Supporting the design of multimodal interactions: a case study in a 3D sculpture application

Daniela G. Trevisan
Computing Institute - UFF
Niteroi, RJ, Brazil
daniela@ic.uff.br

Felipe Carvalho
Tecgraf Lab. - PUC-Rio
Rio de Janeiro, RJ, Brazil
kamel@tecgraf.puc-rio.br

Alberto Raposo
Department of Informatics - PUC-Rio
Rio de Janeiro, RJ, Brazil
abraposo@inf.puc-rio.br

Carla M. D. S. Freitas
Informatics Institute - UFRGS
Porto Alegre, RS, Bazil
carla@inf.ufrgs.br

Luciana P. Nedel
Informatics Institute - UFRGS
Porto Alegre, RS, Bazil
nedel@inf.ufrgs.br

Abstract

This paper explores the idea of developing a hybrid user interface that was conceived from splitting a previous single desktop application for 3D volume sculpture into three different interactive environments (Wimp, Augmented Reality and Head-Mounted Immersive Virtual Reality). To achieve such goal we illustrated how the OpenInterface platform can facilitate the development process in order to allow several modalities for user interaction within and along the environments. Components for user interaction such as speech recognition and 3D commands were inserted into the multimodal platform as well as components for handle fusion of modalities.

1. Introduction

In one of the earliest multimodal concept demonstrations, Bolt [2] had users sit in front of a projection of "Dataland". Using the "Put That There" interface, users could use speech and pointing on an armrest-mounted touchpad to create and move objects on a 2-D large-screen display. Semantic processing was based on the users spoken input, and the meaning of the deictic "there" was resolved by processing the x,y coordinate indicated by the cursor at the time "there" was uttered. Since Bolt's early prototype, considerable strides have been made in developing a wide variety of different types of multimodal systems [19] and [6].

Although several systems for multimodal interfaces are being built, their development is still a difficult task. The tools dedicated to multimodal interaction are currently few and limited or focused on a specific technical problem such as the fusion mechanism [7], and mutual disambiguation [21], or they are devoted for specific interaction paradigms. For example, the Toolkit GT2k of Georgia Tech, is designed to support the recognition of gestures [16] while the Phidgets tool [15] is focusing on tangible interfaces.

During the design phase of multimodal applications, the designer can specify the modality or modalities of interaction that can be applied to a given task as well as how these modalities will be combined or used. In this context we define a multimodal application as an application that includes multimodal data processing and/or offers multimodal input/output interaction to its users.

In order to support such functionalities OpenInterface can be viewed as a central tool for an iterative user-centered design process of multimodal interactive systems. It consists of a component-based architecture, which allows the integration of heterogeneous native interaction components as well as the connection between them to develop new multimodal applications.

In this paper we give an overview of this platform and demonstrate how it can be used to implement and support multimodal interaction in an hybrid user interface.

The remainder of the paper is structured as follows. In Section 2 we review relevant work in the area of multimodal software platforms and we highlight some advantages and features of using OpenInterface platform. Aspects involved in the design of multimodal systems are described in Section 3. An overview of this software platform is presented in Section 4, while the hybrid user interface for the 3D medical application as well as its interaction components are detailed in Section 5. Finally, Section 6 presents final considerations and discusses future directions of the work.

2 Related work on multimodal software platforms

There are several toolkits for investigating multimodal application design. A selection of well-known seminal platforms is listed here, and we highlight shortcomings that OpenInterface aims to overcome.

ICON [9] is a java input toolkit that allows interactive applications to achieve a high level of input adaptability. It natively supports several input devices. Devices can be added to the toolkit using JNI, the low-level Java Native Interface allowing integration with programs written in C.

ICARE [3] is a component-based platform for building multimodal applications. This solution defines a new component model based on Java Beans, and requires all components to be written in java. The platform is not easily extensible, produces non-reusable components, and also requires additional programming effort for integrating new devices or features.

CrossWeaver [26] is a user interface design tool for planning multimodal applications. It allows the designer to informally prototype multimodal user interfaces. This prototyping tool supports a limited number of input and output modalities and is not suitable for integrating additional software components.

Max/MSP [1] is a graphical environment for music, audio, and multimedia; Pure Data [22] is its open-source counterpart. Both provide a large number of design tools, are extensible, and have a large community of users including performers, composers, artists, teachers, and students. The extension mechanisms, however, require low level programming in C, and it is not possible to embed the platforms in external applications.

The goal of Exemplar [11] is to enable users to focus on design thinking (how the interaction should work) rather than algorithm tinkering (how the sensor signal processing works). Exemplar frames the design of sensor-based interactions as the activity of performing the actions that the sensor should recognize. This work provides an Eclipse based authoring environment which offers direct manipulation of live sensor data.

Most of the solutions listed above 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, and are designed for specific interaction paradigms. Moreover none provides a reusable framework for experimenting with various type of software component without the burden of complete re-implementation to match a chosen runtime technology. OpenInterface [2] differs from the above as it focuses on providing an interaction-independent, device and technology independent flexible solution for the fast prototyping of multimodal applications through the facilitation and reuse of existing software and technologies. OpenInterface also looks into the problem of how to support different stakeholders to achieve better collaboration while designing interactive multimodal systems in an ongoing complex design process. The platform offers two base tools, OpenInterface Kernel as a generic runtime platform for integrating heterogeneous code (e.g. device drivers, applications, algorithms, etc.) by means of non-intrusive techniques and with minimal programming effort, while achieving exploitable runtime performances (e.g. low latency, low memory overhead). SKEMMI is the provided design front-end. It supports a multi-level interaction design and allows composition and modification of running applications through techniques such as design-by-demonstration or direct manipulation. These are the reasons why we have chosen to develop our multimodal application using this platform.

[1] Max/MSP, http://www.cycling74.com

[2] www.openinterface.org

3 Fundamental concepts of multimodal systems

In this section we briefly introduce some relevant aspects involved in the design and development of multimodal systems.

Multimodal systems process two or more combined user input modes such as speech, pen, touch, manual gestures, gaze, and head and body movements in a coordinated manner with multimedia system output [20].

A large body of data reported in the literature allows affirming that multimodal interfaces reach higher levels of user preference when interacting with simulated or real computer systems. Users have a strong preference to interact multimodally, rather than unimodally, in a wide variety of different application domains, although this preference is most pronounced in spatial domains. For example, 95% to 100% of users preferred to interact multimodally when they were free to use either speech or pen input in a map-based spatial domain [21].

While recent literature has focused mainly on either speech and pen input or speech and lip movement interpretation, recognition of other modalities also is maturing and beginning to be integrated into new kinds of multimodal systems. In particular, there is growing interest in designing multimodal interfaces that incorporate vision-based technologies, such as interpretation of gaze, facial expressions, and manual gesturing [17, 28].

Although each modality can be used independently within a multimodal system, the availability of several modalities in a system naturally leads to the issue of their combined usage. An interaction modality can be defined as the composition of a language and a device, for instance,

blowing + microphone and *gesturing in the physical space + webcam*. Mechanisms for modalities fusion inform the manner these modalities have to be combined to provide useful information for the application.

The combined usage of multiple modalities opens a vastly augmented world of possibilities in multimodal user interface design. In this sense the CARE (Complementarity, Assignment, Redundancy, Equivalence) design properties [7] has been proposed to adrress such issue. Assignment property is represented by a single link between two components. Indeed a component A linked to a single component B implies that A is assigned to B. Redundancy property is represented by two or more modalities that can be used to perform the same user's task although the expression power of these modalities are not the same. On the other hand, the Equivalence property indicates that the modalities involved in the user interaction have the same expression power. It is important to understand "expression power" as the cognitive, cultural and learning level required by a specific language, while the Complementarity property is applied when the user's task requires two or more parameters as inputs. For instance in the MATIS (Multimodal Airline Travel Information Systems) system by combining synergy, the user can also combine speech and gesture as in the vocal command "show me the USAir flights from Boston to this city" along with the selection of "Denver" (destination city) with the mouse [19].

In our application development we consider the assignment, complementarity and redundancy properties as mechanisms to handle fusion of modalities (See Section 5.1).

4 Multimodal platform description

In this section we describe some technical aspects of the OpenInterface multimodal platform which were necessary to acquire for the development of our case study application.

In OpenInterface, each component interface is described and registered into the repository using the XML-based Component Interface Description Language (CIDL). The C++ kernel then automatically generates proxies to perform the actual integration. Using a graphical front-end or the kernel API (allows embedding the platform within the final application) users can configure components and compose complex execution pipelines for implementing multimodal applications.

4.1 Component description

An OpenInterface component consists of a computing unit, an algorithm, a bundled library, etc. Then, to be integrated into OpenInterface, a component has:

- an id: the id is an unique name for identifying a component among all others OpenInterface components. The structure of the id follow the hierarchical structure;
- a name, a simple human readable word identifying (non uniquely) the component;
- a language, the programming language used to implement the component;
- a container, the description of the delivery format. Basically, it aims at defining the way by which a component will be made available and those ways include jar file (for Java), shared object library (for C/C++), archive (C/C++), directory (Matlab, Java);
- a description of the I/O pins: this is where one describes the interfaces of a component. Basically it describes the logical units and their input and output functions;

An IO element encompasses the declaration of a component's interface. It contains declarations of a component different facets along with their respective sink or source pin. A facet is typically a logical unit inside a component and a pin is a function/method of a given unit.

A *sink pin* represents a way for a component to receive data. Conversely a source pin is the way for a component to send data. Both sink and source pins have mandatory properties:

- Id, each pin has a unique id inside a given facet;
- Interface, namely it is a description of a function's signature. Currently, there are some restrictions on the kind of functions that can be described; the types of the functions parameters are limited to primitives types such as (boolean, byte, char, 8 int, long, float, double, string) and up to 3-Dimensional array of primitive types.

The *Source pin* has an additional property: *Callback*. A *Callback* is the way for a component to call a function it doesn't know about at implementation time. For instance, a mouse driver component would expose a Callback for notifying about the current mouse position and buttons states. A Callback setter will then be called to register the external function. So instead of polling the mouse component for its state, the user will be notified by the registered function only when the state has changed.

For a visual representation of the component and its pins see Figure 2, botton view.

In order to assist the creation of an component description the Component Builder plugin provides the following features: the CIDL generation from source code, and the component packaging and deployment. The user opens the desired source file (which represents the interface of the

component) within the editor and can interactively modify the generated corresponding XML description. The user can directly edit code within the editor by either modifying non-compliant interface or removing undesired functions. Currently only C/C++ and Java source code parsing are supported.

4.2 Pipe description

In order to build a running application, OpenInterface uses the concept of Pipeline as an interconnection and configuration of components as illustrated in Figure 1. It allows control over the components life-cycle and execution site (remote, local), and provides low level (threshold, filter, etc.) and high level (multicast, synchronization, etc.) data flow control for building up a complex systems. A pipeline also supports dynamic reconfiguration of connections at runtime.

A pipeline thus defines and configures connections between components using the PDCL (Pipeline Description and Configuration Language see [13] for more details). It provides simple embedded dataflow controls, such as direct function calls and asynchronous calls, as well as simple mechanisms for extending the pipelines expressiveness in order to simplify intuitive interaction implementation.

The pipeline also allows isolating component in separate processes or distributing the execution of an assembly over a set of computers running an instance of OpenInterface Kernel. The distribution can either be performed seamlessly using the PDCL syntax or with connectors implementing well known protocol.

In order to provide support to this stage Figure 2 shows a typical interactive session in the design-time visual editor. While the left pane of the visual editor contains a hierarchical view of the project being built, the right pane contains in different tabs respectively the integrated components, the adapters, and the annotations elements. Using a drag and drop technique, users can initiate the conceptual assembly of the desired components (Figure 2, top) and further refine the pipeline to actually implement the desired behavior (Figure 2, middle).

4.3 Graphical Editor

This section describes the Eclipse-based end-user interface to the OpenInterface Kernel. SKEMMI editor provides dataflow editing features while also trying to put more emphasis on informal prototyping through machine learning components and on the collaboration of different actorsdesigners and programmers within the complex iterative design process [14].

To help users abstract from implementation details or dive into them at their best convenience, SKEMMI editor

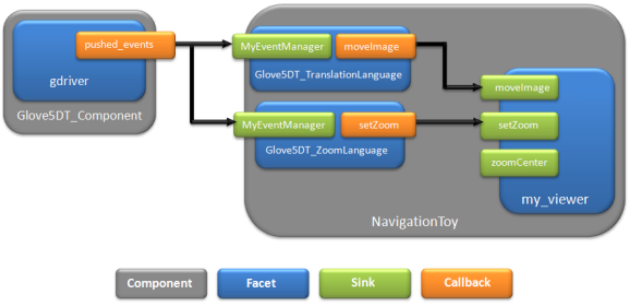

Figure 1. OpenInterface pipeline.

provides the ability to seamlessly navigate through three design-levels by the use of a three-level scale. The common representation of components is to depict them, their ports, and each linking among input and output ports. However, this information may be superfluous in a first overall design sketch. For example, when initially designing interactions, less emphasis is put on ports and types. Basically required components are elicited, logical links are drawn between them, notes and documentations are added to describe the overall concept. To support this "brainstorming" design phase the graphical editor provides a "workflow" level prospect that shows only components, conceptual links among them, and annotations as illustrated by Figure 2 top view. This level can be further refined to an actual implementation level (see Figure 2 center view). The same workflow is augmented with technical details such as ports, data types, and etc. Conceptual links can be instantiated to apply the mapping between design concept, notes, requirement and available components is made at this stage. The third level gets (visually) rid of all the interconnections and focuses on a single component design (see Figure 2 botton view). It also allows redefining a component interface at this stage if it does not suit the designer requirements.

5 Demonstration example

Before to start describing our system we introduce herein some previous works that have been done on the context of Hybrid User Interfaces. The concept of Hybrid User Interfaces is not new. Two previous works in this area deserve attention: the first work, from Rekimoto [24], has HUI following the ideas of ubiquitous computing using different devices. Some intuitive operations (pick-and-drop and hyperdragging) were created in order to transfer data among the devices. The second work is a similar project called EMMIE [5], which used different displays and devices supporting co-located and remote collaboration. The proposed system used augmented reality to place information in the space but also provided some private visualization of information. Another interesting work is the Mag-

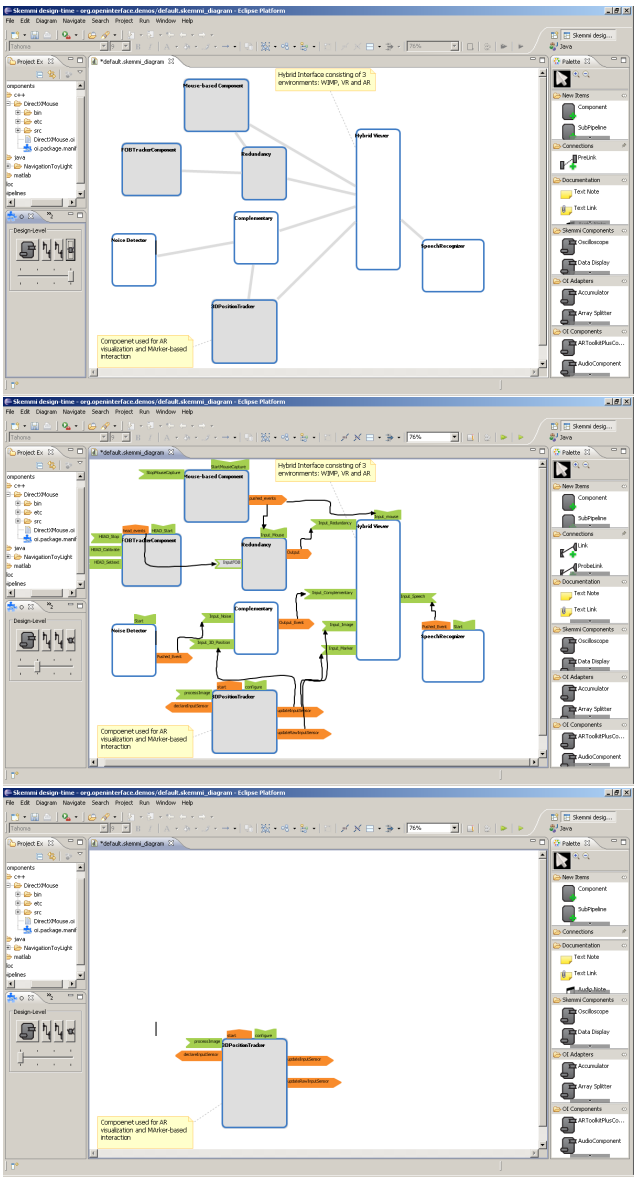

Figure 2. Three-level view of the 3D Hybrid sculpture within SKEMMI. Top: workflow; Middle: Dataflow; Bottom: component.

icMeeting [23], that created a collaborative environment using augmented reality based on tangible interfaces. By attaching 3D objects and WIMP interfaces to those tangibles interfaces, the system provided interaction with 3D and 2D data. Nakashima and colleagues [18] presented a system using WIMP interfaces and 3D objects. WIMP interfaces were projected on a table and 3D objects were displayed on an illusion hole. The system supported collaboration among some users. The Studierstube [25] also presented a collaborative system using augmented reality for co-located and remote users. Benko and colleagues [1] developed an application with cross-dimensional gestural interaction techniques, enabling interactions using 2D (via desktop) and 3D (via AR) visualizations, as well as the transference of objects between these kinds of visualization.

In this context, recent studies have demonstrated that in certain situations a mix of 3D and 2D interaction is preferred in relation to exclusive use of one or the other [8]. Keeping many interaction environments in the same workplace would be of great advantage for tasks composed by many different types of sub-tasks.

With this work we demonstrate how to transform an already implemented single desktop 3D applications into 3 complementaries interactive environments with support to multimodal user's interactions.

5.1 System overview

We start by describing our interaction scenario, which is providing a set of tools for volume sculpting. These tools were described by [12] and used three-dimensional geometries associated to the virtual hand metaphor [4]. The virtual hand is represented as different cursors with formats that reproduce in three dimensions the sculpting tools the user can apply to the 3D volume. The sculpting tools are 3D Rubber (which allows erasing parts of the volume), 3D Digger (that behaves as a carving tool) and 3D Clipper (represented as a cutting plane).

The system showed in Figure 3 was initially developed to be used in a desktop using 2D and 3D mouse-based interactions. In order to address the design issue of keeping dimensional congruence [8], which says the interaction technique used to perform a task must match the spatial demands of that task, we propose here to split user's interactions along a hybrid user interface.

In this work we are focusing only on the task provided by the 3D Digger tool as shown in Figure 4. This tool eliminates voxels in the interior of a virtual sphere, the region being defined by a 3D point P and a ray r specified by the user. The selection of a voxel for removal is done calculating the Euclidean 3D distance d of its center to P. The voxel is removed if $d < r$.

In order to provide user interaction within and through

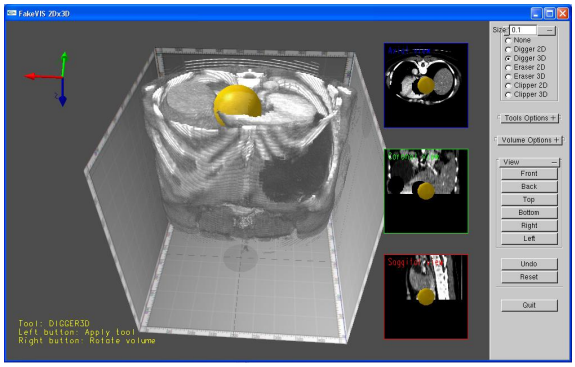

Figure 3. First version of the viewer with 2D and 3D visualizations of an interactive sculpting session where the volume of interest is a torso dataset.

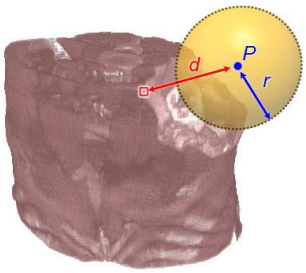

Figure 4. 3D Digger tool: eliminating voxels from the volume.

the three environments (WIMP, AR - augmented reality and HIVR – Head-Mounted Immersive VR), we have designed interactions for each environment as well as transitions between them.

Aiming to provided support for that we developed and integrated into the OpenInterface platform eight components: (1) the viewer itself, five interaction components - (2) image capture, (3) 3D tracking movements to support volume and tool manipulation, (4) noise detection, (5) user head tracking, (6) speech recognition and two components for fusion (7) complementarity fusion and (8) redundancy fusion. All these components, together with a mouse interaction component were integrated into the OpenInterface platform allowing the rapid prototyping of our hybrid environment with multimodal user interactions.

The initial system overview that was idealized is presented in Figure 5 and it could be implemented in a very similar way by the SKEMMI Graphical Editor as is illustrating Figure 2.

Details about each component are given in the next subsections.

5.2 Hybrid viewer

As mentioned before the idea of the hybrid viewer system consists on splitting the previous single desktop viewer (see Figure 3) to allow several modalities for user interaction within and through the three environments (WIMP, AR and HIVR Head-Mounted Immersive VR). In this way, the hybrid viewer component consists of the final user interface, where the combined or assigned interaction events from other components are continuously arriving.

Figure 5 shows an overview of the required interaction devices and components of the hybrid user interface while Figure 6 shows the interactions to perform transitions along the environments. All components involved in such interactions are described in the next sections.

When the application is launched by the OpenInterface execution pipeline all interaction components as well as the hybrid viewer are launched simultaneously. The hybrid viewer starts the WIMP user interface with mouse-based speech interaction ready for use. In order to dispatch the interaction event to the corresponding interactive environment we developed a dialog control. It is always checking the state of the speech port, and when a speech event arrives in the hybrid viewer the related environment is activated as well as all the user interaction modalities that are available for that specific environment. The user's interaction modalities that will be available in each environment were defined previously, during design time, by specifying the execution pipeline of application as follows.

5.3 Fusion components

Because the CARE properties (see Section 3) have been shown to be useful concepts for the design and evaluation of multimodal interaction, we have decided to reuse those concepts to make them explicit during the application development.

While Equivalence and Assignment express the availability and respective absence of choice between multiple modalities for performing a given task, Complementarity and Redundancy describe relationships between devices, languages or more generally between modalities for performing a given task.

These composition components describe the fusion mechanism of data provided by 2 to n interactions components. The criteria for triggering fusion are twofold: the complementarity/redundancy of data, and time. For the management of time, a temporal window is associated with each piece of data handled by a composition component. A temporal window defines the temporal proximity of two pieces of data $(\pm\Delta t)$ and is used to trigger fusion. Two pieces of data d1 and d2 are combined if their time-stamps, t1 respectively t2, are temporally close: Close (t1, t2) is sat-

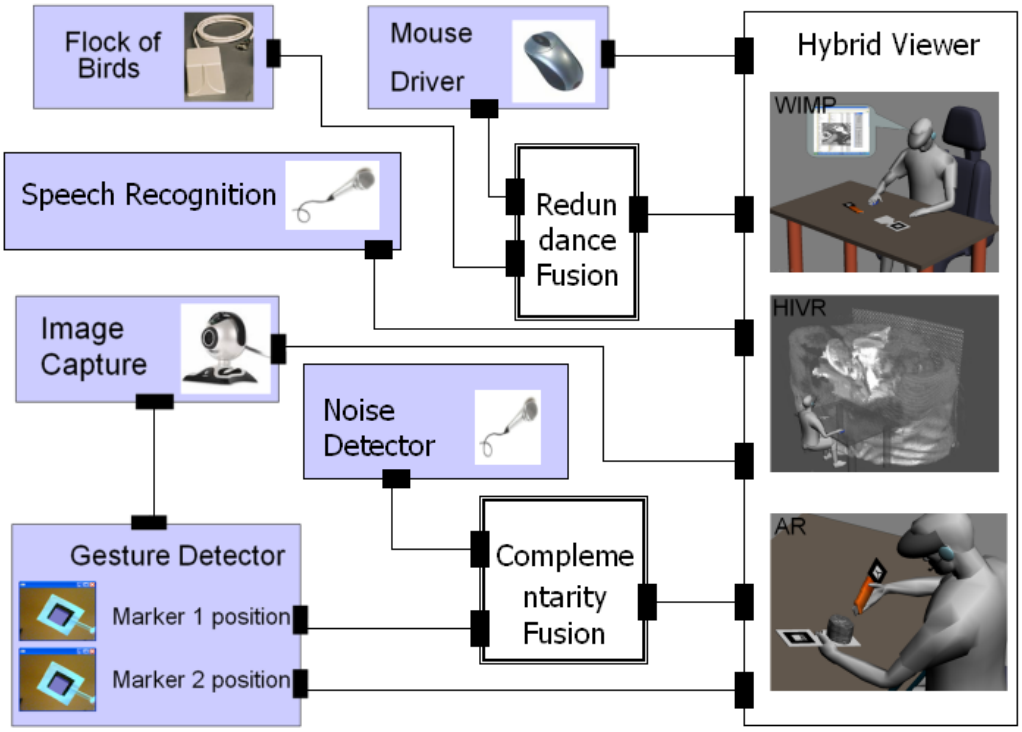

Figure 5. Initial conception of the application scenario overview.

isfied if: $t2 \in [t1 - \Delta t, t1 + \Delta t]$ For the management of data the own activate environment will be used to validate the data available for that user's interaction.

In the following subsections we illustrate how such mechanisms of fusion were implemented in our application.

5.4 Image capture

The Image Capture component used within OpenInterface uses the ARToolkit [3] library to capture the webcam image. After that, this component sends the captured image to the 3D position detector component to be processed, and to the Hybrid viewer component to be visualized. However, the image visualization is activated only when the speech event "AR" is detected by the dialog control in the hybrid viewer component.

5.5 3D position detector

3D Position Detector component uses the ARToolkit library to capture the 3D markers positions in the physical space. We are using a webcam while the user is moving two printed markers in the physical space to interact with the augmented viewer. One detected marker is used to manipulate the volume on the user hand and the other one is used to interact with the volume using the tracked tool (see Figure 7).

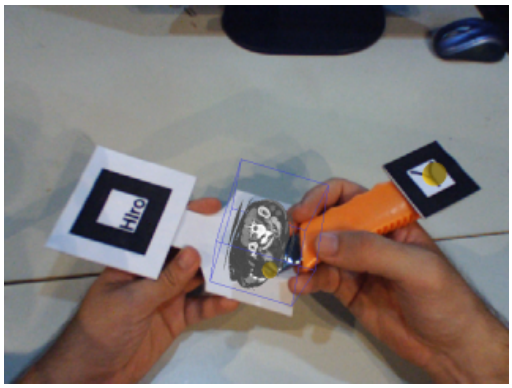

Figure 7. Bimanual and multimodal user interaction in the AR environment.

[3] http://sourceforge.net/projects/artoolkit

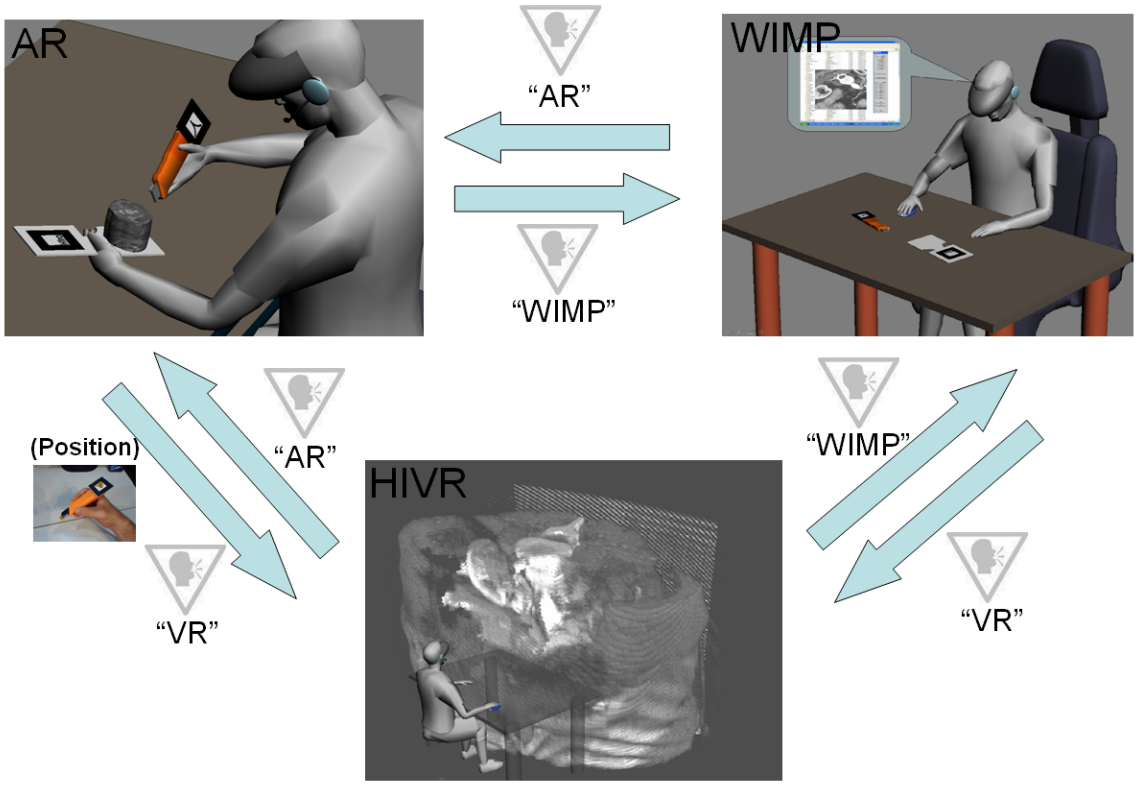

Figure 6. User's interaction along the three environments: WIMP, HIVR and AR.

5.6 Noise detector

The NoiseDetector component provides a single command that can be activated using a microphone. The technique is simple and does not require much processing. The component takes small sound buffers from the microphone and when it detects a sufficient noise, it triggers a callback function passing true (otherwise, it passes false). In this implementation, it passes true when more than 40% of the samples (absolute value) is greater than 0.8 (their maximum value is 1.0). This component was implemented in C++ using the Portable Real-Time Audio Library [4] to capture sound and then to process the signal.

The best way to generate such noise is by blowing directly into the microphone. Each time the noise is detected a single event is sent to the complementary component to be fusioned with the 3D location data sent by the 3D position detector component. At this point we are combining two complementaries signals (e.g. noise + 3D position) to perform a single interaction in the hybrid viewer which is erase voxels. In the case of signals detection failure or if the time stamp between the two detected events is not satisfying the Close function (previously described in Section 5.3) none event is sent to the Hybrid viewer component and then none action is performed.

5.7 User head tracking

Once the user is inside the HIVR environment, the mouse-based interactions become available for use, but, if they were not being used, the system will enable the head-tracking, and the virtual camera will follow the user head movement (e.g redundancy mechanism of fusion). We have implemented this technique using a Flock of Birds[5] motion tracker as a component into the OpenInterface platform. This component is always sending the user head orientation to the hybrid viewer, however it is activated only when the speech event "VR" is interpreted by the dialog control.

5.8 Speech recognition

To implement vocal commands to control transitions between the environments as illustrated in Figure 6 we have used Sphinx-4 [6]. Sphinx is a state-of-the-art speech recognition system written entirely in the Java programming lan-

[4] http://www.portaudio.com/

[5] http://www.ascension-tech.com/

[6] http://sourceforge.net/projects/cmusphinx

guage and it is now an interaction component available in the OpenInterface platform.

Sphinx uses speech recognizers based on statistical Hidden Markov Models and the Java Speech API Grammar Format (JSGF) to perform speech recognition using a BNF-style grammar. Our grammar to allow user transitions from one environment to another is as follows:

#JSGF V1.0;
grammar navigation;
public <navigation> =|VR|AR|Wimp;

Commands recognition results provided by this component seems to be satisfactory in terms of user interaction performance. In a very preliminary evaluation approximately 9 in 10 commands were successfully recognized after a short time spent for the user training model. These results were obtained using a head set microphone and none interference with the noise detector component was observed.

6 Final Considerations and Future Works

In this work we have demonstrated how systems with multimodal interactions can be quickly designed and implemented using the multimodal software platform OpenInterface. We have used the such multimodal platform to assembly several developed interaction components in a hybrid environment using as case study application a volume sculpting tool.

The main advantages in using such platform can be summarized as follows:

- Integrate new modalities, devices, and functional cores (i.e. components) into the platform. The software can be provided in any supported programming languages (C/C++, Java, Matlab and .NET; extensions can be easily added), and semi-automatic tools are provided to ease that process.
- Use the graphical interface to dynamically or statically combine components, and generate a running application. External applications can also control the pipeline by using the provided multi-language API.

It is worth noting that no significant events recognition delay was observed after integration of the several components into the multimodal platform and all components used in this approach are open source and can be freely downloaded from Internet. With that we hope to motivate the development of several multimodal applications in many fields of application.

As a result of this implementation some design issues that hybrid environments with support to interaction of transition should be able to address were identified. They are dimensional task demand [8] and task continuity [10] and [27]. Obviously, there is no way to say definitely what the best interaction technique or set of interaction technique best suits an entire category of interaction tasks. In this sense our purpose in future works is to attempt to identify tendencies and build a body of reusable knowledge leading toward better overall interface design. For that, we intend to develop a protocol for usability evaluation concerning to the multimodal user interaction, in order to verify these design issues while the user is performing hybrid tasks and compare the results with the previous one [12] obtained from user tests in a single desktop environment.

On the other hand, we have also observed that the concept of continuous interaction space became more evident in the context of hybrid user interfaces following the ideas of Ubiquitous computing, which argues that interaction environments should not reside only in the user desktop, but also in other devices and in the surrounding world. In this direction, in next versions of our system we intend to provide dynamic components for interaction according to different contexts of use (e.g. user, environment, task or platform).

7 Aknowledgments

This work was developed in the scope of the projects CNPq/INRIA EDGE and CNPq/FNRS MIDAS. Special thanks to Lionel Lawson from *Universite catolique de Louvain*, Belgium, for the OpenInterface support.

References

[1] H. Benko, E. Ishak, and S. Feiner. Cross-dimensional gestural interaction techniques for hybrid immersive environments. In *Virtual Reality 2005*, pages 209–216. IEEE, 2005.

[2] R. A. Bolt. Put-that-there: Voice and gesture at the graphics interface. *Computer Graphics*, 14(3):263–270, 1980.

[3] J. Bouchet and L. Nigay. Icare: A component-based approach for the design and development of multimodal interfaces. In *Proc. of 11th International Conference on Human-Computer Interaction HCI International*, Vienna, 2004. Lawrence Erlbaum Associates.

[4] D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. *3D User Interfaces - Theory and Practice*. Addison-Wesley, 2005.

[5] A. Butz, T. Hollerer, and et al. Enveloping users and computers in a collaborative 3d augmented reality. In *IWAR 99*, pages 35–44, 1999.

[6] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset:multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM International Multimedia Conference*, pages 31–40, New York, 1997. ACM.

[7] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four easy pieces for assessing the usability of multimodal interaction: The CARE properties. In *INTERACT*, pages 115–120, 1995.

[8] R. Darken and R. Durost. Mixed-dimension interaction in virtual environments. In *VRST'05*, pages 38–45. ACM, 2005.

[9] P. Dragicevic and J.-D. Fekete. Support for input adaptability in the icon toolkit. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 212–219, New York, NY, USA, 2004. ACM.

[10] E. Dubois, L. Nigay, and J. Troccaz. Assessing continuity and compatibility in augmented reality systems. *Journal of Universal Access in the Information Society*, 1(4):263–273, 2002.

[11] B. Hartmann, L. Abdulla, M. Mittal, and S. R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–154, New York, NY, USA, 2007. ACM.

[12] R. Huff, C. Dietrich, L. Nedel, C. Freitas, and S. Olabarriaga. Erasing, digging and clipping in volumetric datasets with one or two hands. In *ACM International Conference on Virtual Reality Continuum and Its Applications*, pages 271–278. ACM, 2006.

[13] J.-Y. Lawson. Openinterface description languages specification. *Technical report, available at http://www.openinterface.org/platform/documentation*, 2006.

[14] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 245–254, New York, NY, USA, 2009. ACM.

[15] S. MGreenberg and C. Fitchett. Easy development of physical interfaces throught physical widgets. In *Proc. of the UIST 2001 Annual ACM Symposium on User Interface Software and Technology*. ACM, 2001.

[16] S. MGreenberg and C. Fitchett. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *Proc. of the International conference on multimodal interfaces - ICMI'03.*, page 8592. ACM, 2003.

[17] C. Morimoto, D. Koons, A. Amir, M. Flickner, and S. Zhai. Keeping an eye for HCI. In *SIBGRAPI'99, XII Brazilian Symposium on Computer Graphics and Image Processing*, pages 171–176, 1999.

[18] K. Nakashima, T. Machida, K. Kiyokawa, and H. Takamura. A 2d-3d integrated environment for cooperative work. In *Symposium on Virtual Reality Software and Technology*, pages 16–22. ACM, 2005.

[19] L. Nigay and J. Coutaz. Problem space, fusion and parallelism in multimodal interfaces. In *Interface to Real and Virtual Worlds*, pages 67–76, 1993.

[20] S. L. Oviatt. *Multimodal Interfaces*. Lawrence Erlbaum, 2003.

[21] S. L. Oviatt, A. DeAngeli, and K. Kuhn. Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings of Conference on Human Factors in Computing Systems (CHI'97*, pages 415–422, New York, 1997. ACM Press.

[22] M. Puckette. Pure data: another integrated computer music environment. In *in Proceedings, International Computer Music Conference*, pages 37–41, 1996.

[23] H. Regenbrecht, M. Wagner, and G. Baratoff. Magicmeeting: A collaborative tangible augmented reality system. *Virtual Reality - Systems, Development and Applications*, 6(3):151–166, 2002.

[24] J. Rekimoto and M. Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *CHI 99*, pages 378–385. ACM, 1999.

[25] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, L. M. Encarnação, M. Gervautz, and W. Purgathofer. The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments*, 11:33–54, 2002.

[26] A. K. Sinha and J. A. Landay. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, pages 117–124, New York, NY, USA, 2003. ACM.

[27] D. G. Trevisan, J. Vanderdonckt, and B. Macq. Conceptualizing mixed spaces of interaction for designing continuous interaction. *Virtual Reality Journal*, 8(2):83–95, 2004.

[28] M. Turk and G. Robertson. Perceptual user interfaces. *Communications of the ACM*, 43(3):32–70, 2000.