A Graphical UIDL Editor for Multimodal Interaction Design Based on SMUIML

Dumas, Bruno; Signer, Beat; Lalanne, Denis

Publication date:
2011

Document Version
Peer reviewed version

Link to publication
Citation for published version (HARVARD):
ABSTRACT
We present the results of an investigation on software support for the SMUIML multimodal user interaction description language. In particular, we introduce a graphical UIDL editor for the creation of SMUIML scripts. The presented graphical editor is fully based on SMUIML for the underlying representation of data as well as for the dialogue modelling. Due to the event-centered nature of SMUIML, the representation of the multimodal dialogue modelling in the graphical SMUIML dialogue editor has been realised via a state machine. A real time graphical debugging tool is also integrated in the editor. Compared to existing multimodal dialogue editors, the SMUIML graphical editor offers a dual graphical and textual editing as well as a number of operators for the temporal combination of modalities.

Author Keywords
Multimodal Interaction, UIDL, Graphical Editor, SMUIML, HephaisTK

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Graphical user interfaces, Prototyping, Theory and methods

INTRODUCTION
Multimodal interfaces aim to improve the communication between humans and machines by making use of concurrent communication channels or modalities. They have been shown to bring more comfort and better expressivity to users. Nevertheless, multimodal interfaces are difficult to realise due to a number of reasons. First, multimodal interfaces are typically composed of a number of state of the art recognition technologies, such as speech recognition or gesture identification systems based on pattern matching. Typically, developers have to master a number of these state of the art recognisers for different modalities in order to create advanced multimodal interfaces. Second, a combination of input for the same modality can lead to ambiguous interpretations based on factors such as the ordering of input events, the delay between events, the context of use or specific user profiles. Fusion algorithms that take adaptation into account are therefore required. Last but not least, multimodal human-machine dialogue modelling is desirable in order to facilitate the development of complex multimodal interfaces.

A potential track of investigation to address the challenges introduced by multimodal interaction design is the use of a modelling language in combination with a multimodal framework and development environment. A multimodal user interface description language (UIDL) forms the key element in such an approach. The UIDL is used to define the behaviour of the multimodal framework, to perform the dialogue modelling and as the underlying format for the GUI development environment. A multimodal user interface description language is typically situated at the Abstract User Interface (AUI) layer. Furthermore, software support for the UIDL is provided for the definition, modelling or interpretation of user interfaces descriptions.

We present our explorations of such a language-based approach in the context of the Synchronized Multimodal User Interfaces Modelling Language (SMUIML) and the corresponding software support. In particular, we present a graphical UIDL editor for SMUIML and discuss its support for designing multimodal interactions. The graphical editor offers an alternative to the pure text-based editing of our XML-based language which is often more tedious and can easily lead to errors. We start by discussing related work in the context of modelling languages as well as graphical editors for multimodal interaction design. We then introduce the SMUIML language as well as some results from our research on the language design for modelling multimodal interaction. This is followed by a description of the different supportive software components for the SMUIML language with a particular focus on the graphical UIDL editor. After an overview of the planned future work, we provide some conclusions.

RELATED WORK
There have been a number of formal language approaches for the creation of multimodal interfaces over the last decade. Some of these approaches are positioned in the context of a
multimodal web, propagated by the World Wide Web Consortium’s (W3C) Multimodal Interaction Activity and its proposed multimodal architecture\textsuperscript{1}. This theoretical framework describes the major components involved in multimodal interaction, as well as potential or existing markup languages to be used to relate those different components. Many elements described in this framework are of practical interest for multimodal HCI practitioners, such as the W3C EMMA markup language\textsuperscript{2} or modality-focused languages including VoiceXML\textsuperscript{3}, EmotionML or InkML\textsuperscript{4}. The W3C framework inspired Katsurada et al.\textsuperscript{[9]} for their work on the XISL XML language. XISL focuses on the synchronisation of multimodal input and output, as well as dialogue flow and transition. Araki et al.\textsuperscript{[1]} propose the Multimodal Interaction Markup Language (MIML) for the definition of multimodal interactions. A key characteristic of MIML is its three-layered description of interaction, focusing on interaction, tasks and platform. Ladry et al.\textsuperscript{[11]} use the Interactive Cooperative Objects (ICO) notation for the description of multimodal interaction. This approach is closely bound to a visual tool enabling the editing and simulation of interactive systems, while being able to monitor system operations at a low level. Stanciulescu et al.\textsuperscript{[15]} followed a transformational approach for the development of multimodal web user interfaces based on UsiXML. Four steps are necessary to go from a generic model to the final user interface. One of the main features of their work is a strong independence from the available input and output channels. A transformational approach is also used in Teresa XML by Paterno et al.\textsuperscript{[13]}. Finally, at a higher level of modeling, NiMMIT\textsuperscript{2} is a graphical notation associated to a language used for expressing and evaluating multimodal user interaction.

Graphical editors for the definition of multimodal dialogues can broadly be separated into two families. These two families differ in the way how a dialogue is represented which is often driven by the underlying architecture. On the one hand, stream-based architectures favour a direct representation of data streams, with building blocks consisting of processing algorithms that are applied to the streams in a sequential manner. There has been a trend for graphical editors for stream-based multimodal architectures in the past few years. Petshop for ICO\textsuperscript{11}, Squidy\textsuperscript{10} or Skemmi\textsuperscript{12} for Open-Interface are examples of graphical editors for stream-based architectures. On the other hand, event-driven architectures result in a state machine-based representation of the multimodal human-machine dialogue. In this category, fewer examples exist for the representation of multimodal interaction, the most prominent one being IMBuilder from Bourguet\textsuperscript{4}.

Note that the graphical editors introduced in this section have all been built from scratch and they are not based on a previously defined formal language, with Petshop for ICO forming the only exception.

\textsuperscript{1}http://www.w3.org/TR/mmi-arch/
\textsuperscript{2}http://www.w3.org/TR/emma/
\textsuperscript{3}http://www.w3.org/TR/voicexml20/
\textsuperscript{4}http://www.w3.org/TR/InkML/
a description language: a highly formal language approach that perfectly fits for configuring a tool, a loosely formal language approach which is good for communicating the details of an application and a "middle" approach focussing on the modelling. Along these three approaches, a formal language can also be used as a learning tool as shown in Figure 2. As a learning tool, a modelling language helps teachers in communicating the features of a particular application domain to their students.

In [8] we presented 9 guidelines for a multimodal description language. These guidelines should be used as design tools or as language analysis criteria:

- Abstraction levels
- Modelling the human-machine dialogue
- Adaptability to context and user (input and output)
- Control over fusion mechanism
- Control over time synchronicity
- Error handling
- Event management
- Input and output sources representation
- Finding the right balance between usability and expressiveness

SOFTWARE SUPPORT FOR SMUIML

As a language defining a full model of multimodal human-machine events and dialogues, SMUIML already provides modelling capabilities as well as a reflection basis. However, the language shows its true potential when linked to a range of different supportive software solutions. In the following, we briefly introduce software support within SMUIML for interpretation and then discuss the latest software addition in the form of a graphical editor for designing multimodal human-machine dialogues.

The HephaisTK Framework

A framework for the creation of multimodal interfaces has been developed in our research lab. This framework, called HephaisTK, uses SMUIML as its scripting language. Note that it is out of the scope of this paper to provide a detailed description of HephaisTK. However, the interested reader can find more details about HephaisTK in [7].

Basically, a description created in SMUIML with the structure shown in Listing 1 is used to configure the HephaisTK framework. The <recognizers> part indicates which recognisers have to be loaded by the framework. It further provides some high-level parameters such as whether a speech recogniser is able to recognise different languages. The <triggers> are directly bound to the different fusion algorithms provided by HephaisTK. The <actions> part defines the semantics to be used when communicating fusion results to a client application. Last but not least, the <dialog> part of SMUIML is used in HephaisTK for a number of specific goals.

First and foremost, by providing a description of the human-machine dialogue flow, the HephaisTK DialogManager agent stays in a consistent state with the client application. The clear separation of the SMUIML <dialog> into transitions and contexts allows the different triggers to be enabled or disabled depending of the current context. Since only a subset of triggers has to considered in a given context, the load on the recognisers is reduced and the overall recognition rate is improved. The <dialog> part of SMUIML also helps with the instantiation of the different fusion algorithms present in HephaisTK. In the case of the Hidden Markov Model-based fusion algorithm that is integrated in HephaisTK, the definition of the human-machine dialogue in SMUIML is also used to generate a set of all expected trigger input sequences. The set of all expected sequences is then injected into a series of Hidden Markov Models (one per context of use), so that the fusion engine is ready to be used when launching the HephaisTK framework.

In the context of the HephaisTK framework, SMUIML is applied at multiple levels: at the multimodal dialogue description level, at the recogniser launch and parameterisation level as well as at the fusion engine instantiation level. Thus, SMUIML is typically used during the later stages of multimodal interface development, i.e. at the system design and runtime stages.

```
Listing 1. Basic layout of a SMUIML script
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_desc client="client_app">
    <recognizers>
      <!-- ... -->
    </recognizers>
    <triggers>
      <!-- ... -->
    </triggers>
    <actions>
      <!-- ... -->
    </actions>
    <dialog>
      <!-- ... -->
    </dialog>
  </integration_desc>
</smuiml>
```

![Diagram](image.png)

Figure 2. Four purposes of modelling languages.
The SMUIML Graphical Editor

As SMUIML is a language that is derived from the XML metalanguage, a standard text editor is sufficient for creating SMUIML documents. Even if the language has been proven to be expressive in a qualitative study [8], the editing of “raw” XML easily leads to errors which can only be identified when interpreting a SMUIML document at runtime. Other issues with the text-based editing of SMUIML scripts include the lack of an explicit representation of the relationships between different elements as well as the difficulty to produce and maintain an accurate mental model of complex dialogue scenarios. Furthermore, the necessity of having to learn a new language can represent a serious challenge for some users. In order to overcome these shortcomings, we have developed a graphical editor for the definition of new SMUIML scripts.

The goal of our SMUIML graphical editor was to provide a usable and expressive tool for creating SMUIML scripts for HephaisTK to developers who are not fully proficient with multimodal interfaces. The dialogue editor offers a graphical representation of SMUIML-encoded multimodal human-machine dialogues. Furthermore, it supports the creation of sets of actions and triggers and can be used to generate the Java configuration file with all calls related to the SMUIML script. As shown later, the graphical representation of a multimodal dialogue closely follows the SMUIML logic presented in the previous section. The SMUIML graphical editor tool was created by using the Eclipse[^5] open development platform as a basis. Eclipse is widely used and known among development teams and its usage provides expert users with a set of well-known interface elements. The SMUIML graphical tool itself was created using the Graphical Editing Framework (GEF)[^6] and the Eclipse Modeling Framework (EMF)[^7].

The main window of the graphical editor is shown in Figure 3. The central part of the tool is dedicated to the actual dialogue representation. As stated before, the multimodal human-machine dialogue in SMUIML typically takes the form of a state machine. A graphical representation of this state machine is used to depict the multimodal dialogue in

[^5]: http://www.eclipse.org
[^6]: http://www.eclipse.org/gef/
[^7]: http://www.eclipse.org/modeling/emf/
the graphical editor. Note that the editor also provides access to a textual version of the SMUIML script that is currently edited. Any changes done either in the graphical or the textual representation of a given SMUIML script are reflected in the other representation. The checking of errors is achieved in real-time for both the graphical and textual representations.

On the right-hand side of the window are a set of toolboxes and most of them are related to the different parts of a typical SMUIML file. The Palette toolbox presents the basic building blocks for creating the dialogue state machine, in particular states and transitions. The selection tool is also presented in the Palette toolbox. The Operators toolbox offers some operators for the combination of different modalities as defined in the SMUIML specification. Note that these operators are tightly linked to the CARE properties [6]. Seq and corresponds to sequential-constrained complementarity, Par and to sequential-unconstrained complementarity, Seq or to equivalence and Par or to redundancy. The next toolbox is devoted to input triggers and contains a list of all triggers defined for a given application, as well as a New trigger button to create new triggers. Last but not least, the Actions toolbox lists all actions that have been defined for a given application and also provides a New action button. Triggers and actions are added to these toolboxes as they are defined along the creation of the multimodal dialog.

Figure 4 shows the graphical presentation of Richard Bolt’s infamous “put that there” example [3] in the graphical editor whereas the corresponding textual SMUIML specification is shown in Listing 2. In the same way as in Figure 3, states (contexts) are visualised as blue ellipses. Along the actions taken by users, HephaisTK might stay in the same context or switch to another context of use. In the “put that there” example, there is only as single start context with a transition starting and pointing to it. This transition contains the overall description of the “put that there” action. It asks for five different input triggers in order that the action will be fired. Namely, three speech triggers (“put”, “that” and “there”), as well as two pointing event triggers. Furthermore, three temporal combination operators are used in this example. The main transition uses a Seq and operator asking for a “put” speech trigger to be followed by a “that” and “there” sub-event. Those two sub-events use a Par and combina-

<table>
<thead>
<tr>
<th>Listing 2. SMUIML description of the “put that there” example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;context name=&quot;start&quot;&gt;</td>
</tr>
<tr>
<td>&lt;transition leadtime=&quot;1500&quot;&gt;</td>
</tr>
<tr>
<td>&lt;seq_and&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;put_trigger&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;transition&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;that_trigger&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;object_pointed_event&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;par_and&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;there_trigger&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;object_pointed_event&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;/par_and&gt;</td>
</tr>
<tr>
<td>&lt;/transition&gt;</td>
</tr>
<tr>
<td>&lt;par_and&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;there_trigger&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;trigger name=&quot;object_pointed_event&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;/par_and&gt;</td>
</tr>
<tr>
<td>&lt;/transition&gt;</td>
</tr>
<tr>
<td>&lt;seq_and&gt;</td>
</tr>
<tr>
<td>&lt;result actions=&quot;put_that_there_action&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;result context=&quot;start&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;/seq_and&gt;</td>
</tr>
<tr>
<td>&lt;/context&gt;</td>
</tr>
</tbody>
</table>

The graphical editor also contains an integrated debugging tool. This debugging tool is launched along the client application and indicates in real-time the context the HephaisTK framework is currently in. It also highlights the transition that lead from the previous context to the current one. In the example illustrated in Figure 5, the application starts in the start context. An RFID reader that is connected to the framework detects a tagged music album and transmits the information. Based on the trigger_album trigger, a transition is fired and the application moves to the registereccd state and starts playing the music album. Then the user utters a stop command and holds at the same time an RFID tag labelled “stop” close to the RFID reader. This simultaneous action fires the transition going from the registereccd context back to the start context. As illustrated in this example, the graphical debugging tool al-

Figure 4. Graphical description of “put that there”
allows developers to visually analyse the behaviour of an application in real-time.

Figure 5. Graphical debugging tool

FUTURE WORK
While the presented SMUIML graphical editor looks promising and offers some features not available in other graphical editors for creation of multimodal interfaces, we plan to perform a detailed evaluation of the presented solution in the near future. First, we are going to evaluate the usability of the presented graphical editor for multimodal interaction design. However, in addition to the usability of the graphical editor, we also plan to evaluate the expressiveness of the presented approach. It is not enough to guarantee an effective and simple definition of multimodal interactions based on the graphical editor. We also have to ensure that the editor and the underlying SMUIML language are expressive enough to describe multimodal interactions of arbitrary complexity.

Another important future direction is to support the flexible adaptation of multimodal interfaces. The idea is to no longer have a fixed combination of modalities, but rather provide a context-dependent adaptation of multimodal user interfaces. This can either be achieved by extending the SMUIML language with the necessary concepts or by introducing another language for the adaptation of the multimodal interaction. In this view, the abstract user interface definition would rely on SMUIML while the concrete, context-dependant user interface specification would be based on a new language. The final user interface would be realised by HephaisTK [5].

Last but not least, multimodal interfaces should not be treated isolated from the underlying content that they are providing access to. Today’s document formats do often not offer access to specific semantic subparts and media types embedded in a given document [14]. If we are able to model and get access to information at a finer granularity level, the different modalities of interaction can also be associated with specific parts of a document. Within the MobiCraNT project, we are currently investigating innovative mobile cross-media applications. As part of this research effort, we are developing a new fluid cross-media document model that offers the fine grained access to information described earlier. By no longer defining the modality of interaction at the document but rather at the level of cross-media information units, we aim for a more flexible solution where the structure of a document as well as the multimodal interface to interact with its parts is adapted in a content-dependent manner.

CONCLUSION
We have presented our exploration on software support for multimodal UIDL based on the SMUIML multimodal dialogue modelling language. Thereby, we focussed on two particular software components: the HephaisTK framework which is used to interpret the SMUIML language (at the runtime stage) and the SMUIML graphical editor for the graphical design of multimodal interaction dialogues (at the system design stage). The SMUIML graphical editor aims to provide a user-friendly way to create multimodal applications based on HephaisTK and SMUIML. Compared to other graphical dialogue editors, our solution supports temporal constraints and a number of operators for the combination of multiple modalities. While these concepts already form part of the underlying SMUIML language, the graphical editor makes these concepts accessible via a user-friendly interface. Users further have the possibility to freely switch between the graphical and textual dialogue representation. The presented graphical editor for SMUIML further introduces a number of usability-oriented features such as automatic layouting, the clear identification of input modalities via specific icons as well as the possibility to customise various features of the graphical editor. Last but not least, the SMUIML graphical editor offers an integrated debugging tool supporting developers in analysing the real-time behaviour of their applications.

ACKNOWLEDGMENTS
The authors wish to thank Saïd Mechkour for his work on the SMUIML graphical editor. The work on HephaisTK and SMUIML were funded by the Hasler Foundation in the context of the MeModules project and by the Swiss National Center of Competence in Research on Interactive Multimodal Information Management via the NCCR IM2 project. Bruno Dumas is supported by the MobiCraNT project in the context of the Strategic Platforms programme from the Brussels Institute for Research and Innovation (Innoviris).
REFERENCES


