Published in "Journal on multimodal user interfaces 3(3): 237-247, 2010"
which should be cited to refer to this work.

Description Languages for Multimodal Interaction: A Set of Guidelines and its Illustration with SMUIML

Bruno Dumas · Denis Lalanne · Rolf Ingold

Received: date / Accepted: date

Abstract This article introduces the problem of modeling multimodal interaction, in the form of markup languages. After an analysis of the current state of the art in multimodal interaction description languages, nine guidelines for languages dedicated at multimodal interaction description are introduced, as well as four different roles that such language should target: communication, configuration, teaching and modeling. The article further presents the SMUIML language, our proposed solution to improve the time synchronicity aspect while still fulfilling other guidelines. SMUIML is finally mapped to these guidelines as a way to evaluate their spectrum and to sketch future works.

Keywords Markup languages · Multimodal interfaces · Multimodal modelling · Human-machine interaction

1 Introduction

Multimodal interfaces have drawn much interest since the last decades. The promise of a more natural interaction by using and combining different modalities such as speech, gestures, emotions or gaze direction, along with the perspective of giving better expressive power to humans when interacting with computers and electronic devices, helped the creation of multiple multimodal systems and applications. This decade saw also the development and enhancement of a number of natural communication means recognizers, such as speech recognizers or posture recognizers. Meanwhile, studies on the use of multimodal systems, as well as new algorithms for the fusion and fission of modalities were achieved.

But multimodal interfaces still remain difficult to create. On one hand, the use of state-of-the-art technologies such as speech or gesture recognition implies advanced knowledge in those state-of-the-art technologies from people wishing to develop multimodal interfaces; on the other hand, research fields such as modalities fusion, fission and synchronization remain only partly explored. To resolve this, tools and frameworks targeted at easing the creation of multimodal interfaces have been developed, and in the same process opened another broad question: how to best represent and model multimodal human-machine interaction? This article explores one of the possible ways to address this problem: description languages, and some of their characteristics. In particular, this article tries to answer two questions: what would be the uses of description languages for multimodal interaction? And how should such languages be able to describe best multimodal interaction and its distinctive features? Answering this last question leads us to introduce a set of nine guidelines, covering different user- and system-centered aspects that should be handled by such description languages. The article finally presents SMUIML (*Synchronized Multimodal User Interaction Modeling Language*), a description language for multimodal human-machine interaction, which served as a testbed for these guidelines.

Section 2 of this article presents multimodal interaction description languages and multimodal interfaces toolkits related to this study. The third section presents different approaches to the problem of formal description and modeling of multimodal interfaces. Guidelines

B. Dumas · D. Lalanne · R. Ingold
Computer Science Department, University of Fribourg
Boulevard de Pérolles 90, 1700 Fribourg
Switzerland

E-mail: {firstname.lastname}@unifr.ch

Table 1 State of the art languages and their features.

	Layers	Events	Time	Plasticity	Web-oriented	Error handling	Data Modeling
EMMA							X
XISL		X	X		X		
ICO		X	X			X	X
UsiXML	X	X		X	X		
TeresaXML	X	X					
MIML	X				X		
NiMMiT	X	X	X				

for languages targeted at describing multimodal interaction are described in the fourth section. The fifth section introduces the SMUIML language, a language for description and modeling of multimodal human-machine interaction derived from those guidelines. Evaluation and positioning of the SMUIML languages are presented in the sixth section. The seventh and final section concludes this article.

2 Description Languages for Multimodal Interaction

Interesting attempts at creating a full-fledged language for description of user-machine multimodal interaction have come up in the past few years. A number of the approaches presented below revolve around the concept of a "multimodal web", enforced by the World Wide Web Consortium (W3C) Multimodal Interaction Activity and its proposed multimodal architecture. This theoretical framework describes major components involved in multimodal interaction, as well as potential or existent markup languages 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, or modality-focused languages such as VoiceXML or InkML. The works of the W3C inspired Katsurada et al. [15] for their work on the XISL XML language. XISL focuses on synchronization of multimodal input and output, as well as dialog flow and transition. Another approach of the problem is the one of Araki et al. [1], who propose MIML (Multimodal Interaction Markup Language). One of the key characteristics of this language is its three-layered description of interaction, focusing on interaction, tasks and platform. Ladry et al. [16] proposed the ICO notation for the description of multimodal interaction. This approach is closely related to a visual tool allowing edition and simulation of interactive systems, while being able to monitor at a low level a systems operation. Stanciulescu et al. [22] followed a transformational approach for developing multimodal web user interfaces based on UsiXML, also in the steps of the W3C. Four steps are achieved to go from a generic model to the final user interface. Thus, one of the main features of their work is a strong independence to the actual input and output available channels. This transformational approach is also used in Teresa XML (see Paterno et al. [18]). Finally, at a higher level of modeling, NiMMiT (see De Boeck et al. [7]) is a graphical notation associated to a language used for expressing and evaluating multimodal user interaction.

All these approaches seem, at first glance, rather different one from the other. Some features are however common between most of them. Table 1 lists a number of features that could be extracted between the different approaches, as well as some more specific features, related to current research in multimodal interaction. EMMA is included in the table, even if this language targets primarily data transfer between entities of a given multimodal system; in this regard, EMMA perfectly addresses input and output data source representation; in fact, this is the only language to fully address this. XISL is a language targeted at web interaction, and offering a SMIL-like language for multimodal interaction; thus, it provides control over time synchronicity (e.g. with parallel or sequential playing), at least on the output side. The ICO notation targets safety-critical applications, with simulation capabilities; it has events description capabilities but lacks layers of abstraction. On the contrary, UsiXML and TeresaXML were based on specific layers of abstraction, namely AUI (abstract user interface), CUI (concrete user interface) and FUI (final user interface), with XSLT transformations managing crossing from one layer to another. UsiXML also takes into account plasticity. Furthermore, TeresaXML offered extensive events management capabilities. MIML, targeted at multimodal web interaction, also offers layers of abstraction: it is composed of three different languages, managing user/machine interaction, events description, and input/output representation, respectively. Finally, NiMMiT also takes into account separate layers and events management capabilities.

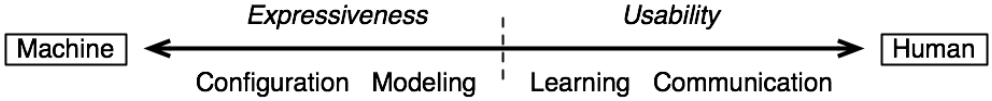

Fig. 1 Different purposes for a multimodal user interaction description language.

A detailed analysis of Table 1 reveals that description languages for multimodal user-machine interaction either are complete on a input/output data stream management point of view, but lack abstraction and expression power, or focus on higher levels of user-machine interaction description, thus losing control over data or even events. For example, ICO, which addresses multimodal interaction at a low-level, is thus forced to have precise data modeling and error handling; on the contrary, a language such as UsiXML offers higher-level abstraction, with fewer control over the lower layers. It is also to be noted that particular features are incompletely addressed by state of the art languages. Control over time synchronicity has been addressed on the output side of multimodal interaction, but those languages still lack capability to manage complementary, assigned, redundant or equivalent (see CARE properties in Coutaz et al. [6]) input events. As for error handling, most languages lack ways to express graceful recovery from recognition errors, data fusion mistakes, or event system errors. Most languages also lack context and user model modeling, and furthermore plasticity (user and context adaptability) control, with the exception of UsiXML.

Most of the languages described above focus on the "multimodal web", and thus assume interpretation by browser plugins, or even future versions of the browsers themselves. Description languages for multimodal interaction can however be used to be interpreted by dedicated tools for the creation of standalone multimodal applications. On the subject of tools allowing creation of multimodal interfaces, the use of specific languages for configuration has been frequent, but sparsely studied. Cohen et al. [5] worked on Quickset, a speech/pen multimodal interface, based on Open Agent Architecture, which served as a test bed for different fusion methods. Bourguet [3] endeavored in the creation of a multimodal toolkit in which multimodal scenarios could be modeled using finite state machines. This multimodal toolkit is composed of two components, a graphical user interface named IMBuilder, which interfaces the multimodal framework itself, named MEngine. Multimodal interaction models created with IMBuilder are saved as a XML file. Flippo et al. [13] also worked on the design of a multimodal framework, geared toward direct integration into a multimodal application. The general framework architecture is based on agents, while the fusion technique itself uses frames. Configuration of the fusion is done via a XML file, specifying for each frame a number of slots to be filled and direct link to actual resolvers implementations. Lastly, Bouchet et al. [2] proposed a component-based approach called ICARE thoroughly based on the CARE (see Coutaz et al. [6]) design space. These components cover elementary tasks, modality-dependent tasks or generic tasks like fusion. The components-based approach of ICARE has been used to create a comprehensive open-source toolkit called OpenInterface [19]. Components are configured via CIDL XML files.

3 Description Languages Spectrum

An interesting question to ponder is: who is the user of description languages? The answer is not as obvious as it could appear. People who would want to design multimodal interaction with such languages could be either engineers creating multimodal systems based on programming tools, designers using higher-level tools for creating multimodal interfaces without having to delve too much into code, or even advanced users wishing to customize interaction. These different users will not have the same approach to multimodal interaction description languages. 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, perfectly fit for configuring a tool, a loosely formal language approach, good at communicating the details of an application, and a "middle" approach, focused on modeling. Along those three purposes, a fourth purpose for formal languages can be added: learning tool. In summary, formal languages can be used for the following purposes:

- configuration
- communication
- modeling
- learning

Hence, formal languages can help configure a multimodal system, thus working as scripting or programming languages; they can be used as communication tools to help exchange and structure ideas about a multimodal systems; formal languages with a thoroughly

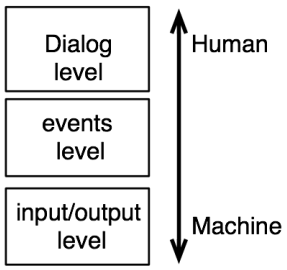

Fig. 2 Different levels for a multimodal user interaction description language.

thought structure can encourage careful modeling of a multimodal application; finally, as potential communication and modeling tools, they can be interesting learning means to tackle multimodal interaction. Thus, every description language is expected to find a balanced place on an axis running from highly formal to loosely formal, from configuration to communication (see Figure 1).

Formal languages for description of multimodal interaction can be approached from two different directions: either from expressiveness, or from usability (see De Boeck et al. [7]). Expressiveness covers technical features such as extensibility, completeness, reusability, or temporal aspects consideration; usability covers more human features such as programmability or readability. Any formal language will have to find its place between those two general requirements. Some languages will for example tend more toward expressiveness, letting all edition be done through a dedicated GUI (visual programming tool); others will focus on usability. In case of a visual programming tool, the usability of the description language is not that important, since it is hidden to users and to be read by the toolkit using the description. An interesting approach is to seek balance between usability and expressiveness: that is, a language able to configure a multimodal system, with high level modeling, and readable enough to be used as a learning tool, or even a communication tool. To attain those objectives, such a language will need to have a number of clearly separated and specifically dedicated parts, closely tied together (see Windgrave [23]). We introduce three different levels that will be used in the rest of this paper: a human-machine dialog-level, an input/output level, and a middle events-level in order to create a link between the human-centered part and the machine-centered part (see Figure 2).

4 Guidelines For Languages For Multimodal Toolkits

Sire and Chatty [21] describe what one should want from a multimodal user interfaces programming language. The requirements they express in their article include features such as modality agnosticity, extensible event definition mechanisms or reusable components. From their proposal, the analysis of the state of the art languages referenced in Section 2 and Table 1, and open changes described in key bibliographic references [17] [11], the following guidelines for a multimodal description language have been derived. These guidelines are to be seen as a "checklist" of potential features a given multimodal interaction description language can provide. By no means should every language follow all of them. Guidelines should be used as design tools, or as language analysis criterias.

G1 *Abstraction levels*: different abstraction levels are advised, as multimodal interaction description can be huge: for example, a description language should separate description of the events and description of the human-machine dialog. Also, reusable parts or structures can greatly help programmability.

G2 *Modeling the human-machine dialog*: there should be some way to model the human-machine dialog, be it with a state machine, with an imperative approach with control structures, a declarative approach, or another approach.

G3 *Adaptability to context and user (input and output)*: as multimodal interfaces often offer redundancy between modalities, adaptability to context and user (also called plasticity) should be taken into account by a language dedicated at describing multimodal interaction. It is worth noting that adaptability can be considered from an input and an output point of view. On the input side, adaptability would focus on using user information and context to help recognition and fusion processes; on the output side, message selection, modalities and output coordination would be achieved according to user and context.

G4 *Control over fusion mechanism*: algorithms used to fuse multimodal input data can be quite complex and deliver different results according to the algorithm or its settings. Thus, description languages should take into account fusion parameters and ways to control them, for example by allowing choice between different algorithms, or by allowing management of fusion parameters.

G5 *Control over time synchronicity*: actual human-machine dialog description should give control over time synchronicity: when multiple events can all lead to a given action, how should the system fuse data if

Table 2 Eight guidelines for four purposes.

	G1	G2	G3	G4	G5	G6	G7	G8
Communication	X	X	X	X	X			
Learning	X	X	X	X	X			
Modeling		X	X	X	X	X	X	
Configuration		X	X	X	X	X	X	X

those events are activated at the same time? Thus, the fusion process would greatly benefit from control over time synchronicity, for example by taking into account the CARE properties [6].

G6 *Error handling*: error handling should be taken into account early on. Multimodal systems feature a large number of potential error sources, from the recognizers to the integration to the answer selection. Hence, a language for description of multimodal interaction should provide some way to handle errors and recognition mistakes, for example by allowing default choices to be specified, or encouraging the design of guided dialogues.

G7 *Events management*: a mechanism for events description and management should be taken into consideration, as events seem a natural way for people to think about how their multimodal application should work (see section 6 for more details).

G8 *Input and output sources representation*: some way to represent the actual input and output sources can also be interesting, as the creator of the multimodal user interface wants to have control over which recognizer is used, and possibly be able to tune some parameters.

G* *Find the right balance between usability and expressiveness*: this follows from the discussion in Section 2, and is maybe the single most important guideline: any description language should find its place (and role) between usability and expressiveness, between the human and the machine. This guideline is at a higher level and thus is not integrated in the following tables.

Table 2 presents the eight guidelines above and matches them with the four different purposes of a description language identified in section 3. Guideline were ranked from the most user-focused (abstraction levels) to the most system-focused (input/output data representation), hence the diagonal shape adopted by the crosses in the table. The next section presents SMUIML, a language for description of multimodal interaction, which will help illustrate most of the presented guidelines.

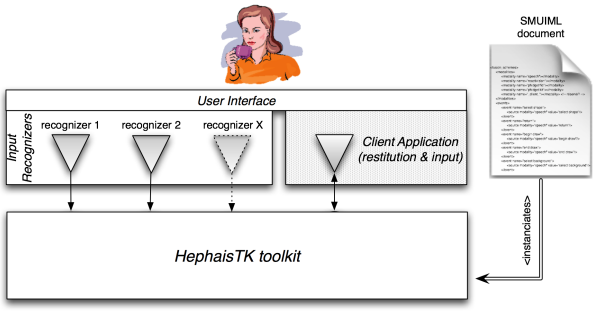

Fig. 3 The link between the HephaisTK toolkit and the SMUIML document.

5 SMUIML

SMUIML stands for Synchronized Multimodal User Interaction Modeling Language. As its name implies, the language seeks to offer developers a language for describing multimodal interaction, expressing in an easy-to-read and expressive way the modalities used, the recognizers attached to a given modality, the user-machine dialog modeling, the various events associated to this dialog, and the way those different events can be temporally synchronized.

Description languages can be used as configuration scripts for tools allowing creation of multimodal interfaces. The SMUIML language has been primarily designed with this goal in mind. Thus, the SMUIML language is able to configure a toolkit for creation of multimodal interfaces (Figure 3). This toolkit, named HephaisTK, is outlined in the subsection 5.6. It is however to be noted that the subject of this article lies in description languages and guidelines for the creation of such languages. A deeper discussion on multimodal toolkits architectures and the HephaisTK toolkit can be found in Dumas et al. [9]. Besides the goal of configuration scripting, SMUIML follows the goal of multimodal interaction modeling as it tries to guide the user by giving them a pattern for the creation of their applications; SMUIML is also used as a tool in a course on multimodal interaction, so, obviously, has also the goal of a learning tool.

Developers can find a complete XML Schema reference description of SMUIML available in [8].

The SMUIML language itself is described in the following subsections. The subsection 5.1 gives an overview of the language structure, followed by subsections 5.2 to 5.5, which give a detailed view of the three different levels described by the language. Finally, subsection 5.6 shows how SMUIML is used in the context of the HephaisTK toolkit.

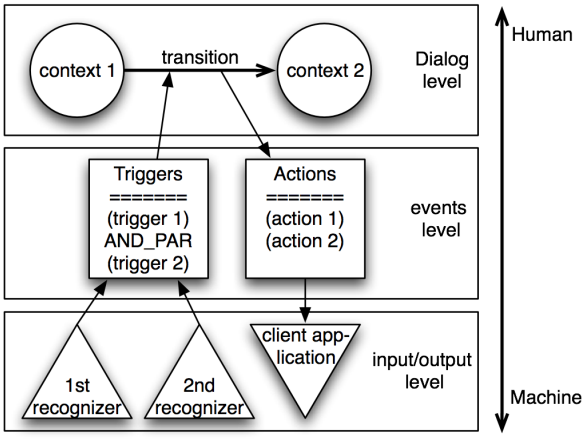

Fig. 4 The three levels of SMUIML.

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
 <integration_description client="client_app">
    <recognizers>
    <!-- ... -->
    </recognizers>
    <triggers>
    <!-- ... -->
    </triggers>
    <actions>
    <!-- ... -->
    </actions>
    <dialog>
    <!-- ... -->
    </dialog>
  </integration_description>
<smuiml>
```

Fig. 5 General SMUIML layout

5.1 SMUIML structure

The way a SMUIML file is split allows a clear separation between three levels necessary to the integration process. As shows Figure 4, <recognizers> are at the lower, input/output level, <triggers> and <actions> form a middle level, devoted to events management, and the upper level contains the <dialog> description. This abstraction in three different levels allows components definition and reusability. In order to enhance reusability, the upper dialog level allows definition of clauses that can be later used and extended.

A typical SMUIML document is divided in a number of sections (see Figure 5). First, the overall integration description is proper to a given client application. Whenever a new client application asks the HephaisTK toolkit for a handle, it will have to give some identifier name (in the case of Figure 5, "client_app"). This identifier is used in the SMUIML file to identify which integration description is to be used for the current application. Hence, multiple applications can be described in a same script, for example in a case where multiple small applications could access concurrently a number of available input modalities.

For a given client application, four main sections form the description of the multimodal interaction scenario. The first part, <recognizers>, indicates which particular recognizer will be tied to which modality. It also allows definition, per recognizer, of a number of variables that will be of use to the client application. The second section, <triggers>, lists the different events that will be of interest for the client application. The focus of this section is to model as generic triggers all events coming from the different recognizers. <actions> form the third section, and have the same goal of giving a generic model to the output side of the toolkit. In particular, those <actions> will serve as a description of the toolkit-to-client application messages, as well as indicate the potential variables that would have to be transferred to the client application. <triggers> and <actions> form a set of building blocks, using the results of the various recognizers defined in the first section, and further called in the definition of the <dialog>. This dialog is described by means of a finite state machine. States are described by means of <context> elements, and to a given <context> are attached a number of <transition> elements.

A complete SMUIML example is given in Figure 6. This example is taken from a multimodal drawing application realized with help of the Reactivision [14] computer vision framework and Sphinx 4 speech recognition system. The example features two recognizers, three input triggers, one output action and part of the overall human-machine interaction dialog. All these elements will be detailed in the coming subsections.

5.2 SMUIML recognizers

At the recognizers level, the goal is to tie the multimodal dialog scenario with the actual recognizers that the developer wishes to use for his application. In the context of the HephaisTK toolkit, all recognizers are identified by a general name throughout the toolkit. This general identifier is hence used in SMUIML. The HephaisTK toolkit keeps a list of recognizers, and their associated modality (or modalities). For example, if a number of speech recognizers are available, using one of them or the other is a matter of changing the "name" attribute of the speech-related ¡recognizer¿ element. As every events are defined afterwards in relation to the

modality, and not a particular recognizer, this allows for more flexibility in the creation of the implementation, Moreover, design-wise, it enhances readability and allows developers to design their application before thinking about which recognizer they intend to use. It is to be noted that the <recognizers> part is the only one where a tight link between the SMUIML language and a given tool (in this case, the HephaisTK toolkit) appears. The modality the different recognizers provide is also indicated. The "modality" attribute is a string of characters, chosen from a list of keywords. This keywords list is created by the HephaisTK toolkit (see subsection 5.6) at runtime against the available recognizers. In other words, the toolkit knows in real time which modality it is able to offer.

Finally, SMUIML offers the possibility to declare variables attached to a given recognizer.Data types vary from modality to modality, but are expected to be consistent within one modality; for example, every speech recognizers are expected to deliver text outputs.

5.3 SMUIML triggers

Triggers are at the core of the transition mechanism of SMUIML. They describe a sub-set of interest from all the possible events coming from the different recognizers. A set of input events can hence be abstracted behind one trigger name, enhancing as much the script readability.

A standard trigger declaration is shown in Figure 6. In this example, two triggers are defined for the speech modality, regardless of the recognizer actually used. For example, the speech recognizer is simply assumed to send its results as strings of characters, and be scripted by means of a BNF-style grammar. A unique name or number identifies each <trigger>. As for the last trigger, attached to a Phidget (see Greenberg et al. [12]) RFID tags reader in this example, the three values declared are linked to actual tags numbers in the <variable> declarations in the recognizers; the tag numbers could also be used in place of variables.

5.4 SMUIML actions

<actions> are the output equivalent of <triggers>. They describe the messages and their content that will form the communication channel between HephaisTK toolkit and its client application. A typical <action> declaration is shown on Figure 6.

The goal of those messages is to let the client application know in which state the toolkit finds itself in, by

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description
    client="xpaint_client">

    <!-- declaration of recognizers -->
    <recognizers>
      <recognizer
          name="sphinx4" modality="speech"/>
      <recognizer name="reactivision"
          modality="reactivision">
        <variable name="posx"
            value="x" type="int"/>
        <variable name="posy"
            value="y" type="int"/>
      </recognizer>
    </recognizers>

    <!-- declaration of input triggers -->
    <triggers>
      <trigger name="return">
        <source modality="speech"
            value="return"/>
      </trigger>
      <trigger name="operation">
        <source modality="speech"
            value="rotate shape|move shape"/>
      </trigger>
      <trigger name="tools_one_hand">
        <source modality="rfid"
            value="select|line|freehand"/>
      </trigger>
    </triggers>

    <!-- declaration of output actions -->
    <actions>
      <action name="draw_action">
        <target name="xpaint_client" message=
            "draw $oper $shape $posx $posy"/>
      </action>
    </actions>

    <!-- declaration of H/M dialog -->
    <dialog leadtime="1400">
      <context name="modification">
        <transition name="modif_clause">
          <par_and>
            <trigger name="operation"/>
            <trigger name="selected shape"/>
            <trigger name="position"/>
          </par_and>
          <result action="draw_action"/>
        </transition>
        <transition>
          <trigger name="return"/>
          <result context="start"/>
        </transition>
      </context>
    </dialog>

  </integration_description>
</smuiml>
```

Fig. 6 A typical example of a SMUIML script.

means of a clearly defined set of messages. Those messages can contain variables previously defined, such as the $posx variable defined in Figure 6. The client application will then be able to know what content is to be restituted to the user. The choice was made not to offer extensive control over fission of modalities in SMUIML, for two reasons. First, having extensive control over the way content is restituted can go as far as recreating a full rendering language, which was not the goal of this work. Second, mixing as little as possible the model and the view allows for better readability of the language. Hence, restitution of the content is up to the client application. This choice can however be a source of errors, as heavily modal applications would need some way to ensure synchronicity between the toolkit and the application.

5.5 SMUIML dialog

The <dialog> element describes the integration mechanisms of a SMUIML script. In essence, a <dialog> is a finite state machine, with transitions defined by the <triggers> and <actions> events that were presented in the former sections. We are currently working on adding other fusion algorithms into the HephaisTK platform, some of which will need us go beyond the simple state machine paradigm for the description of the human-machine interaction. States of the dialog are described by <context> elements. Each context has a unique name identifying it. One context must have a "start_context" attribute, defining it as the starting state. An "end_context" attribute also exists to describe a final state. A simple example of a <dialog> is represented in Figure 6.

When multiple triggers are present, the developer should be able to clarify time synchronicity issues, i.e. how the incoming multimodal triggers should be considered. The approach of SMUIML is to distinguish between parallel and sequential triggers, and between coupled and individual triggers. To denote those different cases, a set of keywords has been selected. Keyword "par" is for parallel triggers, "seq" for sequential triggers, "and" for coupled triggers, "or" for individual triggers.

Four elements to describe the different behaviors have been designed by mixing those four CARE properties. <par_and> is to be used when multiple properties are to be fused together, as they all are necessary for the meaning extraction process. <seq_and> describes one or multiple individual triggers all necessary in sequence to trigger a transition. <par_or> describes redundant multimodal triggers having similar meanings. Each one is sufficient for the correct meaning to be extracted, but

```
<transition>
  <par_and>
    <!-- Complementarity -->
  </par_and>
  <seq_and>
    <!-- sequenced complementarity -->
  </seq_and>
  <par_or>
    <!-- Redundancy -->
  </par_or>
  <seq_or>
    <!-- Equivalence -->
  </seq_or>
</transition>
```

Fig. 7 Triggers combination elements in SMUIML

they all can be expressed at the same time by the user, increasing as such the robustness and recognition rate (for example, a user issuing a "play" vocal command and simultaneously pushing a play button). Finally, the <seq_or> element is to be used when multiple triggers can lead to the same result, but only one of them is to be provided. Those four integration describer elements can also be combined in order to express all kinds of multimodal interactions. In fact, as shows comments in Figure 7, three of those four elements correspond to three of the four CARE properties of multimodal interactive systems (Complementarity, Redundancy and Equivalence) as defined in Coutaz et al. [6]. The only integration describer element not matched to a CARE property is <seq_and>. The choice to drop the Assignment CARE property was made because we felt it was more meaningful to be able to differentiate sequenced and un-sequenced complementarity, than to differentiate equivalence of choice (one of a number of modalities is required, does not matter if two are selected at the same time) and assignment (one and only one of a number of modalities is required); In fact, assignment can be expressed by using one transition per modality. The CARE properties have revealed themselves a handy tool used by a number of toolkits to help formalize relationships between different modalities; choice was made to include them into SMUIML in order to answer the guideline of time synchronicity control.

5.6 SMUIML language interpretation

In order to provide a tool allowing developers to prototype multimodal interfaces in an easier way than by building them from scratch, a toolkit named HephaisTK has been developed. This toolkit has been designed to plug itself in a client application that wishes to receive notifications of multimodal input events received from

a set of modality recognizers. Output restitution, be it multimodal or not, is managed by the client application. To give an example of an application using HephaisTK and SMUIML, a binding declaration in Java can be:

new HephaistkInitManager("client_app");

"client_app" refers to the application name every SMUIML script has to specify, and which is described in subsection 5.1. To enhance flexibility, callbacks to the client application are generated following the SMUIML script. Changing a few lines of the script can completely change the behavior of the multimodal toolkit, and as the callbacks are rather loosely tied, the developers are completely free to adapt their application, do testing on, or simply ignore, etc. those callbacks.

HephaisTK is built on an architecture based on software agents, as depicted in Figure 8. Agents are dispatched to manage individual modality recognizers, receive and encapsulate data from the recognizers, and send them to an individual central agent named the "postman". This postman agent centralizes all data coming from the dispatched recognizers agents in a database, and distributes the data to other interested agents in the toolkit, which can subscribe to be informed of specific types of data. An "integration committee" of three different agents achieves integration of data. A first agent manages fusion of input modalities, helped by a dialog agent; a fission agent encapsulates the fused data and sends it to the client application, which will be notified of the incoming data by means of an event listener.

HephaisTK is an integrated tool, which needs to be configured in order to send satisfactory information to its client application. Thus, a configuration file, describing the human-machine multimodal dialog wished for the client application, needs to be specified when using the toolkit. This configuration script also has other goals, for example the specification of which recognizers need to be used with the toolkit. Hence, the SMUIML language has been developed as a configuration language for the HephaisTK toolkit. The language is linked to this toolkit, and conversely the toolkit is linked to the language. Nevertheless, the SMUIML language has been created as being as generic as possible, and should be able to model multimodal human-machine interaction independently from the subsequent tool used to interpret the script.

6 Evaluation and positioning of SMUIML

The SMUIML language allowed modeling of a number of different multimodal use cases, from a music player with simple multimodal commands, to an application allowing classification and visualization of documents based on different criterias, to a drawing table with speech and tangible input. Thus, the ability to model and manage different multimodal applications was empirically verified. Nevertheless, the user-friendliness of SMUIML had still to be considered.

In order to compare the languages expressiveness and usability, masters degree students from a course on multimodal interaction were asked to devise their own version of a language allowing description of multimodal interaction. These students had all already a bachelor in computer science degree, and were pursuing their studies to get a master degree. The course on multimodal interaction delivered at the University of Fribourg (Switzerland) is an optional course in their curriculum and amongst the 30 students which were present the year this evaluation was done, six of them chose to delve deeper in multimodal interaction modeling. They already had an introductory course on multimodal interaction, but no extended experience. Thus, they represented developers with a strong interest and passing knowledge on multimodal interaction. These students had to first imagine a multimodal application, draw a storyboard detailing the main use cases of their application, and then they were given three weeks to invent a formalization to describe more deeply their application. They had no knowledge of SMUIML or any other description language on multimodal interaction, although they had already followed a course on multimedia-aimed description languages such as SMIL. The idea behind this task was to see how developers think about modeling multimodal interaction when they only have passing knowledge about multimodality.

First, most of the students tackled the problem by describing what "happens" in the system, i.e. events. Some of them built their language proposal only around events, others made a difference between "input" events and "result" events, and still others built chain of events. Nonetheless, non-specialists of multimodal interfaces, faced with the problem of describing multimodal human-machine interaction, show a tendency to first think about the actual events and their awaited results. Thereafter, most proposals offered a way to model human-machine interaction dialog, either by "knitting" events and actions to and from the system, or by describing fixed chains of events. Then, some students tried to give a description of the actual hardware used; some others did not see the interest of giving this level of detail. It is nonetheless to be noted that the students were to create a high-level formalization of multimodal human-machine interaction. In a more "complete" approach, some link to the actual hardware would have to be

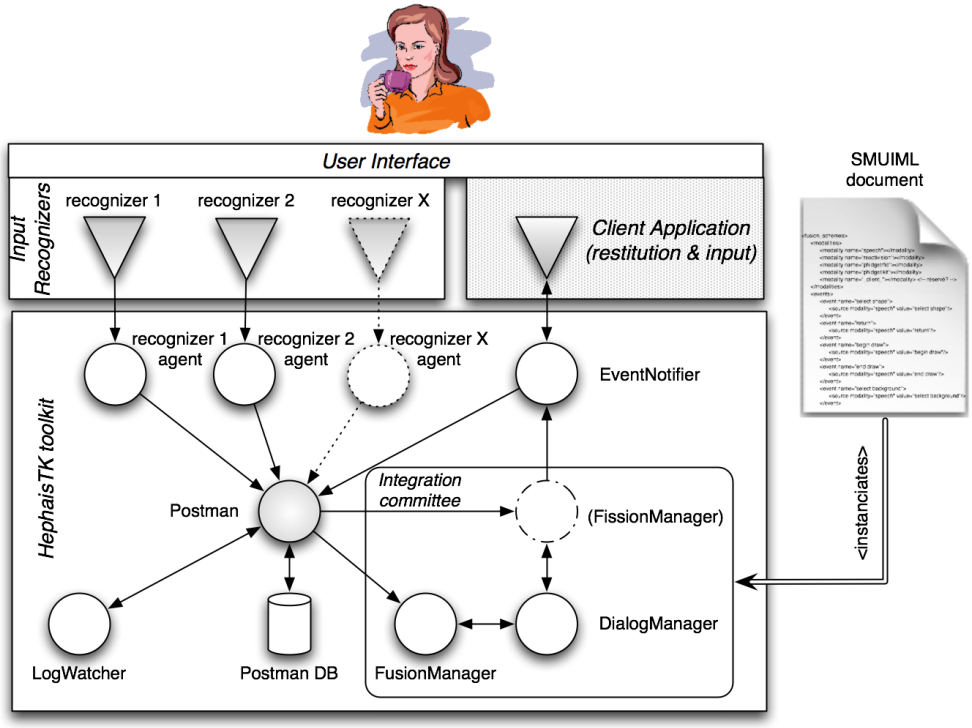

Fig. 8 HephaisTK toolkit architecture.

Table 3 The SMUIML language and the eight guidelines.

	SMUIML	<dialog>	<triggers>	<actions>	<recognizers>
G1. Abstraction levels	X	X	X	X	
G2. HMI modeling	X	X			
G3. Adaptability to context/user					
G4. Control over fusion process	X	X			X
G5. Time synchronicity	X	X			
G6. Error handling					
G7. Events management	X		X	X	
G8. I/O data representation	X				X

specified, and some of the students paid attention to this.

Finally, the students were confronted with SMUIML, and give a qualitative evaluation of it. Once again, the basic idea behind it was to get the opinion of non-experts in the field of multimodal interaction about the language. In regard to usability, two remarks were raised about the <dialog> part: first, with complex application, the whole context diagram could become tedious to analyze and read; second, every trigger, action or context is identified with a unique name, without distinction, which can easily lead to duplicate names. Thus, as usable as a description language can be, producing such documents for large-sized applications can become tedious. On a final note, interestingly enough, a final feature that was not addressed by most of the students is error handling. In fact, modality recognizers and multimodal integration are often considered as being error-free when designing the first draft of a multimodal system; it is obviously not the case, and even early design should take into account error reduction techniques such as tap-to-speak interfaces, restricted grammars or guided dialogues (See Bourguet [4] for more information on error handling strategies).

When confronted with the guidelines introduced in section 4, SMUIML takes into account abstraction levels by its three layers structure, events description with help of the <triggers> and <actions> elements, and representation of input sources with the <recognizers > elements. Control over the fusion process is allowed by specifying attributes in the <dialog> part: for example, the time frame in which a given meaning frame

has to be fused can be specified with help of the "lead-time" attribute. Time synchronicity aspects are managed by the different <par_or>, <par_and>, <seq_or> and <seq_and> elements. On the case of data modeling, as SMUIML is tied to a toolkit, all data modeling is supposed to be managed inside this toolkit. Sire et Chatty admit in this regard that data modeling can be present in multiple places, but could be present in XML modeling, for example to be compliant with EMMA. We believe data modeling could be based on EMMA description but should not be part of the multimodal interaction description itself, mainly for readability reasons. Control structures are a relevant point: basically, a state machine such as the one present in SMUIML dialog description is able to describe any application process, however control structures such as "switch" or "while" statements can enhance the readability of a given language. But control structures imply an imperative programming-oriented language; the way SMUIML dialog description was designed would make unnatural the use of standard control structures such as switch statements or loops: control structures are inherent to the state machine model. A complete overhaul of the language would be needed for the integration of such control structures. Adaptability to user and context is not taken into account yet in HephaisTK and SMUIML, though mechanisms to ease plasticity integration are present. Finally, error handling is also not yet present in the language, as can be seen in Table 3, but is a planned work, that will take place jointly with the enhancement of HephaisTK toolkit on error handling.

Finally, about guideline G* (Usability vs. expressiveness balancing), SMUIML was created with readability in mind, as well as keeping as much expressiveness as possible. Nonetheless, on the usability side, being able to use a Graphical User Interface to create a SMUIML script would go a long way toward better user-friendliness.

7 Conclusion and future works

This article discusses the advantages of having a modeling language for describing multimodal human-machine interaction. It further presents the state of the art and various approaches to this problem. From experiments and study of the state of the art, we propose a set of guidelines for the design of modeling languages for multimodal toolkits.

As mentioned in the article, such modeling languages role is not restricted to the configuration of a toolkit, but could serve as a modeling and communication language. Our thesis is that this kind of language should balance between readability (human-side) and expressiveness (machine-side). In fact, a balance should also be found between the different user-focused and system-focused guidelines: a single description language would have difficulty answering every guideline while still keeping readability.

When considering the SMUIML language in respect to the guidelines devised in section 5, guideline G3 (adaptability to user and context) and G6 (error handling) are lacking. Adaptability to user and context is not planned for the moment, although the software architecture on which HephaisTK is based upon could be extended to help take into account user profile and context when managing and interpreting input events. Plasticity on the output side would be harder to manage, and we have decided to leave it aside. Error handling is a planned future work, in parallel with the integration of advanced error handling in the HephaisTK toolkit, which will take into account errors coming from recognizers, mistakes from the fusion engine and exception. A second planned future work will concentrate on creation of a visual programming tool which would generate the SMUIML script. The challenge will lie in particular in keeping the abstraction levels and in promoting a careful modeling of the multimodal application, which were prominent benefits of the markup language. Furthermore, such a tool would enhance usability, while keeping expressiveness. Lastly, a field study with developers interactive multimodal systems will have to be achieved in order to assess the capabilities of SMUIML in a real context of use.

Acknowledgements Thanks to Raphael Boesch, Guillaume Gnaegi, Andreas Ruppen, Mario Sitz and Tony Svenson for their work on formalization languages in the frame of the MMI course at University of Fribourg. The work on HephaisTK and SMUIML are funded by the Hasler Foundation (http://www.haslerstiftung.ch) in the context of the MeModules project (http://www.memodules.ch), and by the Swiss National Center of Competence in Research on Interactive Multimodal Information Management – NCCR IM2 project (http://www.im2.ch).

References

1. Araki M, Tachibana K (2006) Multimodal Dialog Description Language for Rapid System Development. In: Proc. of the 7th SIGdial Workshop on Discourse and Dialogue, Sydney, Australia, July 2006
2. Bouchet J, Nigay L, Ganille T (2004) ICARE Software Components for Rapidly Developing Multimodal Interfaces. In: Conference Proceedings of ICMI'2004, State College, Pennsylvania, USA, October 2004, ACM Press, pp 251-258
3. Bourguet ML (2002) A Toolkit for Creating and Testing Multimodal Interface Designs. In: companion proceedings of UIST'02, Paris, Oct. 2002, pp 29-30

4. Bourguet ML (2006) Towards a taxonomy of error-handling strategies in recognition-based multi-modal human-computer interfaces. Signal Processing Vol. 86 Issue 12,December 2006
5. Cohen PR, Johnston M, Mcgee D, Oviatt S, Pittman J, Smith I, Chen L, Clow J (1997) QuickSet: multimodal interaction for distributed applications. In: Proc. of the Fifth ACM international Conference on Multimedia, Seattle, USA, 1997
6. Coutaz J, Nigay L, Salber D, Blandford A, May J, Young R (1995) Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties. In: Proceedings of the INTERACT'95 conference, S. A. Arnesen & D. Gilmore Eds., Chapman & Hall Publ., Lillehammer, Norway, June 1995, pp 115-120
7. De Boeck J, Vanacken D, Raymaekers C, Coninx K (2007) High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT, 4(2007), no. 2, September 2007, urn:nbn:de:0009-6-11615, ISSN 1860-2037
8. Dumas B (2009) SMUIML XML Schema reference. http://diuf.unifr.ch/diva/web/downloads/documents/other/2009/smuiml_v04.xsd. Accessed 28 Sept 2009
9. Dumas B, Lalanne D, Ingold R (2008) Prototyping Multimodal Interfaces with SMUIML Modeling Language. In: workshop User Interface Description Languages for Next Generation User Interfaces, CHI 2008, Firenze, Italy, pp 63-66
10. Dumas B, Lalanne D, Oviatt S (2009) Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In: Denis Lalanne, Jrg Kohlas eds. (2009). Human Machine Interaction, LNCS 5440, State-of-the-Art Survey, Springer-Verlag, Berlin/Heidelberg, pp 1-25
11. Garofolo J (2008) Overcoming Barriers to Progress in Multimodal Fusion Research. In AAAI Fall 2008 Symposium Proceedings
12. Greenberg S, Fitchett C (2001) Phidgets: easy development of physical interfaces through physical widgets. In:User Interface Software & Technology, CHI Letters 2001
13. Flippo F, Krebs A, Marsic I (2003) A Framework for Rapid Development of Multimodal Interfaces. In: Proc. of ICMI'03, Vancouver, BC, Nov. 5-7, 2003
14. Kaltenbrunner M, Bencina R (2007) ReacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. In Proceedings of the first international conference on "Tangible and Embedded Interaction" (TEI07). Baton Rouge, Louisiana, 2007
15. Katsurada K, Nakamura Y, Yamada H, Nitta T (2003) XISL: a language for describing multimodal interaction scenarios. In: Proc. of the 5th Intl. Conference on Multimodal interfaces (ICMI 2003), Vancouver, British Columbia, Canada, November 05 - 07, 2003
16. Ladry JF, Palanque P, Basnyat S, Barboni E, Navarre D (2008) Dealing with Reliability and Evolvability in Description Techniques for Next Generation User Interfaces. In: Proceedings of User interface description languages for next generation user interface workshop at CHI08, Florence, Italy, 2008
17. Oviatt, SL (2008) Multimodal interfaces. In: J. Jacko, J., Sears, A. (eds), The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, 2nd edition, CRC Press, 2008, chap. 14, pp. 286-304
18. Paternò F, Santoro C, Mäntyjärvi J, Mori G, Sansone S (2008) Authoring pervasive multimodal user interfaces. In: Int. J. Engineering and Technology, Vol. 4, No. 2, pp 235-261
19. Serrano M, Nigay L, Lawson JYL, Ramsay A, Murray-Smith R, Denef S (2008) The OpenInterface framework: a tool for multimodal interaction. In: Adjunct Proceedings of CHI2008 (April 5-10, Florence, Italy), ACM Press, pp 3501-3506
20. Shaer O, Jacob RJ, Green M, Luyten K (2008) User interface description languages for next generation user interfaces. In: CHI '08 Extended Abstracts on Human Factors in Computing Systems (Florence, Italy, April 05 - 10, 2008). CHI '08. ACM, New York, NY, pp 3949-3952
21. Sire S, Chatty C (2002) The Markup Way to Multimodal Toolkits, W3C Multimodal Interaction Workshop, 2002
22. Stanciulescu A, Limbourg Q, Vanderdonckt J, Michotte B, Montero F (2005) A transformational approach for multimodal web user interfaces based on UsiXML. In: Proc. of the 7th Intl. Conference on Multimodal interfaces (ICMI 2005), Torento, Italy, October 04 - 06, 2005
23. Windgrave C (2008) Chasm: A Tiered Developer-Inspired 3D Interface Representation. In Proceedings of User interface description languages for next generation user interface workshop at CHI08, Florence, Italy, 2008