A Graphical Editor for the SMUIML Multimodal User Interaction Description Language

Bruno Dumas*[1], Beat Signer*, Denis Lalanne+

*Vrije Universiteit Brussel, Belgium, +University of Fribourg, Switzerland
email addresses: bdumas@vub.ac.be, bsigner@vub.ac.be, denis.lalanne@unifr.ch

Abstract

We present the results of an investigation on software support for the SMUIML multimodal user interaction description language (UIDL). In particular, we introduce a graphical UIDL editor for the creation of SMUIML scripts. The data management as well as the dialogue modelling in our graphical editor is based on the SMUIML language. Due to the event-centered nature of SMUIML, the multimodal dialogue modelling is represented by a state machine in the graphical SMUIML dialogue editor. Our editor further offers a real-time graphical debugging tool. Compared to existing multimodal dialogue editing solutions, the SMUIML graphical editor offers synchronised dual editing in graphical and textual form as well as a number of operators for the temporal combination of modalities. The presented graphical editor represents the third component of a triad of tools for the development of multimodal user interfaces, consisting of an XML-based modelling language, a framework for the authoring of multimodal interfaces and a graphical editor.

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

1. 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 richer

[1]Corresponding author.

expressiveness to users [1]. Nevertheless, due to a number of reasons, multimodal interfaces are difficult to realise. First, multimodal interfaces are normally composed of a number of different recognition technologies such as speech recognition or pattern matching-based gesture recognition. In order to create advanced multimodal interfaces, developers typically have to master a number of these state of the art recognisers for different modalities. 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, the modelling of multimodal human-machine dialogues is desirable in order to develop advanced multimodal interfaces.

The last years have seen the emergence of a number of tools for developing multimodal interfaces. These tools can be classified in three main families. The first family consists of full-fledged frameworks, which manage recognisers for different input modalities, offer data processing and analysis capabilities as well as algorithms for multimodal fusion and fission. Most of these frameworks offer a programmatic API in order that software developers can access their features. The second family of tools is frequently linked to the first one and contains graphical editors for the description of multimodal dialogues. Indeed, quite a few of these tools have been built on top of frameworks or multimodal engines and act as a user-friendly entry point to the underlying API. The third family consists of multimodal user interface description languages which are used to model the multimodal human-machine dialogues. As we discuss in this article, very few approaches considered to link these three families together and most existing solutions are focussing only on one or two families.

We present our exploration of a language-based multimodal interaction design approach in the context of the Synchronised Multimodal User Interfaces Modelling Language (SMUIML) and the corresponding software support. HephaisTK is introduced as a framework for the description of the multimodal human-machine dialogue which makes use of the SMUIML language. Finally, we present a graphical UIDL editor for SMUIML and discuss its support for multimodal interaction design. The graphical editor offers an alternative to the text-based editing of scripts in our XML-based language, which is often tedious and can easily lead to errors. Furthermore, our graphical editor focusses on expressing complex temporal relations between input modalities. In this article, we outline how our approach addresses the de-

velopment of multimodal interfaces by a holistic approach encompassing all three families of tools. We start by discussing related work in the context of modelling languages as well as graphical editors for multimodal interaction design in Section 2. In Section 3, we introduce the SMUIML language and some results from our research on the language design for multimodal interaction modelling. Next, we provide a description of the different supportive software components for the SMUIML language in Section 4, followed by the graphical UIDL editor in Section 5. After outlining potential future directions is Section 6, we provide some concluding remarks.

2. Related Work

2.1. Multimodal Dialogue Description Languages

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. Navarre et al. [2] identified five different categories of UIDLs according to the modelling intrinsic characteristics of their underlying notation or language: *Petri net-based*, *state-based*, *flow-based*, *code-based* and *constraint-based* UIDLs.

Among these categories, a number of UIDLs address the description of multimodal human-machine interaction. The work of Katsurada et al. [3] on the XISL XML language is inspired by the W3C framework. XISL is a typical example of a code-based UIDL and focusses on the synchronisation of multimodal input and output, as well as dialogue flow and transition. Araki et al. [4] proposed the Multimodal Interaction Markup Language (MIML) for the definition of multimodal interactions. A key characteristic of MIML is its three-layered description of interaction, focussing on interaction, tasks and platform. Ladry et al. [5] use the Interactive Cooperative Objects (ICO) notation and Petri nets 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 low-level system

operations. Stanciulescu et al. [6] followed a code-based approach for the development of multimodal web user interfaces based on UsiXML [7]. Four steps are necessary to transform a UsiXML document from a generic model to the final user interface. One of the main features of this approach is the strong independence from the available input and output channels. A transformational approach in multiple steps is also used in Teresa XML by Paterno et al. [8]. DISL [9] was created as a language for specifying a dialogue model which separates multimodal interaction and presentation components from the control model. Finally, NiMMiT [10] is a high-level graphical modelling notation associated with a language used to express and evaluate multimodal user interaction as state transitions. A detailed analysis of multimodal interaction modelling languages can be found in [11].

2.2. Multimodal Authoring Frameworks

Before achieving any work on multimodal UIDLs, researchers sought to create frameworks for realising multimodal interfaces without having to focus on implementation issues such as the management of input sources or low-level data processing. Incidentally, these frameworks evolved to offer advanced features such as fusion algorithms or dialogue modelling. In the following, we introduce the most representative tools in this category. Quickset by Cohen et al. [12] is a speech and pen-based multimodal interface relying on the Open Agent Architecture (OAA) [13], which was used to explore software agent-based multimodal architectures. IMBuilder and MEngine [14] introduced the modelling of multimodal dialogues through a state machine, along with an engine to interpret the dialogue modelling. In their multimodal system, Flippo et al. [15] used the initial explorations of Cohen et al. [12] to create a more generic framework with comparable features. After these initial explorations, the last few years have seen the appearance of a number of fully integrated tools for the creation of multimodal interfaces. Among these tools are comprehensive open source frameworks such as OpenInterface [16] and Squidy [17]. These frameworks share a similar conceptual architecture with different goals. While OpenInterface targets pure or combined modalities, Squidy was created as a particularly effective tool for streams which are composed of low-level data. Finally, in contrast to the linear and stream-based architecture adopted by most other solutions, the Mudra framework [18] adopts a service-based architecture.

2.3. Graphical Multimodal Dialogue Editors

Graphical editors for the definition of multimodal dialogues can broadly be separated into two families. These two families differ in the way 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 towards graphical editors for stream-based multimodal architectures. Petshop for ICO [5], Squidy [17] or Skemmi [19] for OpenInterface and DynaMo [20] 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 [14]. 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.

2.4. Stream-based Versus Event-driven Approaches

As stated before, graphical tools for designing multimodal interfaces can be broadly separated in two families—*stream-based* approaches and *event-driven* approaches. However, we can go further than having such a simple classification since each family of tools is strongly linked to the architecture used by the underlying framework. As an example, consider the Squidy framework [17] which is a perfect representative of the stream-based framework family. A tool such as Squidy typically supports input sources like input devices producing low-level but high-frequency data streams, such as for example a Wii Remote[2] producing streams of accelerometer data or x, y, and z coordinates. As highlighted in Figure 1, a developer using the graphical Squidy editor can then link the data streams to *processing blocks*, which will typically apply filtering or analysis operations to the data. Finally, the processed data stream is forwarded to a client application.

The other family of graphical editors for multimodal dialogue modelling contains event-driven approaches. These approaches typically process low-frequency but decision-level input data. Examples of typical decision-level

[2]http://www.nintendo.com/wii/what-is-wii/

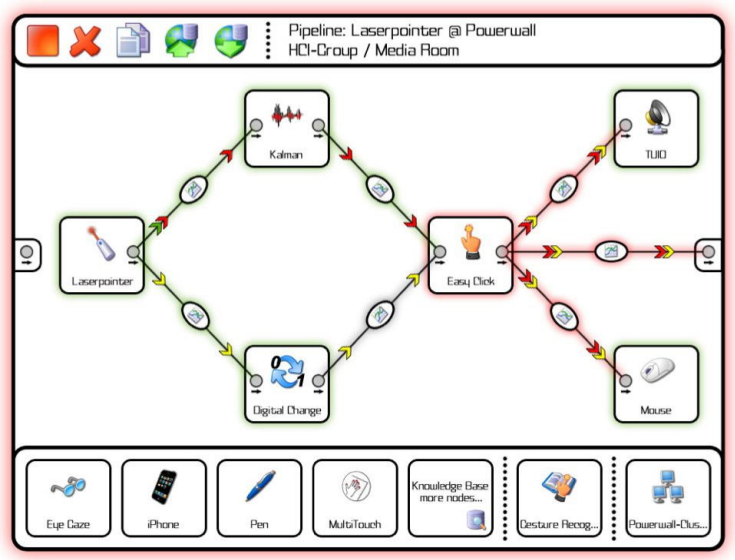

Figure 1: The Squidy stream-based graphical editor [17]

input events are speech utterance, recognised gestures or RFID tag identifiers with optionally attached semantic information. These low-frequency events are used to extract higher-level meaning from the temporal combination of different events. Graphical editors for event-driven frameworks provide a representation of the multimodal dialogue modelling, for example by using a state machine as in our tool presented in the previous section. In this case, the state machine represents application contexts as well as the transitions between these contexts and therefore outlines how the human-machine dialogue is expected to develop. Once again, graphical modelling and the underlying runtime implementation are deeply linked.

On the one hand, event-driven frameworks and their corresponding graphical editors follow a radically different approach to describe multimodal dialogues than stream-based frameworks. Indeed, event-driven frameworks are well suited for applications using semantically rich input modalities such as speech. Such frameworks frequently integrate multimodal fusion at the decision level, with high-level rules and combinators. However, event-driven approaches face problems in describing and managing high-frequency streams of information, such as accelerometer data. In general, it is necessary to delegate the interpretation of a low-level data stream to an external recogniser which delivers low-frequency high-level events with the effect of losing infor-

mation as part of the process. On the other hand, stream-based frameworks easily manage high-frequency streams of information and offer the possibility to redirect a given input data stream to a different function. These approaches are ideal candidates for *autonomic adaptation* based on available devices and services. This is for example the approach followed by the DynaMo framework [20], which is able to adapt the human-machine dialogue model at runtime based on certain conditions. However, as soon as low-frequency and high-level semantic events are to be processed by the system, stream-based approaches rely on ad-hoc implementations and workarounds to associate semantic-level events with their processing stream.

An alternative to the stream-based versus event-driven approaches is proposed by Mudra [18], which stores all input data in a central *fact base*. From this fact base, low-level stream data can be directly fed to a client application or it can be processed by a recognition algorithm, which might add new semantic-level interpretations to the fact base. The fusion of multimodal events can then be applied at any level and they can also be seamlessly mixed. However, even if this architecture effectively reconciles stream-based and event-driven approaches, it is challenging to create a graphical editor for modelling the human-machine dialogue and Mudra currently relies on a declarative rule-based language to describe the fusion processes.

2.5. Strengths and Limits of Our Approach

While each of the three approaches to create multimodal interfaces have their strengths and weaknesses, they have been rarely combined for achieving better performance. A notable exception is OpenInterface, a complete multimodal authoring framework which is linked to the Skemmi graphical editor. Furthermore, compatibility with UsiXML has been added on top of the framework. Besides OpenInterface, no other tool has investigated the linking of a framework with a modelling language and a graphical editor. IMBuilder and Squidy link the framework with a graphical editor, while Petshop integrates a graphical editor with the lower level modelling. Finally, OpenInterface is not a fully satisfactory approach since the link between the framework and UsiXML has not been defined from the ground up but added after both tools had already been defined.

In the following three sections, we present our triad of linked tools for the development of multimodal interfaces including the SMUIML modelling language, the HephaisTK toolkit as well as the corresponding graphical editor. The SMUIML modelling language forms part of the state-based family

of UIDLs. This family of UIDLs has proven effective when modelling user-machine dialogues at a high level, but tends to be lacking support for high frequency low level streams such as used when fusing finger input data from a multitouch surface. SMUIML makes no exception there and an approach such as the one of Mudra [18] better handles this kind of applications. In the state-based family of UIDLs, only TiMMiT has also been defined from the ground up as a multimodal dialogue modelling language. The work of Jacob [22] and Blanch et al. [21] addressed the modelling of interaction with graphical user interfaces, with limited support for specific modalities. However this work lacks the temporal and semantic rules necessary to fully address the modelling of multimodal interaction. According to Navarre et al. [2], TiMMiT and SMUIML are more comparable, however tool support could be further improved for both tools. SMUIML already had interpretation support through the HepaisTK toolkit, but lacked a graphical editor for the rapid authoring of SMUIML scripts. In this article we therefore present the graphical SMUIML editor as the last member of the triad of tools providing multimodal interaction practitioners with a complete set of tools for the creation of multimodal interfaces. While we provide a general overview of SMUIML and HephaisTK, a detailed description of these two components can be found in previously published work [25, 23, 27][3].

3. 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 to define the use of 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 [23]. As the first component of our triad of tools, the SMUIML language builds the foundation on top of which the other two tools have been implemented. SMUIML was designed to be as simple as possible and is targeting usability as described in [23]. Note that in order to minimise SMUIML's verbosity, we decided not to rely on existing standard multimodal interaction languages.

[3]The tools presented in this article are available as open source software from the following address: http://sourceforge.net/projects/hephaistk/

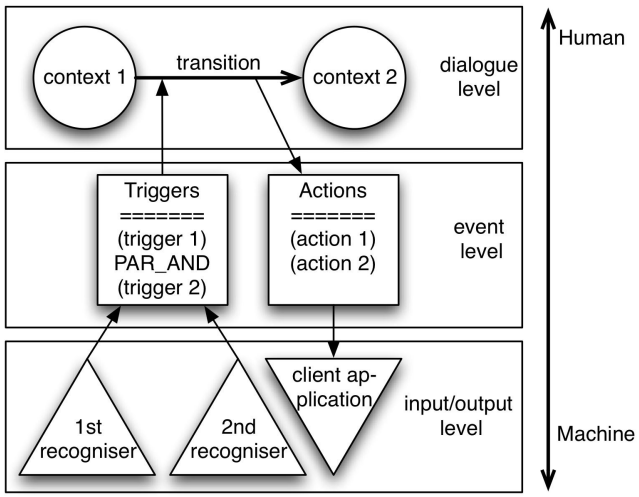

Figure 2: The three abstraction levels of SMUIML

The SMUIML language is divided into three levels of abstraction as shown in Figure 2. The lowest level details the different modalities which are going to be used in an application, as well as the particular recognisers to be used to access the different modalities. The middle level addresses input events (*triggers*) and output events (*actions*). Triggers are defined per modality which implies that they are not directly bound to specific recognisers and 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 of these contexts. The resulting human-machine 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 [14]. Triggers, actions and recognisers are grouped together in elements defined for this role as illustrated in Listing 1. Furthermore, *clauses* allow groups of reusable SMUIML instances to be defined. The combination of modalities is defined based on the CARE properties [24] as well as on the (non-)sequentiality of input triggers. The CARE proper-

ties model the temporal relationships between modalities and the temporal SMUIML descriptors are directly linked to them. As shown in Listing 1, the three levels of abstraction are directly reflected in the basic structure of the language.

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>
```

The spectrum of multimodal dialogue description language uses, on a scale from highly expressive to highly usable, was presented in [25]. Through various workshops, numerous informal discussions and a study of the current state of the art, we envisioned the three description language approaches shown in Figure 3. A highly formal language approach is suited for the configuration of a tool, the less formal language approach is good for communicating the details of an application and the intermediary approach focusses on the modelling. Along these three approaches, a formal language can also be used as a learning tool helping teachers in communicating the features of a particular application domain to their students.

Figure 3: Four purposes of a modelling language (machine to human oriented)

We have presented nine guidelines for a multimodal description language [25], which should be used as design tools or 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

4. Software Support for SMUIML

SMUIML enables the definition of a full model of multimodal human-machine events and dialogues by providing modelling capabilities. However, the language shows its true potential when linked to a range of different supportive software solutions. In the following, we introduce the software support within SMUIML for interpretation, in the form of the HephaisTK framework, and discuss the latest software addition in the form of a graphical editor for designing multimodal human-machine dialogues.

4.1. The HephaisTK Framework

The HephaisTK framework to develop multimodal interfaces based on the SMUIML scripting language is the second component of our triad of tools. As such, the framework uses the language described in the previous section as the underlying model on which all its interpretations are based.

In the HephaisTK architecture shown in Figure 4, the SMUIML document which is used to configure the behaviour of an application is indicated in the upper right corner. The description of the multimodal human-machine dialogue by means of a state machine is then used to drive the framework and enable intelligent multimodal input for an application written in the Java language. The application developer making use of HephaisTK has to focus only on the application logic and output representation. Note that in this section an application which is based on HephaisTK is going to be referred as the *client application*. As illustrated in Figure 4, the HephaisTK framework is based on a software agent framework called JADE [26]. Individual agents

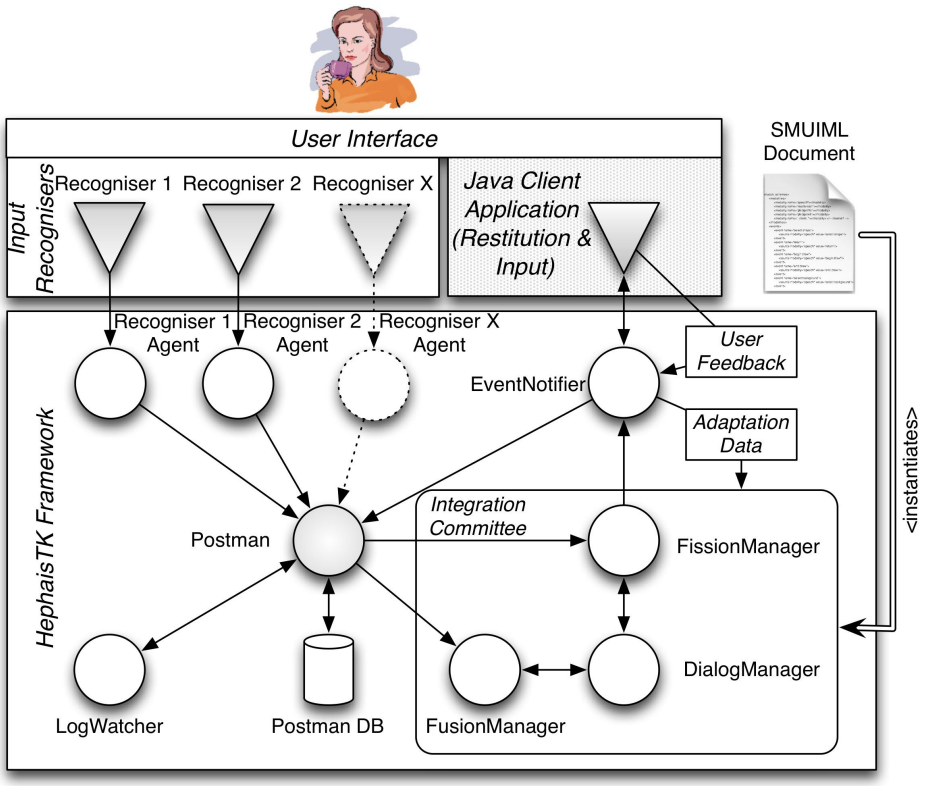

Figure 4: HephaisTK architecture

are responsible for a specific task. In Figure 4, each circle represents an agent or a set of agents assigned to a specific task. Typical examples are the agents responsible for managing input recognisers, such as speech or gesture recognisers. These agents have the task to launch, connect to and monitor recognisers for individual modalities. Their main goal is to gather input data from the recognisers and prepare this data for further processing. For example, a word uttered by a user is captured by the software agent responsible for the speech recogniser and delegated to the `Postman` agent.

The role of the `Postman` agent is to gather and store input data in a database or a memory buffer depending of the system capabilities. The `Postman` agent is also in charge of managing subscriptions to specific types of input data (e.g. speech data), where other agents can subscribe for specific types of input data. This technique is, for example, used to filter input data which is not going to be used by a particular client application. The `Postman`

agent is further responsible for managing requests from agents who are interested in recently captured multimodal input data. In most cases, agents forming part of the `Integration Committee` will be subscribed to a subset of the available input modalities. The `Integration Committee` consists of all agents who are responsible for the dialogue management, multimodal fusion as well as multimodal fission. The SMUIML scripts are directly used by the agents forming part of the `Integration Committee` to drive the behaviour of the entire framework. An example of such a script is shown in Figure 5. 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, linked with the `DialogManager` agent.

The HephaisTK `DialogManager` agent is responsible for interpreting the human-machine multimodal dialogue description in SMUIML. It also ensures that the framework and client application are in a consistent state. Our spoken word from the example mentioned before would be matched against the dialogue description provided by the SMUIML script and possibly trigger a new event. 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 be considered in a given context, the load on the recognisers is reduced and the overall recognition rate is improved. The `<dialog>` part of SMUIML further is used for the instantiation of the different fusion algorithms that are available in HephaisTK [27]. The SMUIML language is applied at multiple levels in the HephaisTK framework including the multimodal dialogue description level, the recognisers' dynamic launch and parametrisation level as well as the fusion engine instantiation level. SMUIML is typically used during the *system design* and *runtime* stages of the multimodal interface development. Please note that HephaisTK and SMUIML provide the interpretation of multimodal input. However, as explained above, the creation and management of an application's output is the responsibility of the developer.

4.2. Developing Applications with HephaisTK and SMUIML

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 [25], the editing of "raw" XML documents can easily lead to errors which are only detected at runtime when interpreting a SMUIML script. 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.

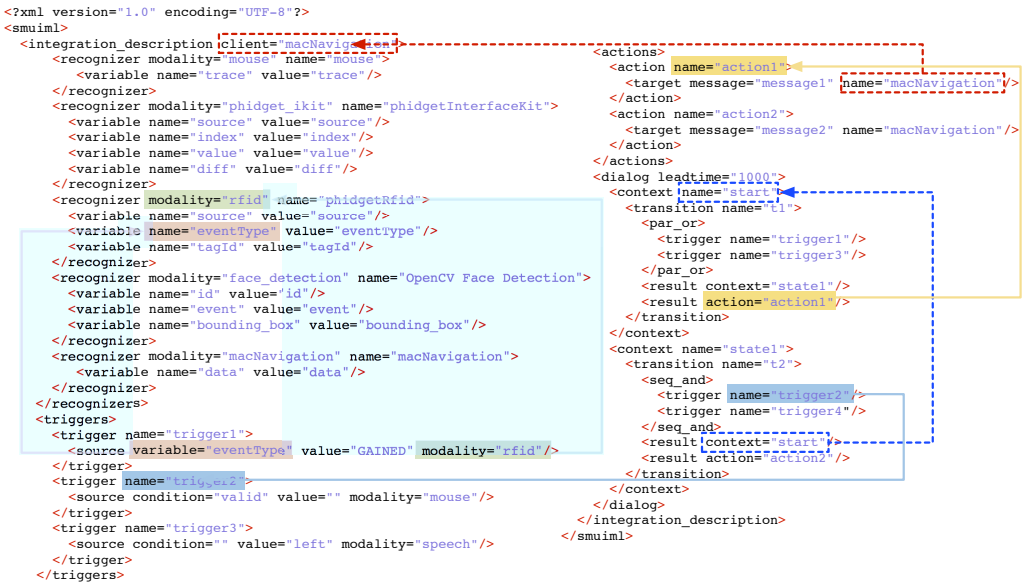

Figure 5: Snippets of SMUIML code showing relationships within the language

Figure 5 shows two SMUIML code snippets. On the left-hand side we see a snippet of code highlighting the description of a set of input recognisers as well as a number of triggers. The right-hand side shows two actions and a small dialogue element. In Figure 5, we can further see how the `<recognisers>`, `<triggers>`, `<actions>` and `<dialog>` SMUIML elements are deeply interlinked. In fact, this mix of elements over all three abstraction levels complicates the editing of SMUIML scripts since developers have to keep track of variables that have been defined at different levels. Further-

more, the editing of a variable name might invalidate a reference to that variable. Overall, the textual editing of long SMUIML scripts can become a tedious process. In order to overcome this issue, we have developed a graphical editor for the creation and manipulation of SMUIML scripts.

5. The Graphical SMUIML Editor

As the last component of our triad of tools for the creation of multimodal interfaces, we present the graphical SMUIML editor, which is linked to the SMUIML language and the HephaisTK framework. The goal of our graphical SMUIML editor was to provide developers who are not fully proficient with multimodal interfaces a usable and expressive tool to create SMUIML scripts. The graphical SMUIML 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 graphical SMUIML editor has been developed based on the Eclipse development platform providing a set of well-known interface elements. The graphical SMUIML tool itself was developed with help of the Graphical Editing Framework (GEF)[4] and the Eclipse Modeling Framework (EMF)[5].

The main window of the graphical editor is shown in Figure 6. The central part of the tool is dedicated to the dialogue representation. As stated earlier, SMUIML represents the multimodal human-machine dialogue via a state machine. A graphical representation of the state machine is used to visualise the multimodal dialogue. Note that the editor also provides access to a textual representation of the currently edited SMUIML script. Any changes that are applied either in the graphical or the textual representation are immediately reflected in the other representation. Furthermore, 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. As the user defines the

[4]http://www.eclipse.org/gef/
[5]http://www.eclipse.org/modeling/emf/

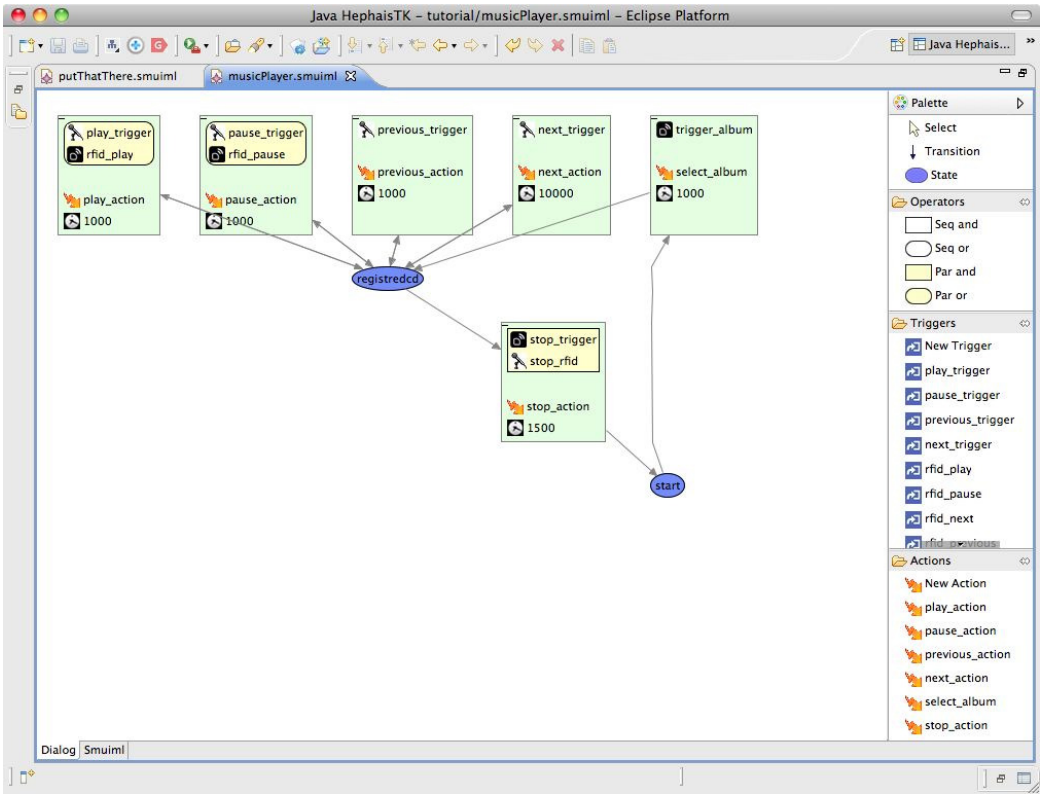

Figure 6: The graphical SMUIML editor showing a dialogue of a music player application

overall modelling of his application in the form of a state machine, he will use the state and transition tools to define this model. The selection tool which is used to select and edit elements in the model also is part of the `Palette` toolbox. The `Operators` toolbox offers a number of operators to combine different modalities as defined in the SMUIML specification. These operators are tightly linked to the CARE properties [24]. The `Seq and` operator 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 the triggers that have been defined for a given application, as well as a `New Trigger` button to create new triggers. Operators and triggers are typically added into transitions and describe the specific conditions which will trigger the transition. Note that temporal operators can be mixed in order to define complex temporal rules. Temporal

windows defined in transitions are generally also linked with a temporal operator. These temporal descriptors allow the interaction designer to describe the sequential or parallel processing of multimodal commands without having to specify tight delays between the different actions. Thus, user variabilities are taken into account, as illustrated by the example in the next paragraph. 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. Actions are used to define the messages that will be sent to the client application when a transition is successfully triggered. Note that triggers and actions are defined by the user. When creating a new trigger, the set of all available input modalities for the running instance of the HephaisTK framework is shown to the user, as well as information about the format of expected input. These elements allow the user to focus on the definition of the user-machine dialogue by freeing them from tedious tasks such as checking the data format of a recogniser for a given modality.

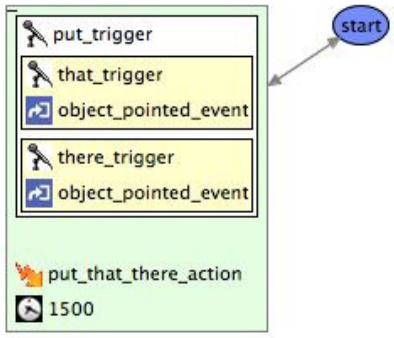

Figure 7: Graphical description of the "put that there" example

As an example, let us consider the graphical representation of Bolt's *"put that there"* [28] example shown in Figure 7. In the graphical editor, states (contexts) are 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 a single `start` context with a transition starting from and pointing to this context. This transition contains the overall description of the *"put that there"* action which 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 are required for this action. Furthermore, three temporal combination operators are used in this example. The main transition uses a `Seq and` operator requiring a *"put"* speech trigger to be followed by a *"that"* and *"there"* sub-event. The two sub-events use a `Par and` combination operator to define 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"* $point_1$ $point_2$ *"there"* or *"put"* $point_1$ *"that there"* $point_2$ 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 in the form of a message to be sent to the client application if the command has been successfully recognised. In our example, the transition proceeds to the original `start` context.

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 graphical SMUIML editor has been presented to two active researchers in multimodality, independent from the development of HephaisTK, in order to achieve a small expert review. This review resulted in a number of changes to improve the usability of the graphical editor. 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

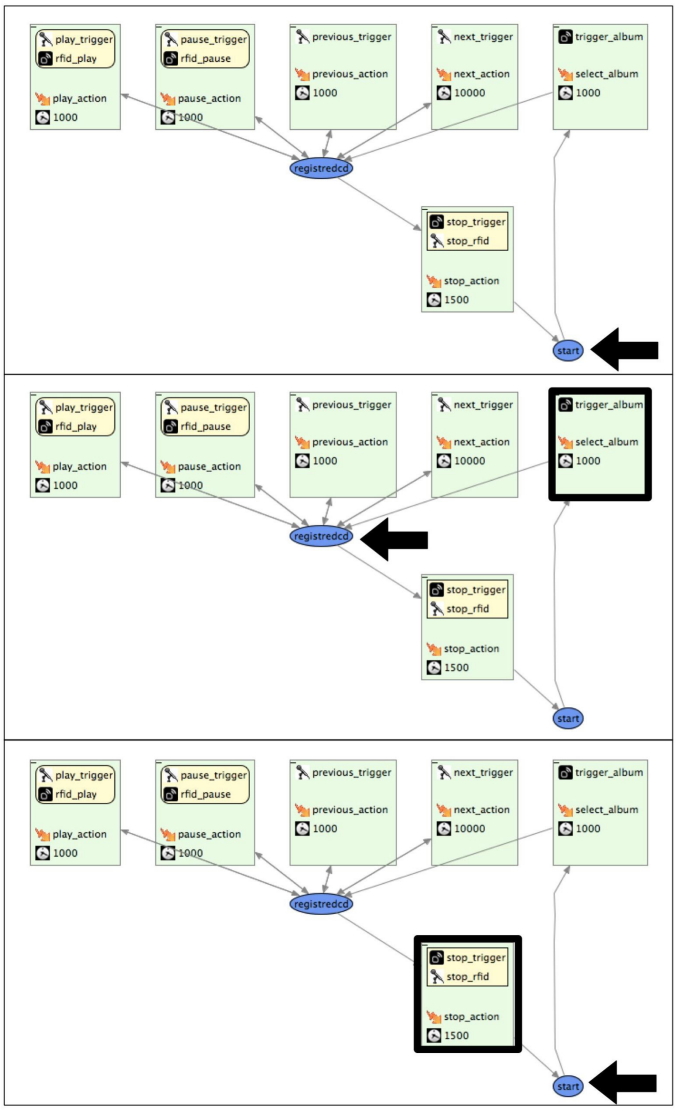

Figure 8: Graphical debugging tool showing three steps going from the `start` context to `registeredcd` and back to `start` again

slightly different colour to emphasise its special status compared to other context nodes. 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. This allows users to visually debug their SMUIML scripts while they are running in the framework. In the example illustrated in Figure 8, the application starts in the `start` context. A RFID reader 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 from the `registeredcd` context back to the `start` context and the visual debugger keeps track of the different events. As illustrated in this example, the graphical debugging tool allows developers to visually analyse the application behaviour in real time.

The graphical editor is used to edit SMUIML scripts as it has been described in this section. It is strongly linked to the HephaisTK framework since it uses a HephaisTK instance running on the same machine in order to identify the available input modalities. The debugging tool forming part of the graphical editor, automatically connects to HephaisTK in order to test a script in real time. All three tools—the graphical editor, the modelling language and the framework—form an integrated solution for the development of multimodal user interfaces, offering the user the right tool for a given task.

6. Future Work

While the presented graphical SMUIML editor looks promising and offers a number of features not available in other graphical multimodal interface editors, in the near future we plan to perform a detailed evaluation of the presented solution. 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.

An important future direction for our triad of tools is to support the flexible adaptation of multimodal interfaces [29]. The idea is to no longer

have a fixed combination of modalities, but to 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 could rely on SMUIML while the concrete context-dependant user interface specification would require the definition of a new language. The final user interface could still be realised based on HephaisTK. However, the graphical editor would also have to support the graphical description of adaptation rules, in order to keep consistency between our three tools.

This new language for flexible multimodal interface adaptation could then be used to provide innovative fluid document interfaces. Today's document formats often do not provide access to specific semantic subparts or embedded media types [30]. However, if we are able to get access to these document substructures, specific substructures or embedded media types can be associated with different modalities of interaction. Within the MobiCraNT project, we are currently investigating innovative next generation 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.

7. Conclusion

We have presented our exploration on software support for a multimodal UIDL based on the SMUIML multimodal dialogue modelling language. In addition to the SMUIML language, we presented two particular software components including the HephaisTK framework for interpretation of the SMUIML language at the runtime stage and the graphical SMUIML editor for designing multimodal interaction dialogues. The graphical SMUIML editor provides 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 them accessible via a user-friendly interface. Users further have the possibility to freely switch between the graphical and textual dialogue representation. The presented graphical SMUIML editor further addresses a number of usability

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 graphical SMUIML editor offers an integrated debugging tool supporting developers in analysing the behaviour of an application in real-time. With this last addition to our triad of tools, we offer an integrated solution for the development of multimodal interfaces, addressing the modelling, framework and visual editing level.

8. Acknowledgments

The authors would like to thank Saïd Mechkour for his work on the graphical SMUIML 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] S. Oviatt, Human-Centered Design Meets Cognitive Load Theory: Designing Interfaces That Help People Think, in: Proceedings of the 14th ACM International Conference on Multimedia (ACM MM 2006), Santa Barbara, USA, 2006, pp. 871–880.

[2] D. Navarre, P. Palanque, J.-F. Ladry, E. Barboni, ICOs: A Model-based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability, ACM Transactions on Computer-Human Interaction (TOCHI) 16(4), 2009, pp. 18:1–18:56.

[3] K. Katsurada, Y. Nakamura, H. Yamada, T. Nitta, XISL: A Language for Describing Multimodal Interaction Scenarios, in: Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI 2003), Vancouver, Canada, 2003, pp. 281–284.

[4] M. Araki, K. Tachibana, Multimodal Dialog Description Language for Rapid System Development, in: Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue, Sydney, Australia, 2006, pp. 109–116.

[5] J.-F. Ladry, P. Palanque, S. Basnyat, E. Barboni, D. Navarre, Dealing with Reliability and Evolvability in Description Techniques for Next Generation User Interfaces, in: Proceedings of the 26th ACM International Conference on Human Factors in Computer Systems (CHI 2008), Florence, Italy, 2008.

[6] A. Stanciulescu, Q. Limbourg, J. Vanderdonckt, B. Michotte, F. Montero, A Transformational Approach for Multimodal Web User Interfaces Based on UsiXML, in: Proceedings of the 7th International Conference on Multimodal Interfaces (ICMI 2005), Torento, Italy, 2005.

[7] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. Lopez-Jaquero, USIXML: A Language Supporting Multi-Path Development of User Interfaces, in: Engineering Human Computer Interaction and Interactive Systems (2005): pp. 134–135.

[8] F. Paternò, C. Santoro, J. Mäntyjärvi, G. Mori, S. Sansone, Authoring Pervasive Multimodal User Interfaces, International Journal of Web Engineering and Technology 4 (2) (2008) 235–261.

[9] R. Schaefer, S. Bleul, W. Mueller, Dialog Modeling for Multiple Devices and Multiple Interaction Modalities, in: Proceedings of the 5th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2006), Hasselt, Belgium, 2006, pp. 39–53.

[10] J. D. Boeck, D. Vanacken, C. Raymaekers, K. Coninx, High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT, Journal of Virtual Reality and Broadcasting 4(2).

[11] J.-S. Sottet, G. Calvary, J. Coutaz, J.-M. Favre, J. Vanderdonckt, A. Stanciulescu, S. Lepreux, A Language Perspective on the Development of Plastic Multimodal User Interfaces, Journal on Multimodal User Interfaces 1 (2007) 1–12.

[12] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, J. Clow, QuickSet: Multimodal Interaction for Simulation Set-Up and Control, in: Proceedings of the 5th Conference on Applied Natural Language Processing (ANLC 1997), Washington DC, USA, 1997.

[13] A. Cheyer, D. Martin, The Open Agent Architecture, Autonomous Agents and Multi-Agent Systems 4 (2001) 143–148.

[14] M.-L. Bourguet, A Toolkit for Creating and Testing Multimodal Interface Designs, in: Adjunct Proceedings of the 15th Annual Symposium on User Interface Software and Technology (UIST 2002), Paris, France, 2002.

[15] F. Flippo, A. Krebs, I. Marsic, A Framework for Rapid Development of Multimodal Interfaces, in: Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI 2003), Vancouver, Canada, 2003.

[16] M. Serrano, L. Nigay, J. Lawson, A. Ramsay, R. Murray-Smith, S. Denef, The OpenInterface Framework: A Tool for Multimodal Interaction, in: Proceedings of the 26th International Conference on Human Factors in Computing Systems (CHI 2008), Florence, Italy, 2008, pp. 3501–3506.

[17] W. A. König, R. Rädle, H. Reiterer, Squidy: A Zoomable Design Environment for Natural User Interfaces, in: Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI 2009), Boston, USA, 2009.

[18] L. Hoste, B. Dumas, B. Signer, Mudra: A Unified Multimodal Interaction Framework, in: Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI 2011), Alicante, Spain, 2011, pp. 97–104.

[19] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, B. Macq, An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-the-Shelf Heterogeneous Components, in: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009), Pittsburgh, USA, 2009, pp. 245–254.

[20] P.-A. Avouac, P. Lalanda, L. Nigay, Service-Oriented Autonomic Multimodal Interaction in a Pervasive Environment, in: Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI 2011), Alicante, Spain, 2011, pp. 369–376.

[21] R. Blanch, M. Beaudouin-Lafon, Programming Rich Interactions Using the Hierarchical State Machine Toolkit, in: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI 2006), Venezia, Italy, 2006, pp. 51–58.

[22] R. Jacob, A Specification Language for Direct-Manipulation User Interfaces, in: ACM Transactions on Graphics (TOG) 5(4) (1986).

[23] B. Dumas, Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems, Ph.D. thesis, University of Fribourg, dissertation No. 1695 (2010).

[24] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, R. M. Young, Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties, in: Proceedings of the 5th International Conference on Human-Computer Interaction (Interact 1995), Lillehammer, Norway, 1995.

[25] B. Dumas, D. Lalanne, 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) (2010) 237–247.

[26] F. Bellifemine, A. Poggi, G. Rimassa, JADE – A FIPA-compliant Agent Framework, in: Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM 1999), London, UK, 1999, pp. 97–108.

[27] B. Dumas, B. Signer, D. Lalanne, Fusion in Multimodal Interactive Systems: an HMM-Based Algorithm for User-Induced Adaptation, in: Proceedings of the 4th Symposium on Engineering Interactive Computing Systems (EICS 2012), Copenhagen, Denmark, 2012, pp. 15–24.

[28] R. A. Bolt, "Put-that-there": Voice and Gesture at the Graphics Interface, in: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1980), Seattle, USA, 1980, pp. 262–270.

[29] J. Vanderdonckt, G. Calvary, J. Coutaz, A. Stanciulescu, Multimodality for Plastic User Interfaces: Models, Methods, and Principles, in: Multimodal User Interfaces, Signals and Communication Technology, Springer Berlin Heidelberg, 2008, pp. 61–84.

[30] B. Signer, What is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse, in: Proceedings of the 29th International Conference on Conceptual Modeling (ER 2010), Vancouver, Canada, 2010, pp. 391–404.