Component-Based High Fidelity Interactive Prototyping of Post-WIMP Interactions.

Jean-Yves Lionel Lawson[1], Mathieu Coterot[2], Cyril Carincotte[3], Benoît Macq[1]

[1]Communications and Remote Sensing Laboratory (TELE), Université catholique de Louvain (UCL), Belgium
{firstname.lastname}@uclouvain.be

[2] TCTS Lab, Université de Mons (UMONS), Belgium
mathieu.coterot@umons.ac.be

[3] Image department, Multitel asbl, Belgium
carincotte@multitel.be

ABSTRACT

In order to support interactive high-fidelity prototyping of post-WIMP user interactions, we propose a multi-fidelity design method based on a unifying component-based model and supported by an advanced tool suite, the OpenInterface Platform Workbench. Our approach strives for supporting a collaborative (programmer-designer) and user-centered design activity. The workbench architecture allows exploration of novel interaction techniques through seamless integration and adaptation of heterogeneous components, high-fidelity rapid prototyping, runtime evaluation and fine-tuning of designed systems. This paper illustrates through the iterative construction of a running example how OpenInterface allows the leverage of existing resources and fosters the creation of non-conventional interaction techniques.

Categories and Subject Descriptors

H5.2 [Information interfaces and presentation]: User Interfaces. – Prototyping. D2.2 [Software Engineering]: Design Tools and Techniques – Modules and interfaces; user interfaces. D2.m [Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software.

General Terms

Design, Experimentation, Human Factors.

Keywords

Prototyping, component-based architecture, reusable software component, multimodal interfaces, multimodal software architecture, OpenInterface, SKEMMI, post-WIMP, interactive design.

1. INTRODUCTION

Since pioneering studies in the area of Human-Computer Interactions (HCI), there have been leaps in technologies such as graphical user interfaces, input devices, computer vision, speech processing, artificial intelligence, etc. Today we are witnessing a growing interest and a shift from traditional computer interfaces (WIMP) to advanced, entertaining, pervasive and more natural systems (post-WIMP). Due to its cross-disciplinary nature, the design and development of advanced human-computer interactions is not only conceptually but also practically a very challenging task. Seamless reuse and adaptation of the plethora of existing technologies is a key to successful exploration of non-conventional techniques. However, to this date, researchers and practitioners willing to explore new paths, design and implement innovative systems usually have to build ad hoc tools in order to replicate existing results or implement new interaction techniques.

In HCI, the lack of tools and integration methods enforcing replicable practice and reuse of current technology is a recognized problem [9][1] and can explain the low penetration in industrial applications. Seminal works present tools for the iterative design of multimodal systems [11][3]. More recent works focus either on hardware prototyping [4] or tackles specific issues (e.g. visual complexity, [5]) and still requires commitment to an implementation technology.

Most of these solutions require commitment to a specific technology (e.g. programming language, device toolkit), or support a limited number of interaction modalities such as voice, pen, text, and mouse. Moreover none of them provides a reusable platform for experimenting with various type of software component without the burden of complete re-implementation to match a chosen runtime technology. Finally the collaboration of different stakeholders around a central tool in order to design advanced techniques is also not tackled.

1.1 Prototyping Interactions

The development of interactive systems is a complex task; multimodal interactions are even more complex as underlying concepts are still not well known. Hence the development of natural interactions techniques is also hard and involves the integration of the domain-specific functionality with close collaboration between domain experts.

Prototypes (mock-ups or functional) of the future system can be very helpful in refinement of requirements capture by allowing users to test future systems and provide feedback. Low-fidelity prototypes are cheap to produce, foster exploration and encourage design as they are usually poor in visual appearance or level of details. On the negative they can be unrealistic and are not interactive. High-fidelity prototyping on the contrary requires additional skills and resource than low-fidelity techniques (e.g., sketching, paper mockups), it has the advantage of giving users a taste of what the final system would be and is more realistic in term of design and interactivity. Among the drawbacks, they tend to receive more superficial criticisms due to the fact that users believe it is the real system [10].

ICMI-MLMI'10, November 8-12, 2010, Beijing, China.

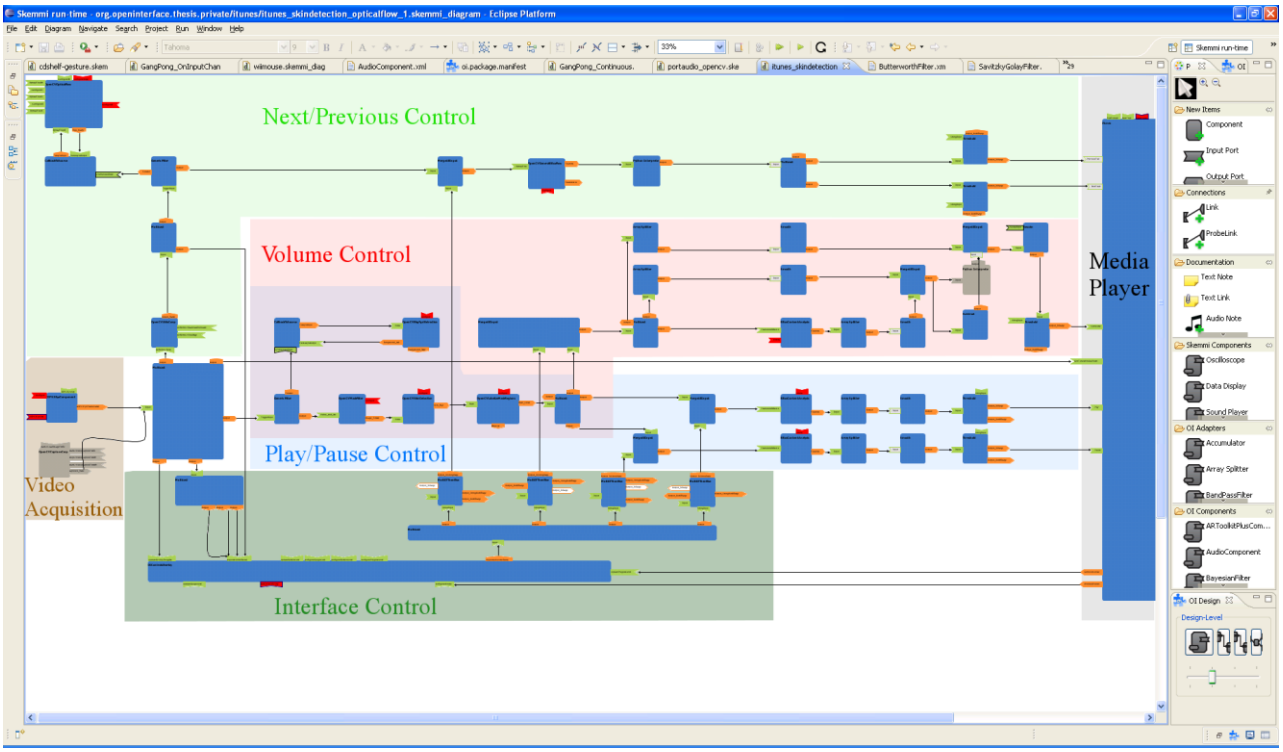

Figure 1: SKEMMI, OpenInterface Development Environment.

In the context of our work [6], we have chosen to handle rapid prototyping by adopting a hybrid (multi-fidelity) approach where we support informal and cheap generation of high-fidelity prototypes through reuse of third-parties software and tools. The aim is to benefit from non-formal techniques (sketching) for quickly producing high-fidelity prototypes. We therefore strive for overcoming some of the drawbacks of high-fidelity prototypes as described by [10].

2. A WORK BENCH SUPPORTING HIGH-FIDELITY INTERACTION TECHNIQUES EXPLORATION

Successful realization of advanced interactions can benefit from early prototyping and such iterative implementation of design requires the easy integration, combination, replacement, or upgrade of software components.

One of the features we aim at providing is the reuse, with minimal implementation tinkering, of the plethora of publicly existing tools. Solutions toward this goal must solve several problems including: programming languages disparity, computation models, software inter-dependencies, software behavior. The flexibility of our approach lies in that we do not constrain third-parties software to a specific architectural style or interface. Instead our approach provides abstractions and operators that allow adapting from the native software domain to our component-based model.

2.1 Runtime Architecture

The integration of heterogeneous software is done by (1) describing all components communication interfaces with the CIDL [6], and (2) by generating proxies to encapsulate original components implementation. The encapsulated components can then be reused in any OpenInterface platform application in a plug and play fashion by using SKEMMI, the configuration visual editor and development environment (see **Figure 1**). It provides dataflow controls, such as direct function calls and asynchronous calls.

2.2 Interaction Design Exploration

2.2.1 Dataflow Design

The basic technique we implemented for designing interaction is the well-known dataflow approach where an application designer draws components and connects output to input ports (see **Figure 1**). To ease this process, implicit data conversion is performed between compatible data types and documentation of component can be examined directly within the design canvas.

This composition technique requires technical knowledge of components interface and behavior while allowing for fine-grained tuning of interaction. To help hiding low levels implementation details from non-programmers, specific components are integrated so as to support techniques such as design-by-demonstration [8].

2.2.2 Design-by-example

Components integrated within the platform and SKEMMI editor allows users to perform design-by-demonstration techniques. Using this method a designer can visually inspect data, record desired behavior (e.g. complex gestures) as interaction patterns and further evaluate their effect at runtime. In that aim, oscilloscopes, various pattern matching algorithms are integrated and ready-to-use to support that informal design technique. Further algorithms and tools can be integrated to help in semi-automatically authoring advanced multimodal behavior.

3. CASE STUDY

The validation of such a system involves putting into practice the concepts described above and evaluating the proposed prototyping tools. In order to do so, we conducted participatory design exercises and realized several case studies. One of the proposed case study applications is detailed in this section. We first outline the test bed we used to conduct our experimentation and then we describe how the application has been implemented and depict the designs of different incremental pipelines. We last conclude with lessons learned and shortcomings retained.

3.1 Targeted Application

In order to exploit and validate the features offered by OpenInterface kernel and SKEMMI, the targeted application aims at managing several controls of standard media player software (iTunes here) by camera-based interactions. The selected user tasks were the volume level control, the play/pause activation, and the possibility to go to the next/previous track. Concretely, the user gestures are captured by a webcam located on a computer and spatially filtered thanks to an intermediate C# interface, displayed over the iTunes application (see **Figure 2**).

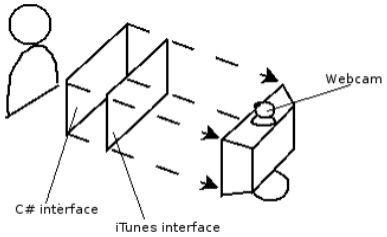

Figure 2 : Application overview.

The C# interface (see **Figure 3**) contains a set of control buttons (like push buttons) as well as the overlaid webcam video flow. Graphic representation of the control buttons are used to define several "regions of interest" (ROI), corresponding to rectangular areas of the current video frame and used as a location where interaction between the user and iTunes could take place.

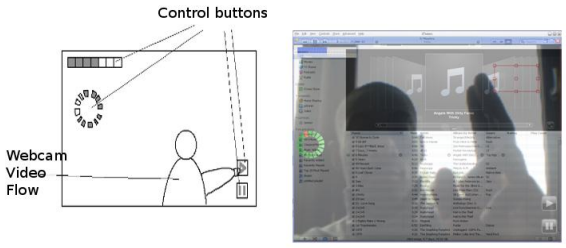

Figure 3: C# interface, displaying the video flow and the control buttons.

At this point of the application definition, the interactions managing the different user tasks are not defined. A set of video processing and a couple "ROI/control buttons" to explore are provided to developers to build these interactions.

3.2 Results

3.2.1 First implementation

Using SKEMMI, the developers have first designed a raw sketch of the application pipeline. This sketch (i.e., a low fidelity prototype) is used to define the global architecture of the application as well as the missing components to implement. Then, the pipeline development cycle is the following: the missing components are developed by the appropriate developer, integrated to the sketch and the pipeline is tested by developers to identify the remaining missing components. Lastly, following the feedback of each developer, the pipeline is improved by the modification of the interactions implementation or by the definition of new components to integrate.

Thus, the fidelity of the prototype increases at each cycle to reach a high fidelity level. At anytime, the development of a pipeline can be stopped, efficient interactions can be validated, or the pipeline can be redesigned from the beginning to explore new kind of interactions. Concerning the iTunes application described in 3.1, a high fidelity prototype (see **Figure 1**) has been obtained within one week after the first sketch. The next section details the various interactions developed and integrated for this application.

3.2.2 Interactions Definitions

For each control defined in 3.1, one or several potential interactions have been defined as described in the following sections. To implement these ones, the integration of new components has been necessary. Specific video processing functions have been mainly provided by the OpenCV library.

Play/Pause Control

In a first time, the control was considered as push buttons: by moving forward its hand, a user activated the play/pause action. A Minoru stereo-webcam was used to measure the mean of the depth within the play/pause ROI and compare it to a given threshold in order to change the play status of a track. However, the depth map update during a hand movement generated too much noise, which decreased the interaction fluidity.

Consequently, we opted for another interaction: the play/pause actions launched by a simple hand movement in the appropriate ROI. The movement detection was implemented as a succession of movement and skin detection on the current video frame. The possibilities offered by SKEMMI to easily combine video processes, to meld different programming languages and to reuse existing library have sped up the exploration and the definition of this interaction.

Volume Level Change

Several ideas were tested to design the volume interaction, such as using a gauge on the C# interface to control volume as it can be done with a potentiometer. Based on optical flow extraction, the user gesture consists in a hand rotation in the gauge ROI to change the volume (as he would turn a real potentiometer). This solution was lacking of precision and provided mediocre usability performances as it was tiring for the user.

The following interaction was then prototyped: to change the volume level, a simple and intuitive gesture is defined as moving the hand on a sound bar (C# progress bar) to the desired level. This interaction is also based on the succession of movement and skin detection. However, the x position of the hand inside the area gets here an importance, as it defines the volume level.

Track Change

The track change is insured by right and left hand movement in a predefined area (a customizable rectangle) of the C# interface.

The user can change the current track with the same simple movement than the one used to turn a book page. An optical flow analysis is performed in the area and direction extraction is used to insure the distinction between left and right movements, corresponding respectively to the next and previous controls.

3.2.3 Fine-tuning

The above system is permanently listening to interactions, and consequently generates a lot of noise, as it is not able to distinguish a user movement dedicated to interact with iTunes from any other movements. To overcome this, we added two clutching/declutching techniques of interaction control.

In the first system, user's second hand is used to validate the interactions. Moving this hand in the gauge ROI enables any interaction to be realized by the other hand. The hand movement is still detected by a succession of movement and skin detection.

The second system uses an OpenInterface component able to detect circles formed by fingers (by thumb and forefinger junction). The association of interactions and circle detection would then validate the current control. Detection of circle is obtained by combination of a component of background subtraction and a component of conditional dilatation.

These two pipelines also need to be fine-tuned to match the requirements of the targeted application. Fine-tuning could be divided in two categories: the adjustments of OpenInterface components parameters and the addition of basic operations (implemented through the Python interpreter component). The fine-tuning step of the iTunes application required one week too.

3.3 Conclusion

The goal of this work was to investigate and better support the realization of advanced interaction techniques. We have elaborated on several interactive design techniques (dataflow programming, design-by-example) and on a tool suite designed to leverage existing tools and methodologies. Concretely, we provided a unifying component-based model and a runtime infrastructure for allowing seamless integration and off-the-shelf components reuse. A collaborative design platform is also provided in order to support designers and programmers in the composition of heterogeneous interface, and in the exploration of various runtime configurations. The complete workbench is available on http://www.openinterface.org/platform.

3.3.1 Achieved Objectives

In [10], a set of problems introduced by high-fidelity prototypes were presented. The approach of our work is to provide tools and techniques that would allow overcoming some of these limitations. The goal is to support rapid high-fidelity prototyping as a cheap and creative activity.

Previous evaluations [6] have showed that the OpenInterface Workbench enables to explore and to effectively implement new interaction techniques. The use of the platform as a central tool enables cooperative engineering activities between different stakeholders (HCI, Image/Video processing ...). As an illustration, the experiment depicted in this paper has been realized in one week; several incremental pipelines have been designed and evaluated. Another week has been necessary to fine tune and evaluate the post-WIMP application designed.

In this case, libraries integrated in OpenInterface (OpenCV, iTunes, VLC, PS3Eye ...) and component simulation through Python scripting sped up the exploration of various interaction techniques.

3.3.2 Current Drawbacks and Future Works

In spite of coarse grained component design, visual complexity of the pipeline grows fast. This sometimes implies to rethink the whole design from scratch and adopt a stricter design of it, to be able to control correctly each interaction of the final prototype. Reducing the visual complexity through hierarchical composition or zoom able interface is only possible to an extent. State-based visual programming can help in defining command generation and in reducing the complexity of the pipelines. Moreover the possibility to visually segment the pipeline and to highlight its main part would facilitate its understanding and manipulation. Last lack concerns the debugging of designed application: as direct access to the code is not provided by SKEMMI, fine-grained debugging can sometime be difficult.

4. REFERENCES

[1] Beaudouin-Lafon, M. 2000. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *Proc .of SIGCHI conf. on Human factors in computing systems*. CHI'00. ACM, 446-453.

[2] Buxton, W. 1983. Lexical and pragmatic considerations of input structures. *SIGGRAPH Comp. Graph.* 17(1):31-37.

[3] Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., and Clow, J. 1997. QuickSet: multimodal interaction for distributed applications. In *Proc. of ACM MULTIMEDIA '97*. 31-40.

[4] Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. of UIST '06*.

[5] König, W. A., Rädle, R., and Reiterer, H. 2009. Squidy: a zoomable design environment for natural user interfaces. In *CHI EA'09*. 4561-4566.

[6] Lawson, J-Y. L., Al-Akkad, A., Vanderdonckt, J., and Macq, B. 2009. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proc. of EICS'09*. ACM, 245-254.

[7] Li, Y. and Landay, J. A. 2005. Informal prototyping of continuous graphical interactions by demonstration. In *Proc. of UIST '05*. ACM, 221-230.

[8] Merrill, D. and Paradiso, J. A. 2005. Personalization, Expressivity, and Learnability of an Implicit Mapping Strategy for Physical Interfaces. In *Proc. of CHI'05*. ACM, 2152-2161.

[9] Myers, B., Hudson, S. E., and Pausch, R. 2000. Past, present, and future of user interface software tools. In *Trans. on Computer-Human Interaction.* ACM, 7(1):3-28.

[10] Rettig, M., 1994. Prototyping for tiny fingers. *Communications of the ACM.* 37, 4, 21-27.

[11] Sinha, A. K. and Landay, J. A. 2003. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *Proc. of ICMI '03*. ACM, 117-124.