UIML
User Interface Markup Language

Ruben Deyhle

Stuttgart Media University (HdM)
Nobelstraße 10, 70569 Stuttgart, Germany
rd016@hdm-stuttgart.de
http://hdm-stuttgart.de

Abstract. UIML is a markup language to describe user interfaces. It simplifies the process of defining user interface elements, their structure, attributes and behavior. Theoretically, UIML can describe user interfaces for all conceivable devices, if a rendering tool for the target platform is available.
By now, UIML is not used in significant amount anymore.

This paper was written for the purposes of the Adaptive User Interfaces Course at Stuttgart Media University (HdM) in 2012 and is focused on the adaptation aspect of UIML.

Keywords: UIML, User Interfaces, XML, Adaptive User Interfaces

1 Introduction

This Paper looks into the basic idea and syntax of UIML.
In this introduction the basics are elucidated, including a short history. The key concepts of UIML are then elaborated in Sec. 2.
Sec. 3 describes the Syntax and important tags of UIML and includes an example.
The Adaptation aspect of UIML is elaborated in Sec. 4.
A Related Work Section (Sec. 5) compares UIML to other, somehow similar technologies.
Finally, the idea and current state of UIML is summarized in the conclusion (Sec. 6).

1.1 Basics

UIML ("User Interface Markup Language") is an XML-compliant meta-language to describe user interfaces. Its goal is to be "one language for all devices": a language to describe and define user interfaces for all existing, and even future devices. It was developed, because "it is too time consuming to hand-code a user interface for each device." ([2])

The approach is to have a markup language that can be extended and also apply to any user interface possible. A UIML document may be interpreted or compiled into a concrete user interface, for example Java Swing.

It is an open specification, so it may be used by everybody. However, as far as we know, UIML is not being used anymore today and many resources and tools for UIML are no longer available. [6]

1.2 History

The first version of UIML was introduced in 1997. It was followed by a "major redesign" in version 2 in 2000 and version 3 in 2002 (a "modest refinement"). UIML 3.1 came out as a draft in 2004. [6]

In 2008 the latest version of the standard, UIML 4, has been released. Since this version UIML is maintained by OASIS, the "Organization for the Advancement of Structured Information Standards"). [4]

The major force behind UIML was Harmonia, Inc. located in Blacksburg, Virginia, USA. However, they lost interest in UIML by now (2012) and only on an old version of Harmonias website some resources on UIML can be found. [7]

Since 2008, nothing seems to have happened with UIML.

2 UIML Key Concepts

UIML is XML-compliant, so it exists of tags and attributes and is much like HTML.

For the sake of separation between presentation, content and behavior, there are only generic, abstract tag names, that are independent of any UI metaphor. For example, there are no tags for a window or a button, because this would limit the language to graphical user interfaces on a classic desktop environment. Instead, there are only abstract `<part>` elements that are mapped to a specific user interface element like a Java button using attributes and associated properties.

The Structure is not necessarily static, instead UIML has a dynamic tree structure, so sub-trees are interchangeable. Template support enables simple reuse of UIML fragments.

"UI" in UIML is simply a set of interface elements, defined in abstract `<part>` elements. These parts have content, and some parts are able to receive input. So the logical user interface element structure is defined by a virtual tree of parts, that is dynamically modifiable.

The behavior of the UI is determined by the `<behavior>` element. As in rule-based languages like Prolog, the behavior of user interface elements is described with condition-action-rules. This way, UIML is able to describe a complete user interface, including frontend interface logic.

However, no automatic translation to different UI metaphors is possible with UIML, and an UIML author always needs some knowledge about the target device. To actually use UIML, a toolkit for translating the UIML code into application code is needed. With available toolkits, the input UIML has to implement specific attributes and part names, so it is not possible to define one single UIML file to generate user interfaces for different applications. But a translating toolkit that accepts very generic UIML files is imaginable, just not existing yet. [5]

3 Syntax

Every UIML document starts with the xml prologue, and a doctype marking the document as UIML document. Like HTML, UIML has a root element (`<uiml>`), which contains all the other elements. There are four child Elements of `<uiml>` defined in UIML: `<head>` for metadata, `<template>` for reuse of fragments, `<interface>` for the UI definition and `<peers>` for mapping to UI toolkits. The `<interface>` element contains organization (`<structure>` child element) as well as behavior (`<behavior>` child element) of user interface parts. Additionally, the `<style>` element defines properties of the interface parts and `<content>` associates words, sounds or images. So in `<structure>` there are only nested `<part>` elements, each with an unique `id` attribute. In `<style>` there is usually a large list of `<property>` elements, that are assigned to the part elements using the `part-name` attribute. [5]

3.1 Example

The following listing is an example UIML file, that creates the user interface depicted in Fig. 1: a simple dictionary with a list of animals and a definition textbox where a click on an animal displays the corresponding definition. Obviously, there is no separation between user interface and content in this example.

The example is written for Harmonias Java UI Toolkit, so Java-AWT-specific names are used.

Example of a UIML file for Java AWT/Swing

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//OASIS//DTD UIML 4.0a Draft//EN" http://uiml.org/dtds/UIML4_0a.dtd>

    <!-- This is Dictionary.ui. Displays one window on the screen containing a list of animals
    and a textbox.  Clicking an animal's name displays a definition in the textbox. -->

<uiml>
```

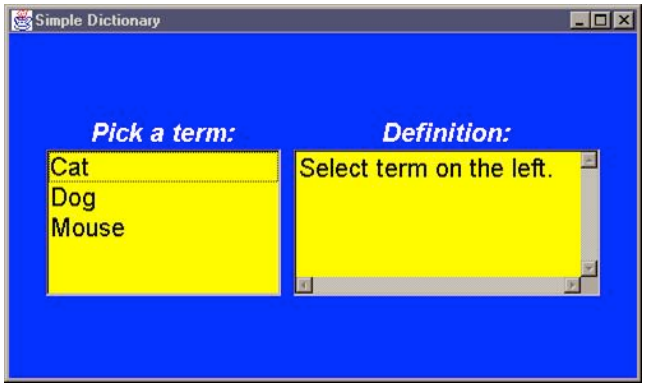

Fig. 1. Example Java Application generated by following UIML Code. Taken from [5].

```
<peers>
  <presentation base="Java_1.5_Harmonia_1.0"/>
</peers>
<interface>
  <structure>
    <part class="JFrame"   id="JFrame">
      <part class="JLabel"   id="TermLabel"/>
      <part class="List"     id="TermList" />
      <part class="JLabel"   id="DefnLabel"/>
      <part class="TextArea" id="DefnArea"/>
    </part>
  </structure>
  <style>
    <property  part-name="JFrame"    name="layout">java.awt.GridBagLayout</property>
    <property  part-name="JFrame"    name="background" >blue</property>
    <property  part-name="JFrame"    name="location"   >100,100</property>
    <property  part-name="JFrame"    name="size"       >500,300</property>
    <property  part-name="JFrame"    name="title"      >Simple Dictionary</property>
    <property  part-class="JLabel"   name="foreground" >white</property>
    <property  part-class="JLabel"   name="gridx"      >RELATIVE</property>
    <property  part-class="JLabel"   name="gridy"      >RELATIVE</property>
    <property  part-class="JLabel"   name="font"       >Helveticabolditalic-20</property>
    <property  part-name="TermLabel"  name="text"       >Pick a term:</property>
    <property  part-name="DefnLabel" name="text">Definition:</property>
    <property  part-name="DefnLabel" name="gridx"      >1</property>
    <property  part-name="DefnLabel" name="gridy"      >0</property>
    <property  part-name="DefnLabel" name="insets"     >0,10,0,0</property>
    <property  part-name="TermList"  name="background" >yellow</property>
    <property  part-name="TermList"  name="gridx"      >0</property>
    <property  part-name="TermList"  name="gridy"      >RELATIVE</property>
    <property  part-name="TermList"  name="fill"       >BOTH</property>
    <property  part-name="TermList"  name="font"       >Helvetica-20</property>
```

```
  <property  part-name="TermList"  name="content">
    <constant model="list">
      <constant id="Cat"   value="Cat"/>
      <constant id="Dog"   value="Dog"/>
      <constant id="Mouse" value="Mouse"/>
    </constant>
  </property>
  <property  part-name="DefnArea"  name="background" >yellow</property>
  <property  part-name="DefnArea"  name="gridx"      >1</property>
  <property  part-name="DefnArea"  name="gridy"      >RELATIVE</property>
  <property  part-name="DefnArea"  name="text"       >Select term on the left.</property>
  <property  part-name="DefnArea"  name="columns"    >20</property>
  <property  part-name="DefnArea"  name="rows"       >4</property>
  <property  part-name="DefnArea"  name="editable"   >false</property>
  <property  part-name="DefnArea"  name="insets"     >0,10,0,0</property>
  <property  part-name="DefnArea"  name="font"       >Helvetica-20</property>
</style>
<behavior>
  <rule>
    <condition>
      <op name="and">
        <event part-name="TermList" class="ItemListener.itemStateChanged"/>
        <op name="equal">
          <property event-class="ItemListener.itemStateChanged" name="item"/>
          <constant value="0"/>
        </op>
      </op>
    </condition>
    <action>
       <property part-name="DefnArea" name="text">Carnivorous, domesticated mammal
         that's fond of rats and mice</property>
    </action>
  </rule>
  <rule>
    <condition>
      <op name="and">
        <event part-name="TermList" class="ItemListener.itemStateChanged"/>
        <op name="equals">
          <property event-class="ItemListener.itemStateChanged" name="item"/>
          <constant value="1"/>
        </op>
      </op>
    </condition>
    <action>
      <property part-name="DefnArea" name="text">Domestic animal related to a wolf
        that's fond of chasing cats</property>
    </action>
  </rule>
  <rule>
    <condition>
```

```
          <op name="and">
            <event part-name="TermList" class="ItemListener.itemStateChanged"/>
            <op name="equals">
              <property event-class="ItemListener.itemStateChanged" name="item"/>
              <constant value="2"/>
            </op>
          </op>
        </condition>
        <action>
          <property part-name="DefnArea" name="text">Small rodent often seen running
            away from a cat</property>
        </action>
      </rule>
    </behavior>
  </interface>
</uiml>
```

(Example taken from [5])

The structure is basically a `JFrame` containing a `List`, a `TextArea` and two `JLabel`s. All Java-specific attributes like sizes and positions are listed as `<property>` elements. One may think this is no real "XML-style", because the properties are not attributes of the interface parts but separate elements at another place in the document. But this way UIML stays abstract and independent of UI metaphors: instead of predefining possible properties as XML-attributes, any property is possible, and not only common properties of graphical desktop user interfaces.

In this example the content is simply defined as constant values in a `<property>` element.

The `<behavior>` part at the bottom of the document contains the rules for each element. Using an element `<op>` with `name` attributes associating a logical operator, logical expressions can be described using XML syntax inside the `<condition>` element.

The `<action>` element simply contains additional properties, which are applied in case the condition is true.

4 Adaptation

This section elaborates how UIML translates into a concrete user interface and to which aspects UIML is able to adapt to.

4.1 Interpretation and Compilation

A UI may be rendered out of a UIML document using two different approaches: either interpreting or compiling.

Due to the XML structure, a UIML UI can be easily interpreted on a client device, much like HTML. As an alternative approach, a UIML document can be compiled into another language, for example Java Swing.

The former UIML website [6] listed renderer/interpreter for C++, HTML, Java, .NET, Symbian, QT, Visual Basic, WML and even VoiceXML for voice-only user interfaces. However, most of the linked resources are not available anymore. Thus, it is difficult to use UIML nowadays. UIML rises and falls with its available rendering tools: a translation of a very generic UIML document into different concrete user interfaces would be possible with a proper renderer.

Whether a UIML document can generate a Java user interface, a web application, a C++ application or an VoiceXML interface is only depending on vocabularies: in the end, the `<part>` attributes already define the type of the resulting user interface. If you use HTML UI elements here, you can't simply generate a C++ user interface from this UIML file.

So UIML adapts not at runtime, but at design time.

4.2 Adaptation Parameters

As said before, the power of UIML depends on the available rendering tools. By design, UIML may adapt to very different user interface requirements. This may be different applications and platforms (there is virtually no limitation here) as well as different devices (like a computer, a smartphone, a tablet, a voice computer or even some future device). Different modalities are supported, UIML may describe a desktop user interface with mouse, keyboard and display as well as a mobile touch-screen interface, a gesture control interface or a pure voice-command-based user interface.

Due to the possible separation between logical user interface and content, UIML may also adapt to different languages.

However, UIML only supports adaptation at design time: an UIML author has to know exactly what user interface he wants in the end.

Due to this limitation, UIML cannot adapt to specific end-user requirements or "the user" in general, and also not to the environment or organization structure at runtime. Once defined, a UIML-generated user interface is "fixed" and underlies the limitations of the target platform, no runtime adaptation is supported by UIML.

(These are the results of a discussion on the adaptation aspects of UIML in the Adaptive User Interfaces Course at Stuttgart Media University in April of 2012.)

5 Related Work

The idea of abstract user interface specifications is not exactly new. It goes back to the early 1980ies and the User Interface Management Systems (UIMSs) that were an early approach on separating the user interface from the application functionality. [8]

By now, there are several technologies like UIML out there that are based on an XML syntax.

Similar to UIML is SEESCOA (Software Engineering for Embedded Systems using a Component-Oriented Approach), which is about the same age as UIML. It also defines an abstract UI structure using XML that has to be transformed into a concrete user interface (SEESCOA uses XSL stylesheets for that). However, SEESCOA can also adapt at runtime by changing into a different UI when needed, for example personalized version for a disabled person. It is centered on embedded systems and not as universal as UIML. [9]

Also comparable to UIML is XForms, also an XML language. It can be used to abstractly describe HTML forms. XForms explicitly defines UI components and could easily be interpreted like HTML in a web browser. It was supposed to be an addition to XHTML 2.0 and could have been implemented in browsers. However, because XHTML 2.0 was abandoned, so was XForms. [10]

One of the latest approaches of an abstract user interface XML language is UsiXML. But, unlike UIML, it has several different abstraction layers that can be translated into each other. It supports task modeling as highest abstraction, while the lowest abstraction layer, the FUI (Final User Interface), is a concrete user interface. [11]

6 Conclusion

UIML adapts to any possible platform, and can describe virtually any user interface. However, it is limited to adapting at design time and is depending very much on the availability of toolkits to transform UIML into actual user interfaces.

UIML was designed to, and according to the authors also is “usable by non-programmers and occasional users”. [1]

While this may be true for plain UIML, it certainly is not true when writing UIML for rendering into an actual user interface like Java. The UIML author needs knowledge about the specific user interface elements, their names and attributes. This way, one may argue that UIML is just syntactic sugar for classical user interface programming.

Due to the rise of graphical user interface editors, that are built-in to the big IDEs in recent time, the premise leading to the development of UIML ([2]) is challenged: is it still too time consuming to build a device-specific (or platform-specific) user interface with all the tools we have today? Maybe they already rendered UIML superfluous before it could even establish itself.

References

1. Marc Abrams, Constantinos Phanouriou, Alan L Batongbacal, Stephen M Williams, Jonathan E Shuster, UIML: an appliance-independent XML user interface language, Computer Networks, Volume 31, Issues 1116, 17 May 1999, Pages 1695-1708, ISSN 1389-1286, 10.1016/S1389-1286(99)00044-4. http://www.sciencedirect.com/science/article/pii/S1389128699000444
2. Phanouriou, Constantinos. 2000. UIML: A Device-Independent User Interface Markup Language. Phd Thesis Virginia Polytechnic Institute and State University. Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.4719&rep=rep1&type=pdf
3. James Helms and Marc Abrams. 2008. Retrospective on UI description languages, based on eight years' experience with the User Interface Markup Language (UIML). Int. J. Web Eng. Technol. 4, 2 (May 2008), 138-162. DOI=10.1504/IJWET.2008.018095 http://dx.doi.org/10.1504/IJWET.2008.018095
4. UIML on OASIS website. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml May 30, 2012
5. UIML 4.0 committee draft specification. http://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf May 30, 2012
6. uiml.org http://web.archive.org/web/20110719172319/http://www.uiml.org/index.php (Snapshot from Jul 19, 2011)
7. Harmonia UIML Resources http://web.archive.org/web/20060712210738/http://www.harmonia.com/resources/presentations/index.htm (Snapshot from Jun 6, 2006)
8. Philip J. Hayes, Pedro A. Szekely, and Richard A. Lerner. 1985. Design alternatives for user interface management sytems based on experience with COUSIN. SIGCHI Bull. 16, 4 (April 1985), 169-175. DOI=10.1145/1165385.317488 http://doi.acm.org/10.1145/1165385.317488
9. Kris Luyten and Karin Coninx. 2001. An XML-Based Runtime User Interface Description Language for Mobile Computing Devices. In Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers (DSV-IS '01), Chris Johnson (Ed.). Springer-Verlag, London, UK, 1-15.
10. Mikko Honkala and Mikko Pohja. 2006. Multimodal interaction with xforms. In Proceedings of the 6th international conference on Web engineering (ICWE '06). ACM, New York, NY, USA, 201-208. DOI=10.1145/1145581.1145624 http://doi.acm.org/10.1145/1145581.1145624
11. Sophie Lepreux, Jean Vanderdonckt, Christophe Kolski. 2010. User Interface Composition with UsiXML. https://lilab.isys.ucl.ac.be/bchi/publications/2010/UsiXML2010/Lepreux-UsiXML2010.pdf 20 Jun 2012