplemental Materials section below.

Dr. Vanderdonckt pointed out a connection between this issue and several others including the Optional Template Elements issue, which involves the ability to define elements in a template that can optionally be omitted by the sourcing element.

Dr. Abrams suggested that the TC consider how the multiple source proposal would impact each element in UIML to determine if there are any special cases or conflicts. The TC concluded that no such special cases or conflicts exist, provided that a clear semantic for how the multiple sources would be applied is defined. The suggested semantic is dependant on the final syntax, but assuming a single source attribute with multiple semi-colon delimited values, the templates would be sources individually in left to right order.

The TC considered the use of multiple individual source attributes instead of a single attribute with multiple delimited values, but this solution is prevented by the XML specification.

Multiple sources present other interesting questions, such as what should the valid values of the “how” attribute be, and should they be applied to each template individually or as a whole. For example, should the how attribute have only one value that applies to all or should it have a value for each sourced template? For simplicity, the TC agreed to have one value for now.

The TC also discussed adding a new legal value for the “how” attribute that would basically do the opposite of “replace”. The semantics of such a value are under discussion.

Dr. Vanderdonckt asked if it makes sense to keep duplicates or conflicts? Does it help traceability? For example, depending on the source or user we may want to use the duplicates instead of a single template. Allows dynamic resolution of how the conflict should be resolved so that the application can pick the template at runtime. The state may not be known until the UI is run. Dr. Abrams commented that so long as the UIML specification defines “one clear winner” for each algorithm, and the purpose is to ultimately draw a UI, this seems sufficient. The TC also requested that Dr. Vanderdonckt send a language example for us to look at.

Mr. Vermeulen is concerned that the solution does not seem like a clean XML solution; however, he could not immediately recommend a better format. The TC will keep this under consideration and search for a solution more in line with the spirit of XML.

After our discussion of multiple sources in UIML, Dr. Vanderdonckt presented a possible solution that would allow UIML to utilize Optional Template Elements. Dr. Vanderdonckt’s write-up on his ideas has been included in section “Should the UIML Specification Support Optional Template Elements?” below. The proposed solution contains a “selector” notion that allows the UIML author to choose which template to activate. The idea for such a template selection mechanism comes from CSS 2. Dr. Vanderdonckt will prepare a UIML example of how selector is used in UIML and compare this to a CSS example.

From here the TC moved on to continue the discussion from last month of the incorporation of variables and arithmetic expressions into UIML. Mr. Schaefer pointed out that the <op> element needs a lessthanorequalto and greaterthanorequalto value to
allow a wider range of comparison.

The TC discussed the types, scope, and lifetime of variables. Mr. Schaefer said that in DISL, they supported integers, floats, and strings. Scope was restricted to the subtree in which the variable was defined. Variables were created when the file is rendered, but it is not clear when the variable is destroyed. The restructure tag can be used to delete parts of the tree and this mechanism could be used to delete variables as well. This would potentially allow variables to be not always global to one file but in between. For example, Parent A sources B which sources C and D. Variables defined at B should be shared across B, C, and D. Deleting B from A should delete the variables as well. Mr. Schaefer volunteered to look further into this.

The TC also discussed how to get “cross file” references to variables, meaning access to variables defined in other files (such as templates)? Such should be possible, but the variables should be defined higher up in the structure, and defined as Global. It could be done with namespaces or perhaps XPath. The TC will look further into this.

To realize arithmetic with variables the TC discussed using the <op> tag instead of adding a new statement element. This requires fewer tags in the language and seems to make sense since an <op> is just an evaluation that returns a result. This could negatively affect readability though.

### 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 create a state machine example and user interface for a calculator to determine if it can be done all in UIML with variables?

### Adjournment

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

### Supplemental Materials

**Document on Multiple Sources**
The purpose of this document is to propose a change to the UIML specification to allow elements to have multiple template sources.

Presently, UIML allows any single element to utilize templates one at a time. An element can describe its relationship to a given template by using the “source” and “how” attributes. The template tree will be merged into the tree of the sourcing element according to the rules implied by the “how” attribute. Once this operation is complete it is assumed that no further templates may be applied to that element.

This operation is generally sufficient for most uses of templates. However, some operations may require a greater degree of control than is afforded by this system. For example, a user may wish to provide a family of “style sheets” to accompany a set of interfaces. One style may easily be applied to multiple documents through the current system, but a single document may not utilize multiple styles in an n-to-1 fashion.

A similar effect may be achieved through the use of cascading templates, but does not sufficiently address all of the cases where multiple styles might be needed. For example, assume that a single project falls under the auspices of two organizations, “A” and “B”. Within the project there are three major interfaces. Each organization has its own style guide. Our documents are as follows:

Style A

Style B

Document 1

Must use Style A

Document 2

Must use Style B

Document 3

Must use Style A and Style B (assume order doesn't matter)

If we assume that the master interface-level style element of Document 3 must source both A and B, then under the current system Document 3 would source A which in turn would source B. However, if Style A sources Style B to satisfy the requirements of Document 3 then Document 1 will always include elements of Style B. The only way to satisfy the requirements of Document 3 without imposing undue constraints on Document 1 or Document 2 would be to create a third style that duplicates A and then sources B.

Clearly this is an undesirable solution as it negates some of the benefits that would lead a client to use style templates in the first place. Clients who utilize UIML templates and UIML-based style sheets will undoubtedly have large sets of documents with widely varying style requirements.
The plausibility of this scenario increases as UIML becomes more involved in collaborative projects between corporate entities.

One possible answer to this problem is to amend the UIML specification to allow the use of multiple sources on a single element. This would allow a client to specify a 1-to-n relationship between an element and a set of style templates. A possible change to the syntax would be as follows:

```xml
<style source="A#x;B#y;C#z"/>
```

Where A, B, and C are each documents containing the templates x, y, and z, respectively. The sources are delimited by semicolons. Each source is evaluated in the order that it appears when reading from left to right. The merging rules remain fundamentally the same. When a source is expanded, it is simply applied to style as it appears at that moment. An example:

**Document A**

```xml
<style id="x">

  <property name="size">100,100</property>

  <property name="color">red</property>

</style>
```

**Document B**

```xml
<style id="y">

  <property name="size">-3,-3</property>

  <property name="text">strawberry</property>

</style>
```

**Document C**

```xml
<style source="A#x;B#y" how="cascade">

  <property name="location">0,0</property>

</style>
```
Assume that a UIML model API is parsing this document. When the source is evaluated, the source string will be broken down into its constituent sources. The sources will then be evaluated one at a time. After the first source is evaluated, Document C will appear as follows:

Document C

```xml
<style source="A#x;B#y" how="cascade">

  <property name="location">0,0</property>

  <property name="size">100,100</property>

  <property name="color">red</property>

</style>
```

At this point, process of importing is normally finished. Under the proposed system the next step would be to expand the second source, resulting in the following:

Document C

```xml
<style source="A#x;B#y" how="cascade">

  <property name="location">0,0</property>

  <property name="size">100,100</property>

  <property name="color">red</property>

  <property name="text">strawberry</property>

</style>
```

The model has now imported the “text” property from Document B, but has detected that Document C already contains a “size” property (which actually originates from Document A). According to the rules of the “cascade” operation, the “size” property from Document B has been ignored.

Each template is imported individually as if the sourcing element references only that template even though the element may already contain elements imported from other templates. This system essentially “flattens” the inheritance hierarchy, ensuring that the importing element functionally inherits from only one template at a time and does so in an explicitly specified order.
This solution should enable the language to avoid the problems seen in other multiple inheritance schemes.

This solution allows a user to apply multiple style templates to a single document without needing to create a complex system of cascaded styles. This would sufficiently address the issue presented in Example 1 and would still adhere to the existing template rules. Most importantly, this would require no major changes to any elements in the UIML language.

However, it should be noted that some objections have been raised to this proposal. Most of these reservations are derived from the notorious vagaries of C++ style multiple inheritance. Although no specific counter-examples have been discovered, it remains a point of debate as to whether this change might allow atypical uses of UIML. If implemented as a “flattening” operation, multiple sources could provide a powerful solution for the current problem without allowing the same abuses as C++.

Nonetheless, unforeseen issues may arise when multiple sources are used in the context of elements other than style. For example, the part element may contain many different types of ancestor nodes of arbitrary depth. This might necessitate a syntax change to restrict multiple sources to the style element.

Further investigation may be necessary to safely introduce this mechanism to all source-capable elements. It would be worthwhile to investigate the benefits of this system, with careful attention paid to the potential abuses.

Should the UIML Specification Support Optional Template Elements?

Yes. A template could be defined as a set of predefined attributes whose value is assigned at design time. A template could be seen as a shortcut for repeating several assignments at the same time (e.g., repeating the same presentation attributes for whole or parts of a user interface).

A template is intrinsically reusable: once defined, it may be reused at any time in principle. In addition, this mechanism supports evolution of systems as a template could give rise to many other templates which could be considered as generalizations or specialization of a source template.

The definition of a template mechanism is particularly straightforward in UIML since all attributes are clearly separated in the four main sections, such as style, structure, and layout. Building a library of templates is a task that is inevitably related to the introduction of templates. But one should take care of not introducing too many templates with only little variations from one template to another, which may induce designers in confusion on what template to apply.

Proposed Solution

The following UML class diagram represents the meta-model of a possible template mechanism. The Template Class is subclassed into several sub-types depicting particular types of templates that could be applied separately and independently of each other, thus ensuring some modularity and flexibility. A template may be declared context-dependent if its application is peculiar to a particular context of use, where the context of use is here referred to as a triple composed of a user, a computing platform, and an environment.
A template parameter consists of any UIML attributes that may be subject to an assignment prior to development. A template parameter is characterized by an id, a name, a definition, a data type, and a template value. Each template may have one to many selectors, such as:

- universalSelector: applies the template to all elements belonging to the UIML.
- elementTypeSelector: applies the template to all elements belonging to the UIML which correspond to the selector’s type (e.g., all containers, all list boxes).
- classSelector: applies the template to all elements belonging to the UIML which corresponds to the selector’s type whose definition makes them part of the class (e.g., all containers having an id greater or equal to CC2, all list boxes having more than 15 items).
- idSelector: applies the template to only one element belonging to the UIML: the one whose id attribute matches the string contained in the parameter.

```
Concrete Interface Model

+id : string(idl)
+name : string(idl)
+creationDate : Date
+modificationDate : Date
+versionNumber : string(idl)
```

```
Template Parameter

-id
-name
-definition
-dataType
-templateValue
```

```
Selector

-id : string(idl)
-name : string(idl)
-scope : string(idl)
```

```
Style
```

```
Layout
```

```
Dialog
```

```
Font
```

```
Color
```

**Affected Sections of the Specification**

In principle, all UIML sections containing attributes whose value may be pre-assigned are subject to a template. A template could govern an entire UI or a portion of it. Several templates could be used for multiple parts and be based on an inheritance mechanism. If a local template is specified for a UIML fragment, then it overwrites the values assigned by a global template if any.