A Graphical UIDL Editor for Multimodal Interaction Design Based on SMUIML

Bruno Dumas
WISE Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bdumas@vub.ac.be

Beat Signer
WISE Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bsigner@vub.ac.be

Denis Lalanne
DIVA Group
Université de Fribourg
Boulevard de Pérolles 90
1700 Fribourg, Switzerland
denis.lalanne@unifr.ch

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 representation of the underlying 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. The editor further offers a real-time graphical debugging tool. 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 (GUI), 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 increase comfort and offer 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 pattern matching-based gesture recognition. 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.

The challenges introduced by multimodal interaction design can potentially be addressed by using a modelling language in combination with a multimodal framework and development environment. A multimodal user interface description language (UIDL) forms the key element of 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 interface 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 purely text-based editing of scripts in our XML-based language, which is often tedious and can easily lead to errors. This graphical editor furthermore focuses on how to express complex temporal relations between input modalities. 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 and 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

Over the last decade, there have been a number of formal language approaches for the creation of multimodal interfaces. 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[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 these components. Many elements described in this framework, such as the W3C EMMA markup language[2] or modality-focused languages including VoiceXML[3], EmotionML and InkML[4], are of practical interest for multimodal HCI practitioners. The W3C framework inspired Katsurada et al. [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. [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. [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. [17] followed a transformational approach for the development of multimodal web user interfaces based on UsiXML. Four steps are necessary to get 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. [13]. DISL [14] was created as a language for specifying a dialogue model which separates multimodal interaction and presentation components from the control model. Finally, at a higher level of modelling, NiMMiT [2] is a graphical notation associated with a language used to express and evaluate multimodal user interaction. An analysis of multimodal interaction modelling languages can also be found in [16].

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. In the past few years, there has been a trend for graphical editors for stream-based multimodal architectures. Petshop for ICO [11], Squidy [10] or Skemmi [12] for OpenInterface are examples of these types 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 [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.

[1]http://www.w3.org/TR/mmi-arch/

[2]http://www.w3.org/TR/emma/

[3]http://www.w3.org/TR/voicexml20/

[4]http://www.w3.org/TR/InkML/

THE SMUIML LANGUAGE

SMUIML stands for *Synchronized Multimodal User Interaction Modelling Language*. As the name implies, SMUIML aims to offer developers a language to describe multimodal interaction and define the used modalities in an easy-to-read and expressive way. The language can further be used to describe the recognisers associated with a given modality, the human-machine dialogue modelling, the various events associated with these dialogues and the way these different events can be temporally synchronised[5]. SMUIML was designed to be as simple as possible and is targeting usability. In order to minimise the verbosity of SMUIML, we decided not to rely on existing standard multimodal interaction languages.

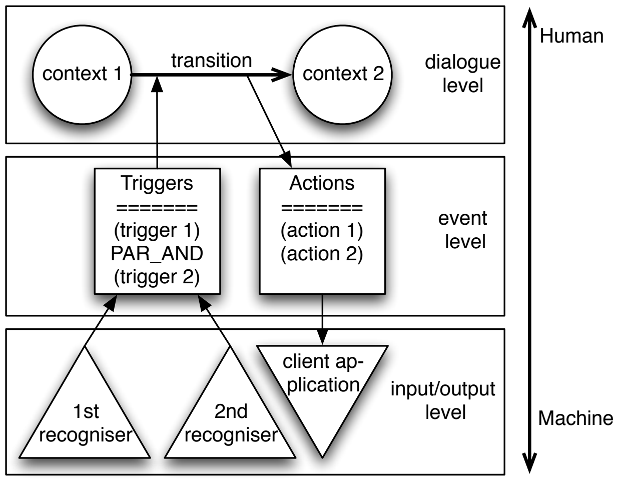

Figure 1. The three levels of SMUIML

The SMUIML language is divided into the three abstraction layers shown in Figure 1. The lowest level details the different modalities which are then used in the context of an application, as well as the particular recognisers to be used to access the different modalities. The middle level addresses input and output events. Input events are called *triggers* and output events *actions*. Triggers are defined per modality which means that they are not directly bound to specific recognisers and they can express different ways to trigger a particular event. For example, a speech trigger can be defined in such a way that the words "clear", "erase" and "delete" will all lead to the same event. Actions are the messages that the framework sends to the client application. The top level of abstraction describes the actual human-machine dialogue by means of defining the contexts of use and interweaving the different input events and output messages between those contexts. The resulting human-machine

[5]An XML Schema as well as some SMUIML examples can be found at: http://sourceforge.net/projects/hephaistk/

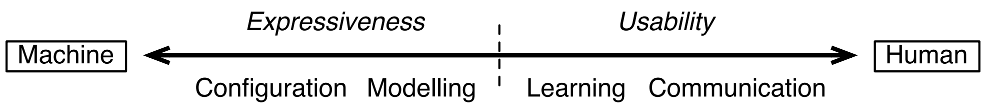

Figure 2. Four modelling language purposes (from machine-oriented to human-oriented) with respect to expressiveness and usability.

dialogue description is a series of "contexts of use", with transitions between these different contexts. Therefore, the description of the multimodal human-machine dialogue in SMUIML has an implicit representation as a state machine, similar to Bourguet's IMBuilder [4]. The combination of modalities is defined based on the CARE properties [6] as well as on the (non-)sequentiality of input triggers. As shown in Listing 1, the three abstraction levels are directly reflected in the basic structure of the language.

The spectrum of multimodal dialogue description language users, on a scale from usability to expressiveness, was presented in [8]. Through various workshops, informal discussions with colleagues and students and a study of the current state of the art, we envisioned three types of approaches for a description language: a highly formal language approach that perfectly fits for configuring a tool, a less 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 (see Figure 2) helping 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

SMUIML enables the definition of a full model of multimodal human-machine events and dialogues by providing 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 the 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

The HephaisTK framework which supports the creation of multimodal interfaces based on the SMUIML scripting language has been developed in our research lab. 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 SMUIML `<dialog>` part is used for a number of specific goals in HephaisTK.

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>
```

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. This set of expected sequences is then injected into a series of Hidden Markov Models (one per context of use) in order to have the fusion engine ready to be used when launching the HephaisTK framework.

The SMUIML language is applied at multiple levels in the context of the HephaisTK framework: at the multimodal

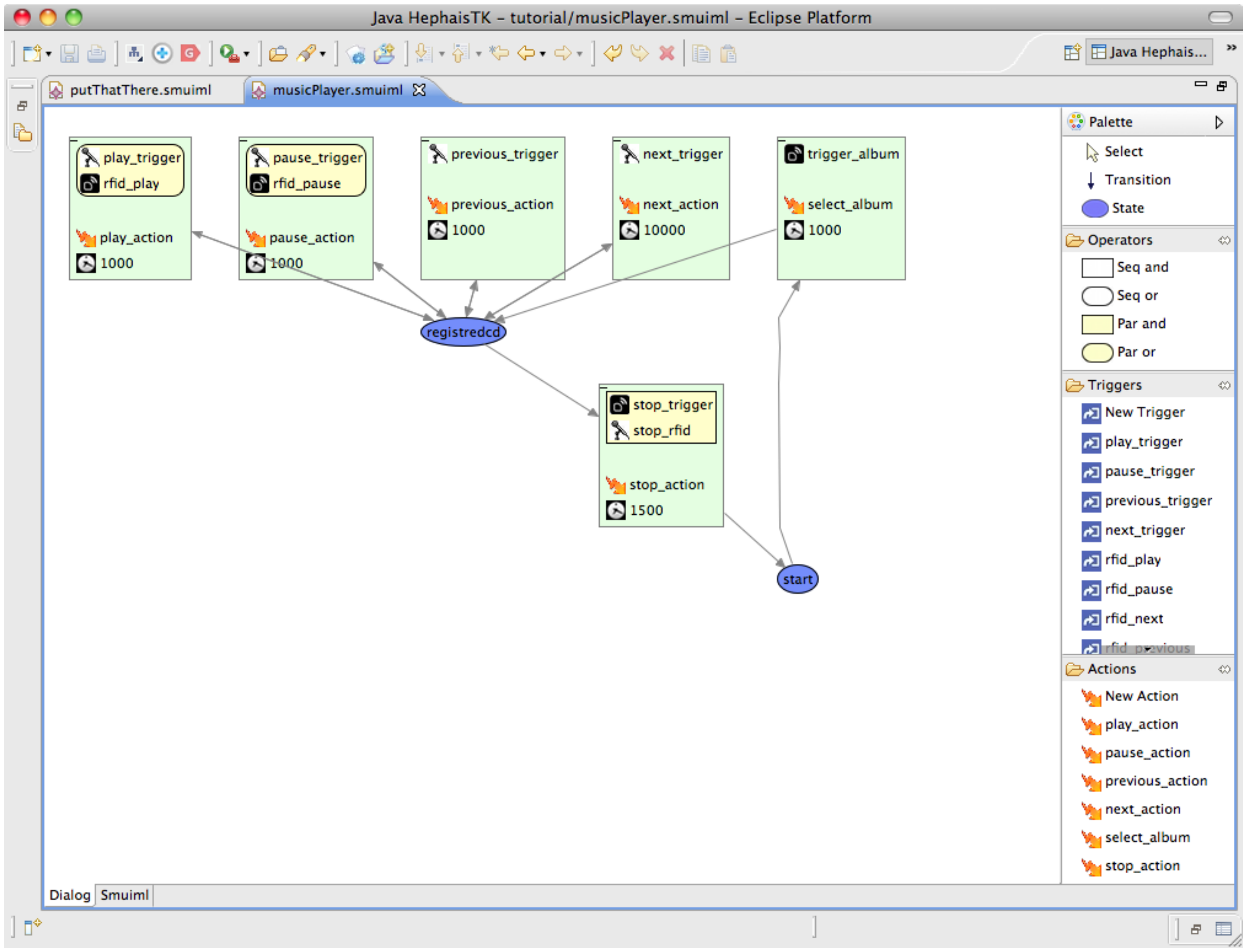

Figure 3. The SMUIML graphical editor with an example dialogue defining the behaviour of a music player application.

dialogue description level, at the recogniser launch and parameterisation level as well as at the fusion engine instantiation level. SMUIML is typically used during the later stages of multimodal interface development, including the *system design* and *runtime* stages. Note that it is out of the scope of this paper to provide a full description of HephaisTK but further details can be found in [7].

The SMUIML Graphical Editor

The SMUIML language is derived from the XML metalanguage and 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 documents can easily lead to errors that are only identified when interpreting a SMUIML script 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 may represent a major 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 developers, who are not fully proficient with multimodal interfaces, a usable and expressive tool for creating SMUIML scripts. 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 a Java configuration with all the calls related to the SMUIML script. The graphical representation of a multimodal dialogue follows the SMUIML logic presented in the previous section. The SMUIML graphical editor has been created based on the Eclipse[6] open development platform. Eclipse is widely used among development teams and provides a set of well-known interface elements. The SMUIML graphical tool itself was developed using the Graphical Editing Framework (GEF)[7] and the Eclipse Modeling Framework (EMF)[8].

[6]http://www.eclipse.org
[7]http://www.eclipse.org/gef/
[8]http://www.eclipse.org/modeling/emf/

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 earlier, the multimodal human-machine dialogue in SMUIML is represented via a state machine. A graphical representation of this state machine is used to depict the multimodal dialogue in the graphical editor. Note that the editor also provides access to a textual version of the SMUIML script that is currently edited. Any changes that are done either in the graphical or the textual representation are immediately reflected in the other representation. For both, the graphical and textual representation, there exists real-time error checking.

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 also forms part of the `Palette` toolbox. The `Operators` toolbox offers some operators to combine different modalities as defined in the SMUIML specification. These operators are tightly linked to the CARE properties [6]. `Seq and` corresponds to sequential-constrained complementarity, `Par and` to sequential-unconstrained complementarity, the `Seq or` operator 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 when they are defined as part of a multimodal dialogue.

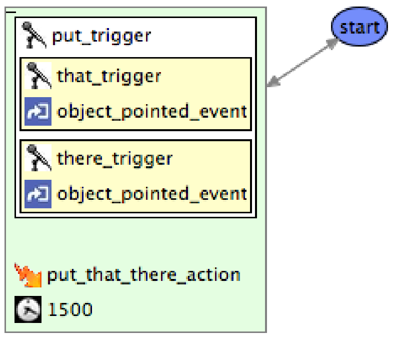

Figure 4. Graphical description of "put that there"

Figure 4 shows the graphical representation of Bolt's "put that there" example [3] in the graphical editor with states (contexts) visualised as blue ellipses. The corresponding textual SMUIML specification is shown in Listing 2. Based on 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. The two sub-events use a `Par and` combination operator, meaning that there should be speech and pointing triggers but without any sequential constraint. This implies that a user can perform the commands in different orders, such as "put that" `[point1] [point2]` "there" or "put" `[point1]` "that there" `[point2]` and both sequences will be correctly recognised. Finally, the transition specifies a time window of 1500 milliseconds for the whole command as well as an action (message to be sent the client application) to be performed if the command has been successfully recognised. In our example, the transition then proceeds to the same `start` context it originated from.

Listing 2. SMUIML description of the "put that there" example

```
<context name="start">
  <transition leadtime="1500">
    <seq_and>
      <trigger name="put_trigger"/>
      <transition>
        <par_and>
          <trigger name="that_trigger"/>
          <trigger name="object_pointed_event"/>
        </par_and>
      </transition>
      <transition>
        <par_and>
          <trigger name="there_trigger"/>
          <trigger name="object_pointed_event"/>
        </par_and>
      </transition>
    </seq_and>
    <result action="put_that_there_action"/>
    <result context="start">
  </transition>
</context>
```

The SMUIML graphical editor has been presented to two expert users in order to achieve a small expert review. This review by experts lead to a number of changes to improve the editor's usability. The modality of each trigger is now indicated by means of an icon. The `start` context which is the only mandatory context in a given SMUIML script is visualised in a slightly different colour to denote its special status compared to other contexts. Finally, users have the possibility to change the colour of connections, contexts or transitions in order to best suit their preferences.

The graphical editor also contains an integrated debugging tool. This debugging tool is launched with the client application and provides a real-time visualisation of the context the HephaisTK framework is currently in. It also highlights the transition leading from the previous context to the current one. In the example illustrated in Figure 5, the application starts in the `start` context. A Radio Frequency Identification (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 `registeredcd` state and starts playing the music album. The user then executes a `stop` command and, at the same time, holds a "stop" labelled RFID tag close to the RFID reader. This simultaneous action fires the transition going from the `registeredcd`

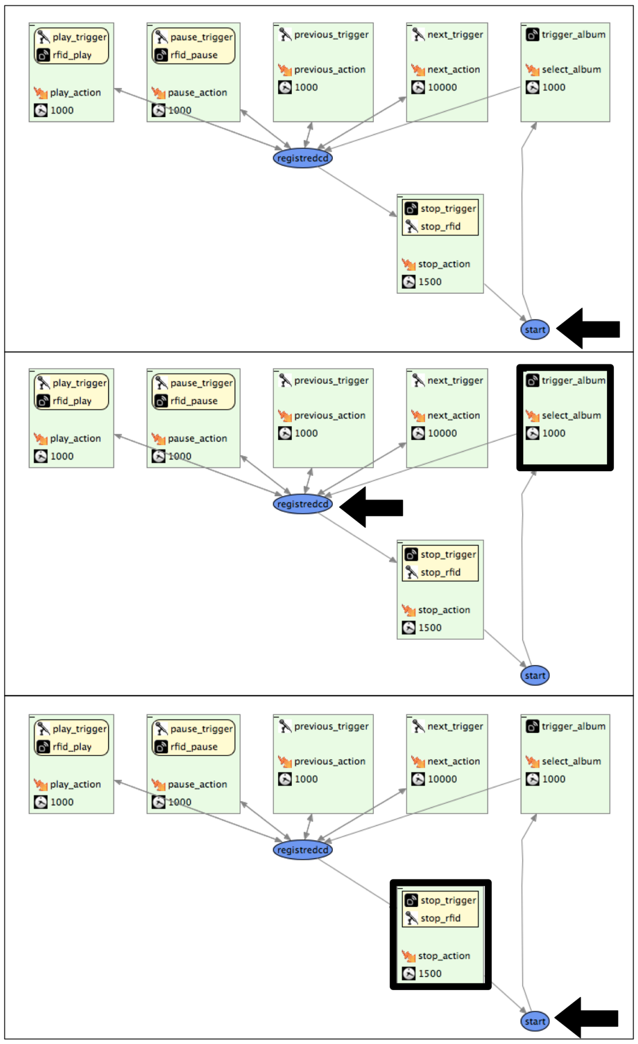

Figure 5. The graphical debugging tool with three different steps going from the `start` context to `registeredcd` and back again.

context back to the `start` context. As illustrated in this example, the graphical debugging tool allows developers to visually analyse the application behaviour in real-time.

FUTURE WORK

While the presented SMUIML graphical editor looks quite promising and offers some features not available in other graphical editors for 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 by asking developers to express a number of multimodal interaction use cases via the tool. In addition, we 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. This study could be a starting point for tackling a more general question: *to what extent do finite state machine-based approaches or event stream-based approaches best represent multimodal human-machine dialogues?*

Another important future direction is to support the flexible adaptation of multimodal interfaces [18]. 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 require the definition of a new language. The final user interface could be realised by HephaisTK [5].

This new language for flexible multimodal interface adaptation could then be used to provide innovative document interfaces. Today's document formats often provide no access to specific semantic subparts or embedded media types [15]. However, if we would be able to get access to these document subparts, specific embedded media types could be associated with different modalities of interaction. Within the MobiCraNT[9] 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 and investigate how SMUIML, in combination with a context-dependant user interface specification language, could be used to provide multimodal access to such a fluid document model.

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 final, 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 SMUIML graphical editor further addresses a number of usability-oriented issues 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 application behaviour.

[9]http://soft.vub.ac.be/mobicrant/

ACKNOWLEDGMENTS

The authors wish to thank Saïd Mechkour for his work on the SMUIML graphical editor. The work on HephaisTK and SMUIML has been 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 MobiCraNT, a project forming part of the Strategic Platforms programme by the Brussels Institute for Research and Innovation (Innoviris).

REFERENCES

1. M. Araki and K. Tachibana. Multimodal Dialog Description Language for Rapid System Development. In *Proc. of the 7th SIGdial Workshop on Discourse and Dialogue*, pages 109–116, Sydney, Australia, July 2006.
2. J. D. Boeck, D. Vanacken, C. Raymaekers, and K. Coninx. High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT. *Journal of Virtual Reality and Broadcasting*, 4(2), September 2007.
3. R. A. Bolt. "Put-that-there": Voice and Gesture at the Graphics Interface. In *Proc. of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '80)*, pages 262–270, Seattle, USA, July 1980.
4. M.-L. Bourguet. A Toolkit for Creating and Testing Multimodal Interface Designs. In *Adjunct Proc. of the 15th Annual Symposium on User Interface Software and Technology (UIST 2002)*, Paris, France, October 2002.
5. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(3):289–308, 2003. Computer-Aided Design of User Interface.
6. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties. In *Proc. of the 5th International Conference on Human-Computer Interaction (Interact 1995)*, Lillehammer, Norway, June 1995.
7. B. Dumas, D. Lalanne, and R. Ingold. HephaisTK: A Toolkit for Rapid Prototyping of Multimodal Interfaces. In *Proc. of 11th International Conference on Multimodal Interfaces (ICMI 2009)*, pages 231–232, Cambridge, USA, November 2009.
8. B. Dumas, D. Lalanne, and R. Ingold. Description Languages for Multimodal Interaction: A Set of Guidelines and its Illustration with SMUIML. *Journal on Multimodal User Interfaces: "Special Issue on The Challenges of Engineering Multimodal Interaction"*, 3(3):237–247, February 2010.
9. K. Katsurada, Y. Nakamura, H. Yamada, and T. Nitta. XISL: A Language for Describing Multimodal Interaction Scenarios. In *Proc. of the 5th International Conference on Multimodal Interfaces (ICMI 2003)*, pages 281–284, Vancouver, Canada, November 2003.
10. W. A. König, R. Rädle, and H. Reiterer. Squidy: A Zoomable Design Environment for Natural User Interfaces. In *Proc. of the 27th International Conference on Human Factors in Computing Systems (CHI 2009)*, Boston, USA, April 2009.
11. J.-F. Ladry, P. Palanque, S. Basnyat, E. Barboni, and D. Navarre. Dealing with Reliability and Evolvability in Description Techniques for Next Generation User Interfaces. In *Proc. of the 26th ACM International Conference on Human Factors in Computer Systems (CHI 2008)*, Florence, Italy, April 2008.
12. J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq. An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-the-Shelf Heterogeneous Components. In *Proc. of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009)*, pages 245–254, Pittsburgh, USA, July 2009.
13. F. Paternò, C. Santoro, J. Mäntyjärvi, G. Mori, and S. Sansone. Authoring Pervasive Multimodal User Interfaces. *International Journal of Web Engineering and Technology*, 4(2):235–261, 2008.
14. R. Schaefer, S. Bleul, and W. Mueller. Dialog Modeling for Multiple Devices and Multiple Interaction Modalities. In *Proc. of the 5th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2006)*, pages 39–53, Hasselt, Belgium, October 2006.
15. B. Signer. What Is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse. In *Proc. of the 29th International Conference on Conceptual Modeling (ER 2010)*, pages 391–404, Vancouver, Canada, November 2010.
16. J.-S. Sottet, G. Calvary, J. Coutaz, J.-M. Favre, J. Vanderdonckt, A. Stanciulescu, and S. Lepreux. A Language Perspective on the Development of Plastic Multimodal User Interfaces. *Journal on Multimodal User Interfaces*, 1:1–12, 2007.
17. A. Stanciulescu, Q. Limbourg, J. Vanderdonckt, B. Michotte, and F. Montero. A Transformational Approach for Multimodal Web User Interfaces Based on UsiXML. In *Proc. of the 7th International Conference on Multimodal Interfaces (ICMI 2005)*, pages 259–266, Torento, Italy, October 2005.
18. J. Vanderdonckt, G. Calvary, J. Coutaz, and A. Stanciulescu. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. In *Multimodal User Interfaces*, Signals and Communication Technology, pages 61–84. Springer Berlin Heidelberg, 2008.