CUIML: A language for generating multimodal human-computer interfaces

Christian Sandor **Thomas Reicher**

{sandor|reicher}@in.tum.de

Technische Universität München
Department of Computer Science
Chair for Applied Software Engineering
Munich, Germany

Abstract

CUIML, the Cooperative User Interfaces Markup Language, was developed as part of the DWARF project. The goal of DWARF is the development of a framework for augmented reality applications running on wearable computers. For wearable systems, the HCI (human-computer interface) consists of numerous I/O devices such as head mounted displays, palm-size devices, and speech recognition systems. This should lead to a multimodal interaction with the user. To make the development of multimodal user interfaces easier, we propose a unified user interface description language. This language describes the user interface in an abstract way and allows to map it to different output and input components. To keep the different parts of the user interface in a consistent state, a controller object is needed.

We used UIML, the User Interface Markup Language, as a starting point. CUIML offers an abstract description language for the views and the controller. By XSL transformations, these presentation free HCI descriptions are converted to markup languages that can be displayed on the various I/O devices. Furthermore the controller, whose internal structure is a deterministic finite automaton (DFA), is also configured by CUIML.

CUIML is a promising approach to solving the problems that occur, when developing a multimodal HCI. On a higher level of abstraction CUIML is a description language for views and controllers of the MVC (Model/View/Controller) design pattern that can be transformed to the required structures at runtime. Because it is based on XML it can be extended easily to support many different wearable systems.

Keywords: XML, multimodal user interfaces, wearable computing, augmented reality, XSL

1 Introduction

CUIML, the Cooperative User Interfaces Markup Language, was developed as part of the DWARF project [1]. The goal of DWARF is the development of a framework for augmented reality applications running on wearable computers. Augmented Reality (AR) is a technology which combines a real-world scene with virtual objects created from a computer. The virtual objects may be virtual artifacts placed into the real-world scene or additional non-geometric information about existing real objects, for example numbers of spare parts for an engine viewed by a mechanic. A definition of AR can be found in [2]. In DWARF we want to use AR on wearable computers as described in [3].

A desired feature of AR on wearable computers is the multimodal human-computer interaction. This means that the system should support user input and system output by various ways. User input could use components such as speech or gesture recognition, system output could use components

such as voice output, 3D graphics, and text. These components have different features and properties as well as similar ones. This can be used to combine different components to complement each other. For example, in a tour guide the way is shown in 3D graphics in a head-mounted display in form of an arrow, but information about a tourist attraction is given in text and 2D graphics. The similar features of devices allows the user to interact via more than one respective input or output components. Another example is user input by pressing the button "OK" or by saying "OK" to a speech recognition component.

In CUIML, we want to describe the human-computer interaction using an abstract language. This language should allow to use various input and output components in combination and in addition to each other. UIML, the User Interface Markup Language is an abstract language for user interface definition we used as a starting point for CUIML. In DWARF we use CUIML for the definition of the presentation layer of the AR systems.

In the next section, we describe the results of a requirements analysis we made about the human-computer interaction in multimodal systems such as AR. In section 3, we give an overview about work related to CUIML in 4 we describe CUIML and give an example of its use in section 5. Section 6 is about the results of our work and future plans.

2 Requirements analysis

First let us define a multimodal system [4]:

> *"In the general sense, a multimodal system supports communication with the user through different modalities such as voice, gesture and typing. Literally "multi" refers to "more than one" and the term "modal" may cover the notion of "modality" as well as that of "mode"*
>
> 1. *Modality refers to the type of communication channel used to convey or acquire information. It also covers the way an idea is expressed or perceived, or the manner an action is performed.*
> 2. *Mode refers to the state that determines the way information is interpreted to extract or convey meaning.*
>
> *(...) multimodality is the capacity of the system to communicate with the user along different types of communication channels and to extract and convey meaning automatically. (...) a multimodal system is able to model the content of the information at a high level of abstraction."*

Based on these characteristics of multimodal systems and on a general rule for HCIs, we derived the following requirements for CUIML:

Modeling information at a high level of abstraction: The information that should be presented to the user is sent via different channels but still should be described in one place. A logical consequence is that this information must be generic. The question "how can this generic information be viewed?" leads to our next requirement.

Mapping of generic information to device-dependent presentations: We need a mechanism to generate presentations from high-level descriptions. This approach has the advantage that additional input and output channels can be added easily by adding new transformers for the new channels. The generic information does not have to be changed.

Synchronization of the different channels: Once we generate the information needed to communicate with the user, how are the different channels synchronized? A design pattern that solves this problem is the Model/View/Controller (MVC, see [5]).

Controller configuration: There has to be a central instance that represents the state of the system that can be changed by user input and external events. This should be configurable at a high level of

abstraction like a DFA.
Short response times of the system: This requirement is not inherent to a multimodal system, but constitutes a basic requirement to all HCIs.

3 Related Work

By literature investigation we found some interesting work addressing these points. Especially important are: UIML, Petri Nets and S. Riss's Workflow Engine, as they are integrated into CUIML.
UIML (User Interface Markup Language), proposed in a paper by Mark Abrams et. al. [6] has already been submitted to the w3 consortium for standardization [7]. Below are the key features of UIML.

- UIML is a generic interface description language that can be rendered to many different device-dependent languages like VoiceXML, Java, HTML and WML.
- A family of interfaces can be created with the common features factored out.
- UIML is web based, and thus a client-server architecture.
- The interfaces generated with UIML can send events to arbitrary destinations, but cannot receive events. Instead a new interface according to the changed context is sent to the client.
- UIML was designed to generate views that do not interact with each other. This forced us to enhance the concepts inherent to UIML.

Petri Nets: The controller that synchronizes the views according to user input has to be described abstractly. A mechanism for this are Petri Nets that have already been suggest as controller for UIs ([8]). Petri Nets are "... a model for discrete event systems which has modeling power more than finite automaton and less than Turing machine" (cited after [9]). But we wanted to keep things simple for our first prototype which led to the next point.
S. Riss's Workflowengine: S. Riss wrote an implementation of a Workflow engine (WFE, [10]) that is based on a DFA. DFAs are not as powerful as Petri Nets, but for most controllers a DFA should be adequate. The DFA is described in an is a high-level description language, scilicet a XML dialect.
XForms [11] suggests to replace the current HTML forms by a mechanism that decouples data, logic and presentation. This sounds promising, but so far only a draft for the data model has been published. Drafts for the user interface and the protocols are still being worked out.
Mozquito [12] implements some parts of the XForms specification by FML (Form Markup Language). But Mozquito has different design goals that makes it unsuitable for our project:

- Mozquito is purely web based. This contradicts the requirement of fast response times as HTTP has no quality of service implemented yet. In our design described in section 4 the communication mechanism is decoupled and thus can be exchanged.
- Lack of a high-level description of the central instance to provide meaning to the user input.

ARML (Augmented Reality Media Language, as discussed in [13]) describes some requirements of our concept, but no details of the implementation are given. The paper does not reveal how that language is interpreted at runtime and how synchronization is achieved.

4 Design of CUIML

In this section we explain the design of CUIML and the runtime structures that are generated. In addition, we give a short introduction about the components that are needed to glue the generated parts together.

Runtime behaviour of the components

The client initialization is shown in diagram 2. The client requests a HCI by sending a HTTP request to a web server that renders the components (refer to figure 1) and sends them back to the client. The rendering is done by applying XSL transformations (eXtensible Stylesheet Language [14]) to the CUIML file. A typical client set up is shown in figure 3.
We describe below the generated components:

Controller: The central component on the client side is the Controller. It embodies the central instance to synchronize the views and keep track of the current state of the HCI. In our prototype, we use S. Riss's WFE implementation.
View: In our concept, a view is the display of markup language. The view can send events to the Controller when the user interacts with it, for example by clicking on a link. To enable this kind of communication an event adapter has to be generated which is not shown in our diagrams for simplicity reasons. The concept of describing and rendering of the views has been adopted from UIML.
Manipulator: Another enhancement over UIML is the concept of manipulators. To achieve faster response times, changes to the view should be done by the manipulators instead of rendering new CUIML descriptions to the according view. Views described by markup languages can be accessed by the Document Object Model (DOM, [15]). For other types of views, proprietary access mechanisms have to be found. For example the Reflection API can be used to access Java objects at runtime.

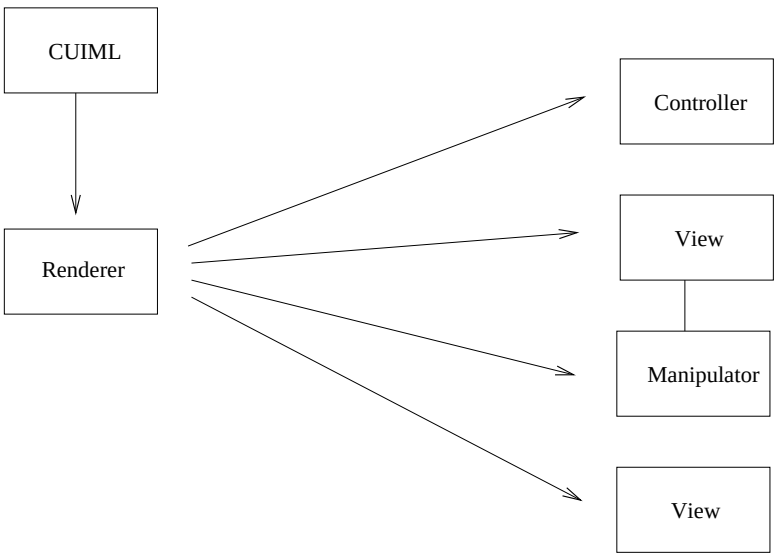


Figure 1: Generation of the HCI components

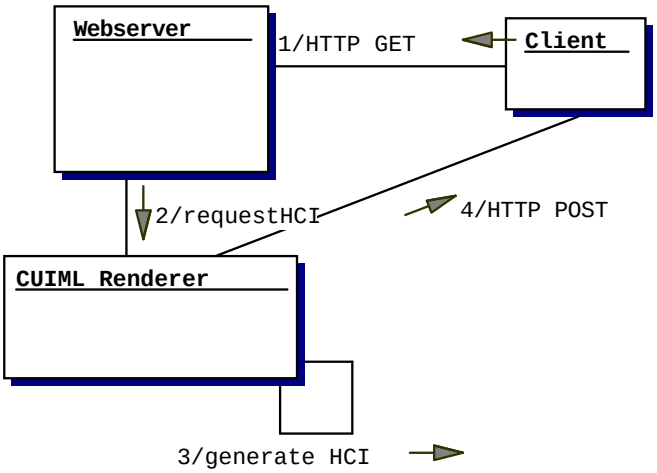


Figure 2: Initialization of the client

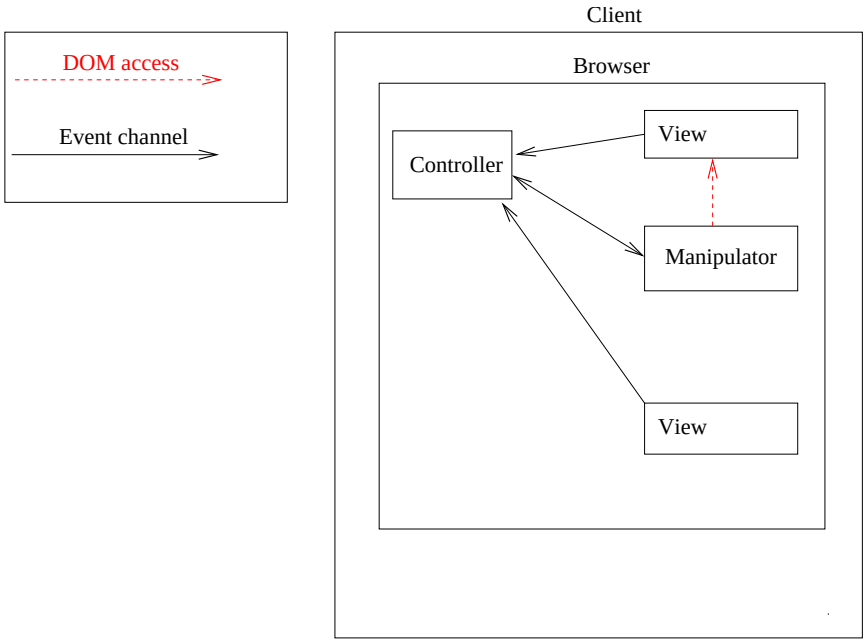


Figure 3: Components running at a client

Describing the HCI with CUIML

A CUIML document follows this template:

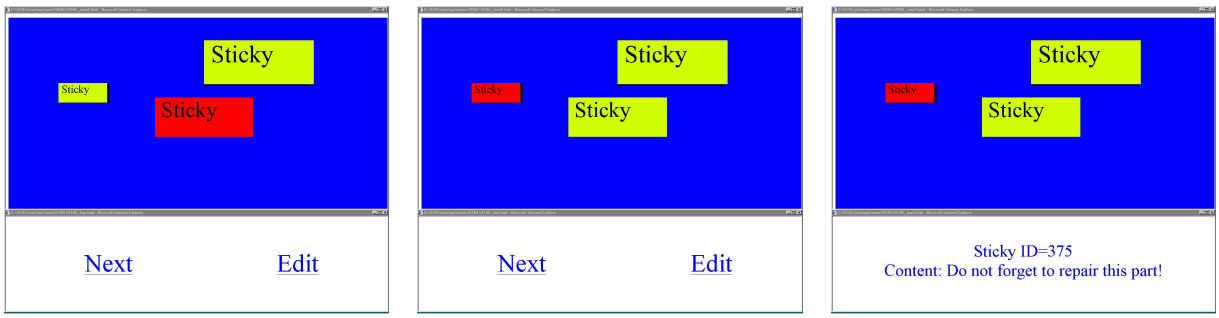
```
<?xml version="1.0">
<cuiml>
  <controller>
  </controller>
  <genericHCI>
      <interface>
      </interface>
      <behaviour>
      </behaviour>
  </genericHCI>
  <mapping>
    <hci type="VRML,HTML">
     <presentation>
     </presentation>
     <logic>
     </logic>
    <hci type="VRML and HTML">
  </mapping>
</cuiml>
```

Below we describe the semantics of the used elements:
`<cuiml>` is the root element of CUIML. There are three mandatory children: `<controller>`, `<genericHCI>` and `<mapping>`. `<genericHCI>` describes the structure and behaviour of the whole HCI (we embedded a subset of UIML for this purpose). The `<interface>` section allows to specify the structure of the HCI. The `<behaviour>` part, which events are fired under which user input and which manipulations occur on what events coming from the controller. If the manipulations become complicated, an external bean can be specified that fulfills the transformations of the view, instead of specifying them in CUIML.
In the `<mapping>` section the generic HCI description is mapped to programming-language-level constructs. The renderer is configured by this section. The two subsections are `<presentation>` and `<logic>`. The `<presentation>` section lists the mapping for the parts described in the `<interface>` section, the `<logic>` part maps the events to the programming-language-level.
The `<controller>` element embeds the Workflow Markup Language, developed by S. Riss ([10]). The language describes a finite state automaton that represents the Controller of the MVC pattern. The states of the automaton represent the states of the HCI and the transitions are most often triggered by incoming events fired by the views. Other possible conditions (events from custom components and timers for example) can be combined to boolean expressions to describe other triggers for transitions.

5 An example

Scenario

In our scenario the user has to choose from three different stickies by inputting “select next sticky” Stickies are virtual notes that can be a attached to real objects. When he has chosen the right sticky, he can edit the sticky by submitting “edit sticky” The GUI consists of a HTML and a VRML view (see figure 4). In the following sections we first describe the generic part of the HCI and then the rules for transforming them to a concrete HCI with a VRML and a HTML view.



(a) Initial state (b) "Next" pressed (c) "Edit" pressed

Figure 4: Screenshots of the scenario

Generic section in CUIML

First we describe the components of the HCI inside the `<interface>` tag: We cut out the description of one output and one input element.

```
<interface>
  <part class="OutputSection" name="output">
    <part class="Sticky" name="sticky1"> </part>
  </part>
  <part class="InputSection" name="input">
    <part class="Choice" name="next"> </part>
  </part>
</interface>
```

Inside the `<behaviour>` section the events that occur are listed. One event ("nextSelected") that should be sent when the user selects the "next" item, and one that can be sent from the Controller ("sticky1 selected").

```
<behaviour>
  <rule>
    <condition> <event part-name="next" class="ChoiceSelection"/> </condition>
    <action> <event class="OutEvent" name="nextSelected"/> </action>
  </rule>
   <rule>
    <condition> <event class="InEvent" name="sticky1 selected"> </condition>
   </rule>
</behaviour>
```

Definition of the Controller

We do not explain how the associated code in CUIML looks like, because the syntax of the Controller description has not been fixed at the time this paper was written. The soon to be published work [10] will explain the CUIML-level description.

Instead we give an explanation of the functionality the controller has to fulfill. The Controller's structure being a DFA is shown in figure 5. For every sticky that can be selected there is an appendant state. There is also a state that will be reached, when the user chooses to edit a sticky.

The transitions are triggered by events sent by the views (for example the "nextSelected" event in the `<behaviour>` section) and can cause the controller to fire events (like the "sticky1 selected" event) to the proper manipulators so that the associated view is updated accordingly.

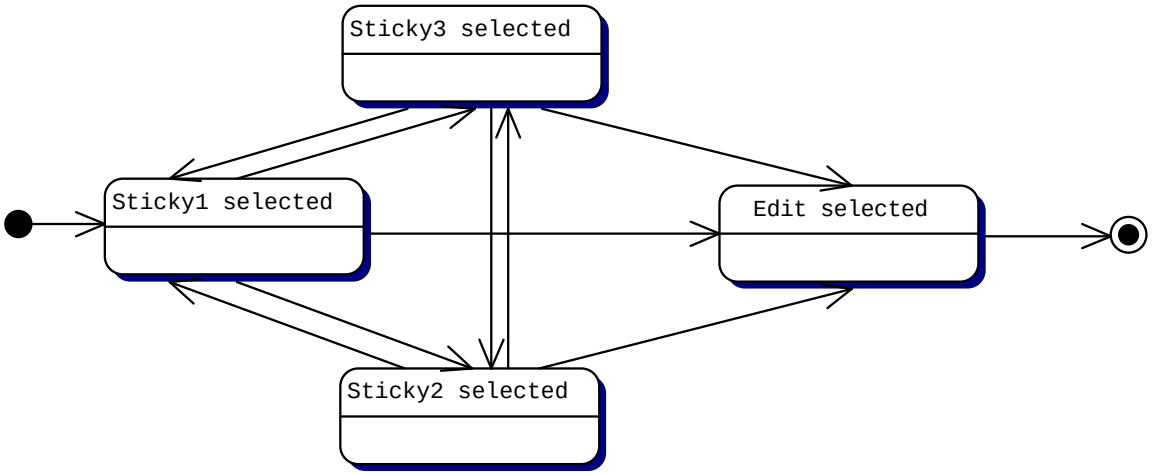


Figure 5: The DFA representing the Controller

Mapping section in CUIML

Inside the `<mapping>` section we define the `<hci>` for VRML and HTML. First we look at the mapping for the HTML view, then we examine the VRML part. The code fragment shows, how the "next" component is mapped to a link

```
<mapping>
  <hci type="VRML,HTML">
    <presentation name="HTML">
      <component name="next" maps-to="link">
        <attribute name="linkLabel" value="Next"/>
      </component>
```

Then inside the `<logic>` section, the events that should be fired and the event adapter to be used are specified. For our example we used Pushlets (see [16]) as an event mechanism, but it would also be possible to use CORBA or any other middleware.

```
<logic>
  <component class="ChoiceSelected" maps-to="onClicked"> </component>
  <component class="OutEvent" maps-to="pushlet.fireEvent"> </component>
  <component name="nextSelected"> <attribute name="param" value="next"/> </component>
</logic>
```

Now let us take a look at the VRML mapping. The `sticky` component is mapped to a VRML description, with the default colour being green:

```
<presentation type="VRML">
  <component class="Sticky" maps-to="VRMLsticky.wrl"> </component>
  <component name="sticky1"> <attribute name="colour" value="green"/> </component>
```

The reaction to inEvents that are fired by the controller are specified. First we define that all incoming events are caught by the `eventReceived` method of an EAI manipulator. Then the manipulations for the event "sticky1 selected" are listed.

```
<logic>
  <component class="InEvent" maps-to"EAI.eventReceived()"> </component>
  <component name="sticky1 selected">
    <action>
      <property part-name="sticky1" name="colour" >red</property>
      <property part-name="sticky2" name="colour" >green</property>
      <property part-name="sticky3" name="colour" >green</property>
    </action>
  </component>
</logic>
```

6 Conclusion

Development of an IDE: CUIML proves to be sufficient for the description of multimodal HCIs. The CUIML descriptions can become quite lengthy and thus it would be desirable to have an IDE that encapsulates CUIML. S. Riss describes in his paper ([10]) how the controller part can be configured by a graphical development environment. The IDE for the other parts of CUIML could be a future project. The main advantage of an IDE would be that HCIs can be designed quickly and without knowing CUIML.
Security: As the CUIML components sent to the client have access to all client ressources, a security concept has to be developed.
More powerful controller: For our simple examples a DFA was sufficient as underlying structure for the controller. But with more complex examples that introduce problems like time-outs and concurrency, a more complex structure will be needed. One possibility would be Petri Nets or even Colored Petri Nets that are even more powerful.

References

[1] DWARF project homepage. http://www.augmentedreality.de.

[2] Ronald T. Azuma. A Survey of Augmented Reality. *Presence*, 6(4):355–385, August 1997.

[3] Steve Mann. Definition of a wearable computer. http://wearcam.org/wearcompdef.html.

[4] L. Nigay and J. Coutaz. A design space for multimodal systems - concurrent processing and data fusion. In *INTERCHI '93 - Conference on Human Factors in Computing Systems*, Amsterdam, 1993. Addison Wesley.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[6] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An Appliance Independent XML User Interface Language. http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html.

[7] UIML 2.0 draft specification. http://www.uiml.org/specs/docs/uiml20-17Jan00.pdf.

[8] W. van Biljon. Extending Petri Nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies*, 28(4):437–455, 1988.

[9] A. Ohta and T. Hisamura. On the Modeling Power of Free Firing Petri Nets with Single-Weighted Arcs(in Japanese). *Trans. Society of Instrument and Control Engineers*, 30(10):1269–1270, October 1994.

[10] S. Riss. *Development and Implementation of a prototype for the description of Workflows with Augmented Reality (AR) by means of a XML-dialect and execution on a webbased Augmented Reality-System.* Diploma Thesis, Technische Universität München, to be published.

[11] XForms. http://www.w3.org/MarkUp/Forms/.

[12] Mozquito. http://www.mozquito.com.

[13] U. Neumann and A. Majoros. Cognitive, Performance and System Issues for Augmented Reality Applications in Manufactoring and Maintenance. In *Proc. of IEEE Virtual Reality Annual International Symposium*, pages 4–11, 1998.

[14] XSL specification. http://www.w3.org/Style/XSL/.

[15] DOM specification. http://www.w3.org/DOM/.

[16] Pushlet homepage. http://www.fluidiom.com:8080/.