Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces

Bruno Dumas[1], Denis Lalanne[1], Dominique Guinard[2], Reto Koenig[1], Rolf Ingold[1]

[1] DIVA research group
University of Fribourg
Boulevard de Pérolles 90
1700 Fribourg, Switzerland
{firstname.lastname}@unifr.ch

[2] Information Management
ETH Zurich / SAP Research
Sonneggstrasse 63
8092 Zurich, Switzerland
dguinard@ethz.ch

ABSTRACT
This paper reviews the challenges associated with the development of tangible and multimodal interfaces and exposes our experiences with the development of three different software architectures to rapidly prototype such interfaces. The article first reviews the state of the art, and further compares existing systems with our approaches. Finally, the article stresses the major issues associated with the development of toolkits allowing the creation of multimodal and tangible interfaces, and presents our future objectives.

Author Keywords
Multimodal and tangible interfaces, multimodal interaction, software engineering.

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces, Prototyping, Theory and methods.

INTRODUCTION

Tangible interfaces have shown much promise since the founding works of Ullmer and Ishii [11]. They have shown particularly interesting concepts when used to create user interfaces which mix tangible interaction with other modalities, such as speech or gesture based interaction (e.g. Papier-Mâché [13]). However the development of such multimodal interfaces remains a difficult task needing knowledge in various state-of-the-art domains like speech recognition, RFID hardware management or gesture tracking. Furthermore, the integration of natural communication means recognition systems and of synthesizers into an embedded application is no trivial task: multiple input modalities imply potential combinations of meaning, i.e. a need for a fusion engine; in particular, the management of synchronized input should be carefully handled. Likewise, multiple or "non-standard" available output modalities should lead to careful choice to return the information to the user, i.e. a need for a fission engine, depending on the context of use and/or on the user profile.

Hence, tools giving access to parts of a multimodal system would relieve some of the tasks implied by the creation from the ground up of a full tangible and multimodal application. Such a tool could take different forms; from straightforward, but practical ones to far more developed ones. In this article, we will compare three different forms such a toolkit could take: a simple and rugged extension to a widely used GUI toolkit, an integrated tool driven by an internal finite state machine describing the user-machine dialog, and finally a pluggable toolkit allowing the control of various input modalities by means of an XML configuration file.

This article first develops the problematic of time synchronization on the side of the input (fusion) and of the output (fission); it then reviews existing multimodal and tangible toolkits. The article further presents the three different architectures of toolkits we developed that could give rise to such multimodal applications development tools. Two of them have been derived from actual multimodal applications, and the last one is a toolkit currently in development. The article then compares these systems with state-of-the-art ones in a table, emphasizing on their major characteristics and on the trade-off between usability and expressiveness of a toolkit. Finally the article concludes with the major challenges of this work and a roadmap to achieve them.

PROBLEMATIC OF TIME SYNCHRONIZATION

As pointed out in the introduction, mixing tangible interaction with other modalities such as speech or gesture recognition involves taking into account time synchronization of modalities, i.e. fusion and fission of the modalities. In this section, standard fusion and fission

mechanisms will be detailed, in order to introduce the toolkits presented in the next sections.

Fusion Mechanisms

One of the main issues of multimodal design is the way fusion of multimodal human messages is performed to reach a robust interpretation. The root of the problem comes from the way input is managed in multimodal interfaces: in standard (WIMP) interfaces, the events, as mouse clicks or keyboard commands, can be processed one at a time, without having to deal with problems of references and co-references. In multimodal systems, not only do these problems arise, but they are in fact a core property of multimodal interaction. A classic example is Bolt's synergic "Put that there" [3], in which a vocal command ("Put") must be actively linked to a pair of pointing events, achieved in a spatially-aware modality. As a matter of fact, other complex problems appear beyond the "Put that there" example: users rarely input at the same exact time co-referenced events. Solving those problems of reference and co-reference is the subject of modality fusion, which we will now develop.

Three main fusion levels are generally considered: data, feature and decision fusion. *Data-level fusion* is not *per se* an inter modality fusion mode, but is used when dealing with multiple signals coming from a similar type of source (e.g. two cameras). *Feature-level fusion* is the favourite type of fusion when tightly coupled and synchronised modalities are to be fused. *Decision-level fusion* is considered as the most used type of fusion in multimodal applications. The main reason is its ability to manage loosely-coupled modalities like, for example, pen and speech interaction.

Decision-level fusion-based multimodal systems have been built following a number of different fusion mechanisms, some of which have appeared recently. All those mechanisms have as a common objective to solve fusion at the semantic level and thus to handle issues such as potentially erroneous interpretations coming from different modality channels, synchronisation and reference/co-reference problems. Some examples of decision-level fusion mechanisms are the following ones: rules-based fusion, frame-based fusion, and hybrid symbolic-statistical architectures [6].

When attempting to mix tangible interaction with other modalities, decision-level fusion is preferred over data level and feature-level fusion, mostly because of the high-level interpretation achieved on data coming from sometimes very diverse input source.

Fission techniques

Fission techniques allow a multimodal application to adapt its output messages according to context and user profiles [18]. Technically speaking, fission consists of three tasks:

- Message construction, where the information to be transmitted to the user is created;
- Output channel selection, where relevant restitution interfaces are selected according to context and user profile;
- Construction of a coherent and synchronized restitution.

Less research has been done on multimodal fission than on fusion. Most applications use few output modalities and, consequently, employ straightforward fission mechanisms. Nonetheless, when multiple output modalities such as text-to-speech synthesis, audio cues, visual cues or animated characters are available, output selection becomes a tricky task. An interesting example of fission strategy applied to multiple output modalities is shown in the SmartKom project [18], where a 3D animated character dialogues with the user: this involved synchronization of gestures of the character, lips of the character and spoken text.

CARE properties

CARE properties have been proposed by Coutaz & al. in their paper entitled "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties" [7]. Four properties are introduced, to describe the potential relationships between different input modalities and their potential meaning in a multimodal system. These properties are:

- Complementarity: multiple complementary modalities are necessary to grasp the desired meaning (e.g. "put that there" would need both pointing gestures and voice in order to be resolved);
- Assignment: only one modality can lead to the desired meaning (e.g. the steering wheel of a car is the only way to direct the car);
- Redundancy: multiple modalities, even used simultaneously, can be used individually to lead to the desired meaning (e.g. user utters a "play" speech command and pushes a button labeled "play", but only one "play" command would be taken into account);
- Equivalence: multiple modalities can all lead to the desired meaning, but only one can be used at a time (e.g. speech or keyboard can be used to write a text).

Those CARE properties have revealed themselves as a tool used by a number of multimodal toolkits to help formalize relationships between different modalities.

RELATED WORKS

In this section we present a state of the art of tools and frameworks dedicated to the rapid creation of multimodal interfaces. Toolkits dedicated to creation of interfaces including multiple modalities, tangible or not, are covered in the first part. The second part of this section is dedicated to toolkits specifically geared toward creation of tangible interfaces.

A number of toolkits for creating tangible and/or embedded applications have seen the light since the founding works of Ullmer and Ishii. As shown by Mazalek [15], three main tendencies have been followed: hardware prototyping, groupware-focused and integrated application.

Phidgets [10] and iStuff [1] are typical examples of the hardware prototyping approach, offering developers a set of hardware components, such as buttons, knobs or LEDs, and software drivers to use those components.

Groupware-focused tools, such as SDGToolkit [17] offer tools allowing creation of group-oriented applications; examples of such groupware-focused tools also include multi-touch tables.

Finally, researchers have begun to work on a mid-way, integrated application toolkit approach; examples include Papier-Mâché [13] and Synlab API [15].

As interesting as these toolkits are, problems arise when you wish to mix them with other modalities such as speech: hardware components and groupware-oriented applications focus on their original tasks and generally do not provide opportunities to accept third party recognizers or an external data source. Hence, if we want to enrich tangible applications with other modalities, we need toolkits able to take into account data coming from very different sources, and still be able to fuse data coming from those sources.

Since intensive work has begun on multimodal systems, the need for tools allowing the rapid creation of multimodal applications has grown. Hence, work has been achieved by researchers toward the creation of such tools. Krahnstoever and al. [14] proposed a framework using speech and gesture to create a natural interface. The output of their framework was to be used on large screen displays enabling multi-user interaction. Fusion was done using a unification-based method. Bourguet [5] endeavoured in the creation of a multimodal toolkit in which multimodal scenarios could be modelled using finite state machines. This multimodal toolkit was composed of two components, a graphical user interface named IMBuilder which interfaced the multimodal framework itself, named MEngine. Flippo and al. [8] also worked on the design of a multimodal framework, geared toward direct integration into a multimodal application. One of the most interesting aspects of their work is the use of a parallelisable application-independent fusion technique. The general framework architecture is based on agents, while the fusion technique itself uses frames. Lastly, Bouchet and al. [4] proposed a component-based approach called ICARE thoroughly based on the CARE [7] design space presented above. These components cover elementary tasks, modality-dependent tasks or generic tasks like fusion. Finally, communication between components is based on events. The components-based approach of ICARE has been used to create a comprehensive open-source toolkit called OpenInterface [2]. In its current state (v0.2), OpenInterface seems to inherit most of the architecture of ICARE. It is finally worth noting that the World Wide Web Consortium (W3C) has introduced its W3C multimodal interaction framework. This theoretical framework describes major components involved in multimodal interaction, as well as potential or existent markup languages used to relate those different components. While somewhat centered on web-based multimodal interaction, many elements described in this framework are of practical interest for multimodal HCI practitioners, such as the W3C EMMA markup language.

THREE POSSIBLE ARCHITECTURES FOR TOOLKITS

In this section we present three toolkits we implemented and experimented. The Java Swing MM extension proposes to extend Java Swing with other modalities so that programmers do not have to learn a new environment. The drawback of this approach lies in the multimodal expressiveness. The *Service Counter System* uses a finite state machine to enable multimodal fusion and as a way to program interaction scenarios, but at the expense of a lot of "hard-coding". Finally, *HephaisTK*, currently under development, uses in its preliminary version a multi-agent architecture to enable an efficient communication between recognizers and the application created with help of the toolkit. Those three toolkits try to mix tangible interaction (coming respectively from Phidgets [10], Papier-Maché [13] or Reactivision [11]) with other forms of input. We present in this section each system, emphasizing on their major characteristics. The next section finally compares these systems with related works.

Java Swing MM extension: usability

A first attempt at providing an architecture to help creation of tangible and multimodal interfaces was achieved by extending the Java Swing GUI framework. The goal of the Java Swing MM extension was to come up with a simple and rugged solution toward rapid prototyping of multimodal and tangible interfaces, based on a standardized GUI framework.

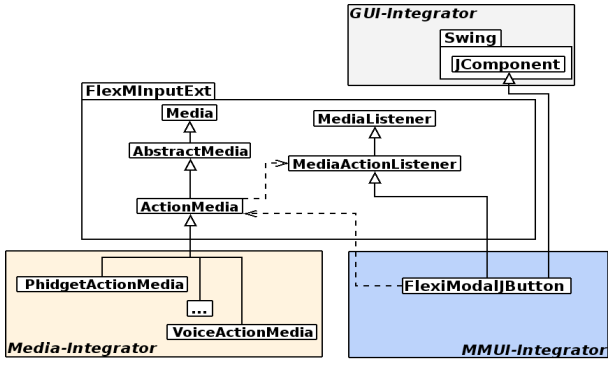

Figure 1. Java Swing MM Extension: MMIButton

The choice of the Java Swing GUI framework as a basis was a default choice at the time we experimented with this approach: in fact, any sufficiently modular GUI framework should be able to be applied the same approach.

The Java Swing MM Extension provides a specific developers' design-pattern which improves any graphical component into a multimodal and/or tangible input component. The strength of this pattern lies in the separation of the different development tasks:

- The GUI-Integrator, which builds new graphical components, does not have to know anything about the actual input recognizers used.
- The Media-Integrator only has to implement the desired media-interface and does not have to know about fusion techniques.
- The MMUI-Integrator only needs to extend the graphical component in order to integrate the desired media listener. The chosen fusion behavior is implemented there.

Any developer can then use the multimodal components as the new components can be handled just as if they were standard GUI components.

The proof of concept for the *Input Extension* was performed through the augmentation of the *Swing JButton* towards a *MMIButton* (The *JButton* is a simple graphical component provided by the *Swing*-Framework used in Java).

The *MMIButton* component has been equipped with the possibility of being activated by keyboard, mouse, external *Phidgets* [9] hardware interface components (joystick, physical button, etc.) and voice. An application level fusion (most recent event counts) was used.

The *Output Extension* provides another specific design-pattern aiming for the same goal: fission at runtime, separation between the different integration development processes, straightforward usage of the designed output synthesizers.

An architecture such as the one proposed by the Java Swing MM extension allows developers to quickly add multimodal and tangible input to an existing application, or develop such an application from the ground up with a minimum of new knowledge to grasp. Conversely, applications built using the Java Swing MM extension are limited to multimodal or tangible input mapped to specific commands. Regarding synchronicity of modalities, equivalence of modalities is offered right away (see CARE properties earlier in the article), as long as all modal or tangible commands lead to a specific action; however complementarity can be achieved only in specific cases, due to the clear separation between the multiple multimodally enhanced components.

SCS, the Service Counter System: expressiveness

The SCS is a toolkit addressing the design and implementation of multimodal user interfaces. It aims to solve the issue in a programmatic way, i.e. it helps the programmer creating a multimodal user interface. Towards this goal it provides:

- An extensible object-oriented abstraction layer (or a framework) of existing input and output libraries.
- A central state machine that can be used to model the interaction flows.

Although designed to serve the design of a broad variety of multimodal user interfaces, the SCS was primarily created to be used for use-cases in which a clear and deterministic interaction flow can be extracted. Thus, it is particularly adapted, but not limited, to semi of fully embedded use-cases such as modeling a rental service, an information desk, an interactive guide, etc.

The SCS is articulated around a central state-machine. Using this automata-generator designed for multimodal interfaces the programmer can describe the human-computer and computer-human interaction flows. In other words, she can design the system's states and transitions. For instance, using the state-machine, one could enforce the fact that after authentication (i.e. after changing the state), a customer can either rent a DVD (first possible transition) or return it (second possible transition). As an example, Fig. 2 represents the automata designed for the smart librarian use case that we will describe later.

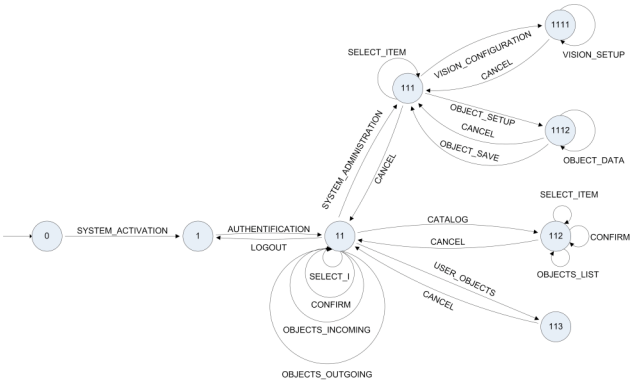

Figure 2. A sample automata modeling the interaction flow.

As mentioned before, the SCS not only provides a way to design the interaction flows of the user interface but also offers to abstract the concerns related to the input and output channels. The abstraction layer is offered by the so called Input and Output drivers. They represent a uniform way for accessing the underlying components and libraries. Each concrete library used as input or output communicates with a SCS driver extending the InputDriver, respectively the Outputdriver class.

This way the "phobs" of the *Papier-Mâché* [13] library, the "event handlers" of the *Phidgets* [10] or the "tags" of the *Sphinx* [19] speech recognition engine can be accessed in a uniform manner by the programmer.

In terms of synchronization of modalities, the SCS enables equivalence of the input channels (see CARE model above). Thanks to the state-machine coupled to both the InputDrivers and the observer pattern, the end user can accomplish an action by sequentially choosing one of the

1..n different modalities activated in each state. This way, browsing the options of a menu bar can be operated first using the user's voice (e.g. with the *Sphinx* SCS driver), then using a physical joystick (e.g. with the *Phidget* Joystick SCS driver), and eventually by tracing the user's arms movements (e.g. with the *Papier-Mâché* SCS driver).

Furthermore, the SCS offers multimodal fission functionalities. The system constructs a feedback to the user's actions using various different output channels. As before, this is operated using the OuputDrivers architecture coupled with notification methods provided by the InputDrivers.

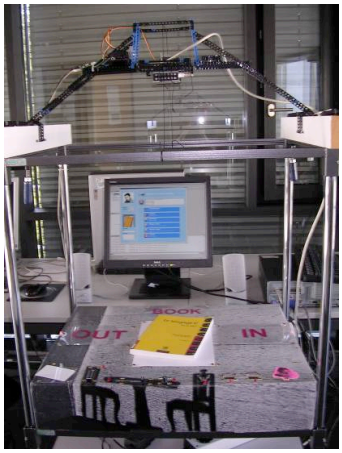

Figure 3. The Smart Librarian use case.

As use case, the Smart Librarian (see Fig. 3) project is a concrete multimodal and tangible application using the SCS toolkit. It models a self-service library. The idea behind it is to emphasize multimodality as a help towards more accessible application software and more natural human-computer interaction schemes. This is particularly important since the targeted user group of the Smart Librarian comprises people without exhaustive computer-skills or disabled people.

Thanks to the SCS architecture, the user can use the communication channel he feels to be the most adapted at each step. As an example consider the user wanting to borrow a book. In order to start this action he can either: talk to the system (saying something similar to "I want to borrow this book" or just "borrow"), use the physical widgets (joystick, button), dispose a book on the "out" zone (monitored by an RFID reader), or make a gesture towards the "out" zone (monitored by the video camera), etc. In order not to confuse the reader with all the activated channels for each state the Smart Librarian uses the fission mechanisms provided by the SCS to turn on signalization LEDs as well as icons on the display and vocal information.

As opposed to the Java Swing MM extension, the SCS toolkit favours as much expressiveness as possible: the built-in state machine allows modelling complex human-computer dialog schemes, and the input and output drivers offer extended versatility to the developer using the SCS toolkit. The price of this expressiveness is a tool asking extended expertise from the developer using it, and the necessity to fully integrate the SCS toolkit into one's multimodal or tangible application.

HephaisTK, an agent-based toolkit: looking for balance

Following the experimentations achieved with the two toolkits surveyed above, the Java Swing MM extension and the SCS toolkit, a third toolkit named HephaisTK was created. This toolkit tries to balance expressiveness and usability.

HephaisTK is intended to be a toolkit allowing rapid creation of multimodal interfaces, offering a predefined set of recognizers as well as the possibility to plug into the toolkit any other modality recognizer, as long as it complies with a given set of conditions, e.g. communication with the toolkit by means of the W3C EMMA language. HephaisTK is designed in the Java programming language, as a multi-platform toolkit. HephaisTK will also offer different fusion mechanisms to allow meaning from incoming recognizers to be extracted, and passed to potential client applications. Finally, free use of HephaisTK by any interested person will be guaranteed and available through GPL licensing.

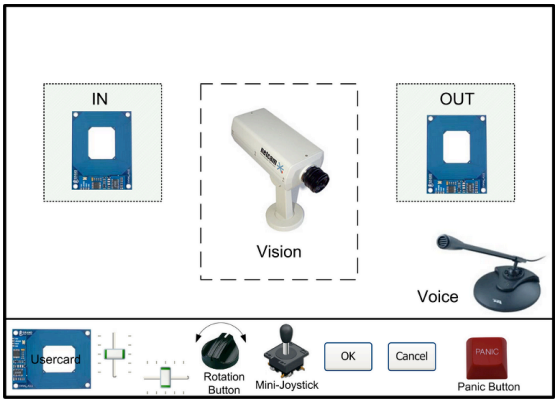

Figure 4. Concept of the Smart Librarian use case.

In its current state, HephaisTK is built upon a software agent system. Each time a new recognizer or synthesizer is plugged into the toolkit, an agent is dispatched to monitor it. Agents manage communication between the different parts of the framework, from the input recognizer to the meaning extraction engines to the output modules. Agents are also used because of their ability to transit from one platform to another.

HephaisTK uses a central blackboard architecture (see Fig. 5). A "postman" centralizes each message coming from the different input recognizers and stores it into a database. Agents interested in a specific type of message can subscribe to the postman, which will accordingly redistribute received messages. Fusion of input modalities is achieved through meaning frames. When a developer wants to use HephaisTK toolkit to monitor input

recognizers, he has to declare the toolkit by means of event listeners. The toolkit manages fusion of modalities, as well as user-machine dialog, by means of an internal finite state machine paradigm; if the general dialog scheme is fixed, behaviour of the fusion engine can be tuned by the developer to match the different CARE properties. The fusion and dialog managers of HephaisTK are scripted by means of a SMUIML (*Synchronized Multimodal User Interfaces Modelling Language*) XML file [8]. This language has been created as a means for the developers wishing to use HephaisTK to easily access the deeper functionalities of the toolkit without having to delve into the code. A typical SMUIML declares recognizers, triggers and actions, and the user-machine dialog in the form of a finite state machine calling those triggers and actions. CARE properties are fully integrated into SMUIML and can be used to specify the way modalities will have to be fused, for example in a parallel or complementary way.

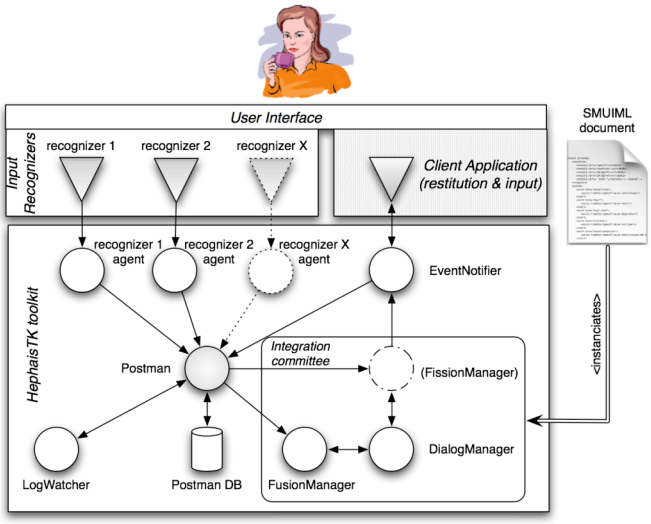

Figure 5. HephaisTK toolkit architecture.

Finally, the support of W3C EMMA language allows plugging of any new human-machine communication means recognizer or synthesizer, able to manage communication via EMMA, in HephaisTK.

This toolkit had to find a balance between usability and expressiveness for the user interfaces developer wishing to use it; in this regard, HephaisTK has been created as able to plug itself into an existing or new application without heavy modifications; the programming is done by means of an XML file able to describe rich interactions. This balance does not come for free, however: first, as the developer's application and the toolkit are separated, the developer has to be careful that the toolkit and her application are in a same "state"; also, the toolkit is a bulky software, maybe too much bulky for simple use cases.

Use cases

In order to get a qualitative evaluation of the three architectures, we tried to model with these architectures two different, already existing multimodal applications.

A first simple use case allowing control of a music player application via speech commands, standard WIMP interface elements and tangible RFID tagged objects has been implemented. This application allowed simple interactions, such as "play", "pause", or "next track" commands, and offered different ways to express the commands. For example, a user could input the desired with help of a tagged object while issuing a "play" command by voice.

Such a simple use case could be easily modelled by means of the Java Swing MM extension: most of the commands were mapped to a given GUI element and to a set of tangible or vocal commands. HephaisTK toolkit could also model this application, provided some work to create the SMUIML script. Conversely, modelling this simple use case by means of the SCS toolkit needed a complete rewrite of the application.

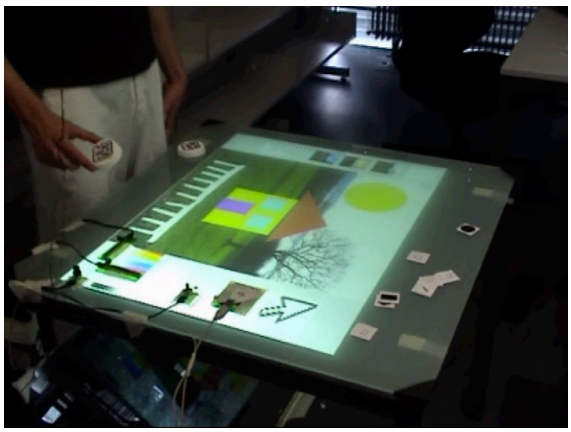

Figure 6. The XPaint drawing table.

A more complex use case followed in the form of the XPaint drawing table (see Fig. 6). On this table, two physical artefacts allow the users to draw on the table a set of shapes or tools selected by means of RFID-tagged tangible objects. Commands can also be selected by means of vocal commands, recognized with help of the Sphinx speech recognition toolkit. Additionally, specific commands as selection of colour or line width are expressed through specific hardware input devices like Phidgets sliders.

The XPaint table could not be modelled entirely using the Java Swing MM extension due to the numerous tight links between the different modalities used: lots of operations (such as painting a coloured shape, which needs speech, Phidgets and the physical artefacts) require complementary fusion of multiple inputs, which the Java Swing MM extension couldn't achieve. Modelling the human-computer dialog could be achieved using the SCS toolkit, but

revealed itself a tedious work. In regard to this, the modelling by means of a SMUIML script proved easier to create, due to the level of abstraction provided by the HephaisTK scripting language. Most of the existing XPaint application code could be also re-used thanks to the clear application-HephaisTK separation, whereas an implementation by means of the SCS toolkit would have needed rewriting much of the program.

SUMMARY AND MAJOR CHALLENGES

Table 1 summarizes the different characteristics of the systems described either in the state of the art, or in the three architectures described above: extensible systems (i.e. toolkits) have the potential ability to add other input modalities in a practical way. Pluggability refers to the ability of a toolkit to insert itself into an architecture without having to rewrite everything. The other characteristics are self-explanatory.

Regarding our three architectures, some interesting aspects have been discovered. The Java Swing MM Extension allows developers to create without much effort multimodally-enhanced interfaces, at the expense of dialogue capabilities: only a simple "one-command-to-one-action" scheme is allowed.

On its side, the SCS toolkit is still a prototype and a number of improvements would be required to make it more productive. However, the SCS proposes an original basis for a programmatic toolkit (or framework) to alleviate the implementation of multimodal interfaces for which a clear deterministic interaction flow can be extracted.

OpenInterface and HephaisTK both offer modularity, one through a component-based approach, the second with help a software agents approach. Moreover, those toolkits give developers more genericity in the way synchronization of input modalities is expressed, compared to the Java Swing MM Extension or Service Counter System. They differ mainly in architecture and technical choices: OpenInterface is a complete multimodal toolkit managing every part of the design of a multimodal application, whereas HephaisTK focuses only on management and fusion of multiple inputs, presenting itself to a client application as a plug-in.

FUTURE WORKS AND CONCLUSIONS

Choosing a type of architecture for a given tangible and multimodal application heavily depends on the type of application itself. Giving some multimodal capabilities to a standard WIMP application does not necessarily require a high level architecture: as the Java Swing MM Extension shows, a simple layer built upon a widely available GUI library already gives interesting multimodal capabilities to an application. However, this type of add-on considers only equivalence/redundancy of modalities and not complementarity.

In order to enable complementary usage of modalities, requiring proper synchronization and fusion mechanisms, one step further is to consider architectures built on finite state machines, like MEngine or the Service Counter System presented in this paper. This type of architecture allows the creation of multimodal interfaces with advanced user-machine dialog, but to the detriment of usability and readability. Also, the user will have to follow strictly the course of events described by the finite state machine.

In the near future, we will concentrate most of our efforts

	ICARE – OI [7]	OpenInterface [2]	IMBuilder/ MEngine [5]	Flippo et al. [8]	Krahnstoever [14]	Quickset [8]	Phidgets [8]	Papier-Mâché [8]	Java Swing MM Extension	Service Counter System	HephaisTK
Architecture traits											
Finite state machine			x							x	x
Components	x	x					x		x		
Software agents				x		x					x
Fusion by frames					x						x
Symbolic-statistical fusion						x					
Programming mechanisms											
Programming via "hard coding"					x	x				x	
Programming via API				x			x	x	x		
Programming via configuration file											x
Visual Programming tool	x	x	x								
Characteristics											
Extensibility		x	x	x					x	x	x
Pluggability							x		x		x
Reusable components	x	x									x
Open Source	x	x						x		x	x

Table 1. Comparison of different multimodal and tangibles toolkits and architectures.

toward the enhancement of HephaisTK. We believe an approach striving to balance usability and expressiveness will be of greater interest to HCI practitioners than other approaches based solely on one of these two qualities. We hence plan to collect and integrate a wider range of available state-of-the-art recognizers and synthesizers, e.g. for pen or gesture tracking. Work will also continue on the SMUIML scripting language used in HephaisTK to model human-computer dialog. Further, we will explore novel fusion methods development as well as fission mechanisms taking into account the context and the user profile. During this work, two specific use cases will be considered: (1) a multimodally enhanced smart meeting room for business use case and (2) a smart living room for home use case. Finally, along the way, user evaluations will target the toolkit usability, and multimodality studies will focus on deeper studies on the usage of multimodality.

ACKNOWLEDGEMENTS

Thanks to Luca Zingg, Daniele Della Bruna and Yannick Thiessoz for their work on the XPaint drawing table. HephaisTK is funded by the Hasler Foundation (http://www.haslerstiftung.ch) in the context of the MeModules project (http://www.eif.ch/memodules), and by the Swiss National Center of Competence in Research on Interactive Multimodal Information Management – NCCR IM2 project (http://www.im2.ch).

REFERENCES

1. Ballagas, R., Ringel, M., Stone, M., Brochers, J. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Envirnments. *Proceedings of CHI'03*, 2003.
2. Benoit, A., Bonnaud, L., Caplier, L., Damousis, I., Tzovaras, D., Jourde, F., Nigay, L., Serrano, M., Lawson, J.-Y. Multimodal Signal Processing and Interaction for a Driving Simulator: Component-based Architecture. *Journal on Multimodal User Interfaces*, Vol 1, No 1 (2007).
3. Bolt, R. Put-that-there: voice and gesture at the graphics interface. *Computer Graphics, 14(3)*, 1980.
4. Bouchet, J., Nigay, L., and Ganille, T. ICARE Software Components for Rapidly Developing Multimodal Interfaces. *Proceedings of ICMI'2004,* State College, Pennsylvania, USA, Oct. 2004.
5. Bourguet, M. L. A Toolkit for Creating and Testing Multimodal Interface Designs. *Proceedings of UIST'02*, Paris, Oct. 2002, pp. 29-30.
6. Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J. QuickSet: multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM international Conference on Multimedia*, Seattle, USA, 1997.
7. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. and Young, R. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties. *Proceedings of INTERACT'95 conference*, Lillehammer, Norway, June 1995, pp. 115-120.
8. Dumas, B., Lalanne, D., Ingold, R. Prototyping Multimodal Interfaces with the SMUIML Modeling Language. To be published in *Proc. of CHI 2008 workshop : User Interface Description Languages for Next Generation User Interfaces (organizers: O. Shaer, R. Jacob, M. Green, K. Luyten)*, Florence, Italy, 2008.
9. Flippo, F., Krebs, A. and Marsic, I. A Framework for Rapid Development of Multimodal Interfaces. *Proc. Of ICMI'2003*, Vancouver, BC, Nov. 5-7, 2003.
10. Greenberg, S., Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. *User Interface Software & Technology, CHI Letters 2001*.
11. Ishii, H., Ullmer, B. Tangible bits: toward seamless interfaces between people, bits and atoms. *Proceedings of CHI'97*, New York, USA, 1997.
12. Kaltenbrunner, M., Bencina, R. ReacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. *Proceedings of TEI'07*. Baton Rouge, Louisiana, 2007.
13. Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. Papier-Mâché: Toolkit Support for Tangible Input. *CHI Letters: CHI 2004*. 6(1).
14. Krahnstoever, N., Kettebekov, S. and Yeasin, M. and Sharma, R. A real-time framework for natural multimodal interaction with large screen displays. *Proc. of ICMI 2002*, Pittsburgh, PA, USA, Oct. 2002.
15. Mazalek, A. Tangible Toolkits: Integrating Application Development across Diverse Multi-User and Tangible Interaction Platforms. *Let's Get Physical Workshop, DCC'06,* Eindhoven, Netherlands, July 2006.
16. Nigay, L., Coutaz, J. A design space for multimodal interfaces: concurrent processing and data fusion. *Proc. INTERCHI 1993 Human Factors in Computing Systems* Amsterdam.
17. Tse, E., Greenberg, S. Rapidly prototyping Single Display Groupware through the SDGToolkit. *Proceedings of the Fifth Conference on Australasian User interfaces (AUIC'04). Dun*edin, NZ, 2004.
18. Wahlster, W. SmartKom: Fusion and Fission of Speech, Gestures, and Facial Expression*s*. *Proc. of the 1st International Workshop on Man-Machine Symbiotic Systems*, Kyoto, Japan, 2002. p. 213-225.
19. Walker, W., Lamere, P., Kwok P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. Sphinx-4: A flexible open source framework for speech recognition. Sun Microsystems*, Tech. Rep. TR-2004-139*, 2004.