OpenInterface: A Lightweight Open Source Platform for Rapid Prototyping of Multimodal Applications

Jean-Yves Lionel Lawson[1], Jean Vanderdonckt[2], Benoît Macq[1]

[1]Communications and Remote Sensing Laboratory (TELE)
[2]Belgian Lab. of Computer-Human Interaction (BCHI)
Université catholique de Louvain (UCL), Belgium
{jean-yves.lawson, jean.vanderdonckt, benoit.macq}@uclouvain.be

ABSTRACT

In this paper we present the *OpenInterface Kernel* as part of an open source platform for supporting the effective prototyping of multimodal interactive applications. Multimodal Interactive applications are based on the assembly of several components, namely, various and sometimes interchangeable modalities at the input, fusion-fission components and also several modalities at the output. Iterative design of such a system requires the easy integration, replacement or upgrade of components and the possibility to derive interaction properties from the component and basic ergonomic properties for the global system. We have designed a thin communication kernel able to manage this in an easy way by providing the research community a mean to solve a gap in the current support for multimodal applications implementation: *OpenInterface Kernel* is a practical light way to assemble various modalities with different implementation language, keeping a high level of performance of the assembled system. The effective connection of components requires their parsing in order to extract their communication features. Further parsing could be used to extract high level interaction properties. A running example illustrates the dynamic and extensible aspects of the platform. It runs on several operating systems and allows fast integration of interaction devices, signal processing tools, domain-independent data fusion, and dynamic runtime connection and re-configuration of interaction modalities.

ACM Classification: H5.2 [**Information interfaces and presentation**]: User Interfaces – *Prototyping; user-centered design*. 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, Verification.

Keywords: Prototyping, component-based architecture, reusable software component, multimodal interfaces, multimodal software architecture, integration techniques.

INTRODUCTION

Theoretical studies have highlighted the main issues at the development level of multimodal applications: content selection ("what to say"), modality allocation ("which modality to use to say it"), modality realization ("how to say it in that modality") and modality combination [22]. Criteria and requirements for the implementation of multimodal interactive systems are given in [34]; a precise way exists for describing, analyzing, and reasoning about multimodal systems prior to their implementation [29].

Currently, there is little research aimed at filling the gap between the design&specification stage and the implementation process of a functional system. There are existing solutions such as [9,13,28] for designing and implementing a multimodal system. They are limited, however, in the sense that they either present a small or hardly extensible number of input devices, they are platform and technology dependent or they do not provide a flexible prototyping environment for a large and heterogeneous number of research products, such as new devices prototype, new algorithms, etc. Prototyping is an important phase of multimodal application development process, as it allows designers to tackle the main issues presented above in an iterative fashion. A designer can *plug and play* with modalities, combine them, and quickly reuse the work done in a previous stage with little knowledge in low level programming. There is no software solution that presently provides this fast multimodal interaction prototyping feature.

In this paper we briefly present (Section 2) existing solutions for the design and implementation of multimodal applications by focusing on the dynamic and extensible aspects of the presented tools. Section 3 provides an overview of the *OpenInterface Platform,* an open source cross-platform software designed to support fast prototyping and implementation of interactive multimodal systems. Section 4 illustrates the platform's dynamic features by realizing a simple multimodal application taking advantage of our platform functionalities.

RELATED WORK

There are several toolkits for investigating the design of multimodal applications. A few are listed here, and their main shortcomings are highlighted.

ICON

ICON [13] is an input toolkit that allows interactive applications to achieve a high level of input adaptability. It is implemented in Java and natively supports few input devices. Devices can be added to the toolkit, but it re-

In Faure, D., Vanderdonckt, J., (Eds.), *Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010* (Berlin, 20 June 2010), Thales Research and Technology France, Paris, 2010. ISBN 978-2-9536757-0-2

quires a great amount of programming using JNI.

ICARE

ICARE [6] is also 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 in java. The platform is hard to extend, produces non-reusable components and also requires a lot of programming for integrating new devices or features.

CrossWeaver

CrossWeaver is a user interface designer's application for planning a multimodal application. 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.

QuickSet

QuickSet [9] mainly focuses on multimodality applied to the development of map-based systems. It uses speech and pen-based gesture as interaction modalities. The extension mechanism uses the Open Agent Architecture [10].

Max/MSP and PureData

Max/MSP [23] is a graphical environment for music, audio, and multimedia; PureData [30] is its open source counterpart. Both software provide a great 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 use the platforms without the graphical environment.

Most of the solutions listed above requires commitment to a given technology or supports a limited number of interaction modalities like voice, pen, text, and mouse, and are designed for specific interaction paradigms.

Our approach is different in the sense that we aim at providing a flexible solution for fast prototyping of multimodal applications by the easy and extensive reuse of existing software and technologies. The rationale for OpenInterface Platform Kernel (Figure 1) is therefore to be a generic platform for integrating heterogeneous code (device drivers, applications, algorithms, etc.) in a single software in a non intrusive fashion and with minimal programming effort. To support the design, implementation and reasoning on multimodal applications it will provide tools for composing, testing and debugging generated prototypes.

OPENINTERFACE PLATFORM

Requirements

Cross-platform, Heterogeneity, Reusability, Extensibility, and Performance

A major requirement of the platform is the ability to enhance and enforce the reusability of work previously achieved while implementing/prototyping new multimodal systems.

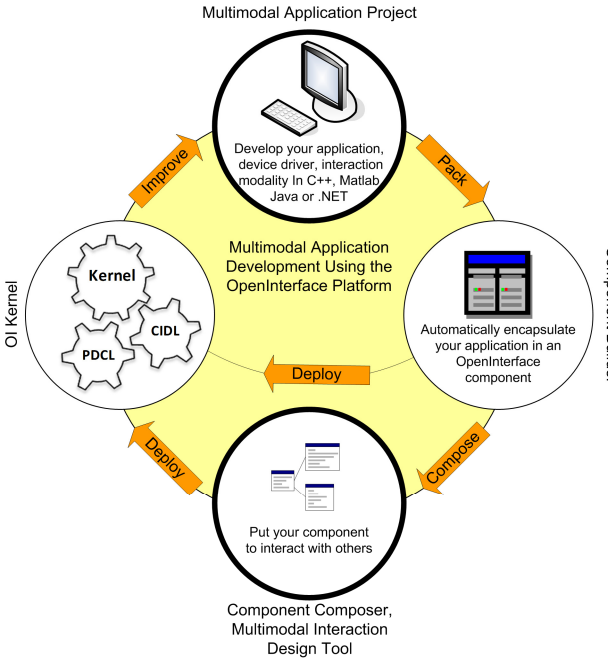

Figure 1. Multimodal Application Development Using the OpenInterface Platform.

Since the platform is intended for fast prototyping of new interaction models, it should also be easy to integrate and test immature signal processing research software like speech, gesture recognition – mostly implemented in high level prototyping languages such as Matlab, Python or scripting programming languages – as well as stable commercial devices provided with a low level API in C/C++ and graphical user interface software written in Java, Tcl/Tk, etc. To achieve good runtime performance of interactive systems, the platform must have good response time, despite the heterogeneous nature of the multimodal application being prototyped. One of the major differences between our approach and ICON is summarized by the heterogeneity requirement. It will be indeed a major advantage when it will come to rapidly integrate software and prototype interactions.

Multimodal Integration Support

Another important concept in multimodal application design and implementation is modality integration, or fusion. Two types of integration can be performed on modalities: feature fusion and semantic fusion. OpenInterface only implements domain independent fusion. Feature fusion is low level, domain-specific and performed between tightly coupled modalities (e.g. lips movement and speech), while semantic fusion is mostly high-level, preferably domain-dependent, and more related to the synchronization of time scale characteristics. Six basic types of cooperation between modalities are defined by [22]: Complementarity, Redundancy, Equivalence, Specialization, Concurrency and Transfer. A similar generic set of domain-independent combination of input modalities is

summarized in [12] where four roles are presented: Complementarity, Assignment, Redundancy, and Equivalence.

From a software point of view, all these generic roles can be expressed by a combination of simple data flow control operations. For instance, Complementarity can be implemented by temporal data synchronization, Assignment and Specialization by direct procedure call, Redundancy by connecting several modalities to the same functional module, etc. To facilitate the implementation of a functional multimodal system, the platform should embed a set of domain-independent fusion mechanisms but also offer a framework for adding new generic or tailored modality combination.

Plug and Play

Finally, our system should provide the ability to easily build a multimodal application by requiring as little programming effort as possible. Modalities, or software integrated within OpenInterface, should be easily assembled together in a plug and play fashion. This would allow for the creation of a network of components managing an advanced multimodal task, and provide the prototyping dimension to the system. Thus, the designers or programmers will then be able to quickly verify their model or specifications. The platform should therefore come with a significant set of application connectors, filters, debugging tools to ease the quick construction of an interactive multimodal system. An additional requirement is to not commit to a communication paradigm (event-based, procedure call, etc.), but rather provide tools capable of implementing the desired behavior. It will allow the platform to easily integrate with other software solutions.

Design

Heterogeneous Components

To achieve the reusability requirement, the platform adopts an extensible modular architecture where *components* are the base objects manipulated by the *OpenInterface Platform*. Our approach is very different from [6] in the sense that we do not define a new component model and we strive for minimal programming efforts when integrating new components. By not specifying an explicit model, it allows for flexibility by having the ability to implement a large set of models for multimodality. Additionally, at the implementation level, it enables communication with existing component models implementation like CORBA [11], EJB [15], and OSGi [28]. Components are unaware of the platform in which they are running; therefore, programmers use their preferred programming language and tools to build their piece of code, only declaring interfaces. Within our system, a *component* is only characterized by its interface and is defined as *a reusable and independent software unit with exported and imported Input/Output interfaces*. This definition is intentionally broad enough to encapsulate all models when implementing a multimodal system. Thus, a component must simply be software with the following mandatory attributes:

1. API (Application Programming Interface): there must be a way to communicate with the component services.
2. Installation/configuration: to facilitate the installation and configuration, the component should be packaged appropriately.
3. Documentation: the component must be well documented to enhance reusability.
4. No explicit dependencies with other components: a component must not make assumptions on the platform or other components existence or features. All required features must either be declared as imported, or packaged within the component.

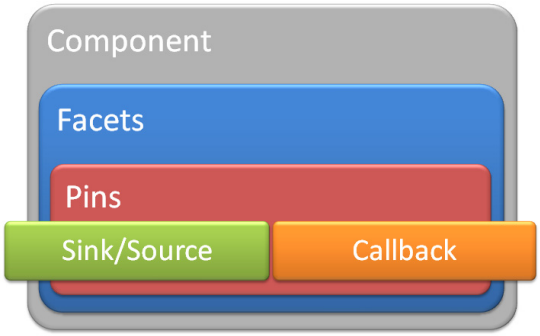

Figure 2. Component, OpenInterface view of any kind of external software.

OpenInterface provides tools to help producing such components. Figure 2 illustrates our view of a *component* as a bundled piece of software that provides a set of services/functionalities (*Facets*) which include input device drivers, signal-processing algorithms, fusion, fission, network communications, graphical interfaces, etc. These facets have well defined input/output interfaces called *Pins*. A pin can be a *Sink* (used to receive data), a *Source* (used to retrieve data from a component) or a *Callback* used to send data (and to import external functionalities into a component).

Pipelines

To build a running application, we introduce the concept of Pipeline as an interconnection and configuration of components as illustrated in Figure 3. It allows control over the components lifecycle and execution site (remote, local), and provides high level (threshold, filter, etc.) and low level (multicast, synchronization, etc.) data flow controls for building up a complex system.

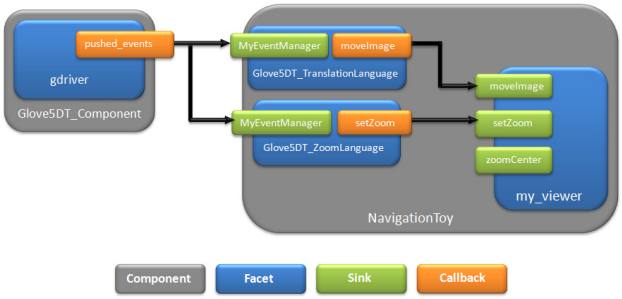

Figure 3. OpenInterface Pipeline.

Adapters/Connectors

Easily assembling components with a pipeline can only be done efficiently if the platform provides an extensible set of integrators which capture and mediate the communication between components. We use the concepts of Adapters/Connectors as entities that mediate interactions among components; that is, they establish the rules that govern component interaction and specify any auxiliary mechanism required [31]. Four categories fully describe the range of possible component interactions [24]: Communication, Coordination, Conversion and Facilitation.

In the platform, adapters/connectors are similar to components, with the difference being that their interface can either be generic or pre-defined. A generic interface can be connected to any components regardless of their interfaces, and Communication (TCP, RPC, etc.) and Coordination (Barrier, Synchronization, etc.) connectors fall into that category. Tailored connectors with well defined interfaces are called adapters in the platform, as they primarily serve the role of data conversion and facilitation.

Connectors are defined and designed so that they also include the definition of modality coordination, fusion. They are the basic tools for implementing semantic fusion (at the level defined by [22] and [12]) and communication paradigm (event-based, remote procedure call, pipe, etc.) within the platform.

Implementation

Heterogeneous Component Integration

Having components declare only their communication interface enforces the requirement for «independence». A component exports inputs/outputs to provide functionalities/services (display an image, device status, etc.) and imports inputs/outputs to request a feature provided by other components.

We have defined a description language called CIDL (Component Interface Description Language) that provides a way for components to declare their interface no matter the language in which they are written. The CIDL is semi-automatically generated from source code and is required by the OpenInterface Platform for manipulating a component. Figure 4 illustrates the syntax of the description language.

Once the CIDL description of the component has been made the platform actually generates code to encapsulate provided binaries into a well defined programming interface, Figure 5 illustrates that process. The encapsulated components can then be easily reused in any OpenInterface platform application in a plug and play fashion by using the pipeline description language presented in the following section.

Component Composition

A pipeline defines and configures connections between components and is described using the PDCL (Pipeline Description and Configuration Language [21]).

DirectXMouse.h

```
#ifndef _DirectInputMouse_H
#define _DirectInputMouse_H

#ifdef __cplusplus
extern "C"
{
#endif

#ifdef DLLEXPORT
		#define MYSPEC dllexport
#else
		#define MYSPEC dllimport
#endif
__declspec( MYSPEC ) void  startMouseCapture();
__declspec( MYSPEC ) void  endMouseCapture();
__declspec( MYSPEC ) void setCb(void(*cb)(int dx,int dy,int screen_x,int screen_y,int
bstate));

#ifdef __cplusplus
}
#endif

#endif
```

DirectXMouse.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Component PUBLIC "-//OpenInterface//DTD Component XML V0.1//EN"
"component.dtd">
<Component id="be.ac.ucl.tele.openinterface.component.win32.mouse.directx">
		<Name value="DirectXMouseComponent"/>
		<Language value="c++"/>
		<Container>
			<Name value="DirectXMouse"/>
			<Format value="so"/>
			<Location>DirectXMouse</Location>
		</Container>
		<IO>
		<Facet id="mdriver">
			<Bin>
			<CustomType type="cppclass" name=""
                                                def="DirectXMouse.h"/>
			</Bin>
				<Source id="pushed_events">
				</Source>
				<Sink id="StartMouseCapture">
				</Sink>
				<Sink id="StopMouseCapture">
				</Sink>
			</Facet>
		</IO>
</Component>
```

Figure 4. OpenInterface CIDL, Simplified Description (bottom) of a C/C++ Mouse Component (top).

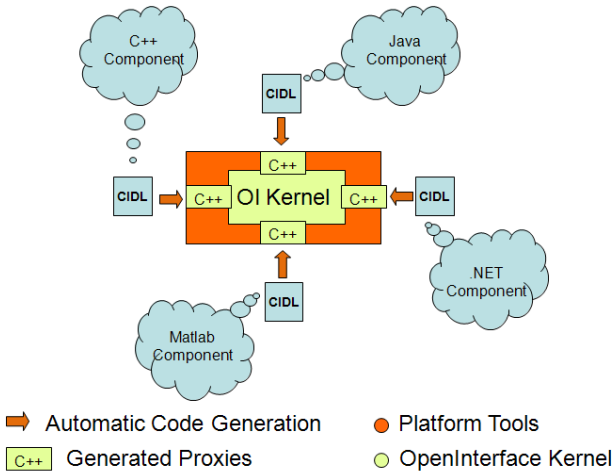

Figure 5. Heterogeneous Integration, Overview.

It provides simple embedded dataflow controls, such as direct function call and asynchronous call, but also ex-

poses a simple but powerful extension mechanism for extending the pipeline expressivity to simplify as much as possible intuitive interaction implementation. Currently, advanced flow control such as multicast (publisher/subscriber), Complementarity and Redundancy/Equivalence modalities fusion (temporal synchronization), data transformation like range filtering, rescaling, smoothing, etc. are distributed with the platform.

Overview

We have implemented the OpenInterface Kernel in C++ to achieve maximum performance, and also because most of the languages have C++ bindings. This allowed us to provide a portable platform capable of integrating heterogeneous software in a single application. An overview of how the platform works is illustrated by Figure 1, where each component is registered into the OpenInterface Platform using the Component Interface Description Language (CIDL). The platform then automatically generates proxies to perform the actual integration. Using an editor or the Kernel API, the user can edit the components properties and compose the execution pipeline (by connecting the components) of the multimodal application. This execution pipeline is either interpreted at once by the kernel or dynamically built by the application. More details on architecture and implementation can be found in [26]. According to the presented overview, the platform use can be decomposed into two main steps:

1. Integrate new modalities, devices, functional cores, i.e. components into the platform. The software can be provided in any of the supported programming languages (C/C++, Java, Matlab and .NET; more extensions can easily be added). This integration phase is well documented, and semi-automatic tools are provided to ease that process. It is usually performed by modality providers, i.e. programmers.
2. Use the pipeline interface to combine components and generate a running application. Currently there are two ways of using the pipeline:
   - Control the platform from the final application using the provided API and bindings. The API is in C++, but bindings for a large set of languages can be generated using the SWIG [4] wrapper. Java bindings are provided with the default installation as an example.
   - Use a development environment plugged to the platform. Currently there is one graphical editor, the OpenInterface Interaction Development Environment [16] (SKEMMI) which is a graphical tool, built on top of the platform, for the iterative design of a multimodal application. This tool is still in its alpha development phase, but is already available for download and evaluation.

This section presented the requirements, design, specification and implementataion of the OpenInterface Kernel. We have stated that having such a tool as an implementation base for various models of multimodality shortens the development phase and provides better multimodal applications, thanks to a high rate of implementation refinement using prototypes. In the next section we demonstrate the dynamic functionalities of the OpenInterface platform with the construction of a simple multimodal music player prototype by reusing existing open source components.

DESIGNING A PROTOTYPE

In this part we prototype an application to provide the ability of navigating in a multimodal way through an MP3 collection and select a song for playback. Figure 6 illustrates a simple open source CD case image collection viewer [25] which has no other function than displaying a set of images in an aesthetically pleasing layout.

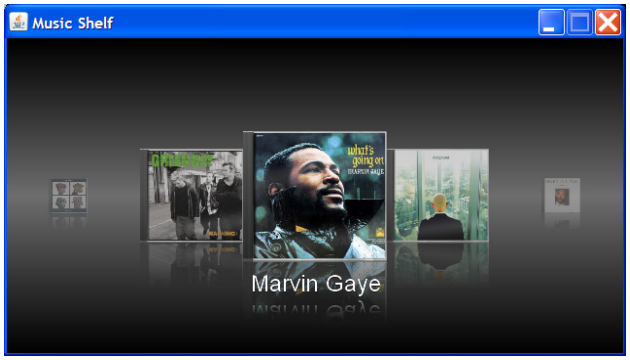

Figure 6. Java Music Shelf, a simple album art viewer.

For this application, we have integrated the following components inside the platform:

- A command-based MP3 player written in Java based on the JLayer library [18]
- A simple gauge level written in C# [8]

We also reused the following already integrated components:

- A Wii remote control (Wiimote) component which provides the device state (accelerometer, button, speaker and infrared points tracking) [35]

The integration process of existing code is fast and straight forward. It is thoroughly described in the platform online documentation.

First Iteration

In this first prototype, we build a pipeline as illustrated in Figure 7.

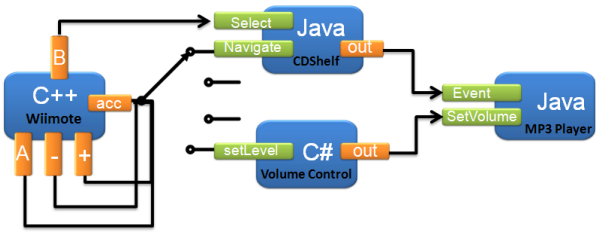

Figure 7. First Iteration of the Multimodal Music Player, Simple control with Wiimote.

The command-based Java mp3 player is the main application and has two interfaces for controlling the playback of a song (*Event*) and modifying the volume level (*SetVolume*). To provide an attractive graphical feedback to the user, the Music Shelf application (Figure 6) is used to display album art for each song, and a visual feedback for the volume level will be implemented by the meter level gauge (Figure 8). To control all of these visual components, we wanted to experiment with the Wii remote control. It is a powerful interaction device that comes with an integrated accelerometer, infra-red dots tracking capabilities, and several action buttons.

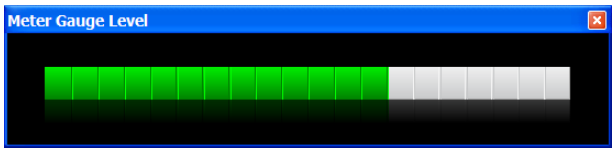

Figure 8. C# Meter Gauge Level for volume control visual feeback.

Task Model

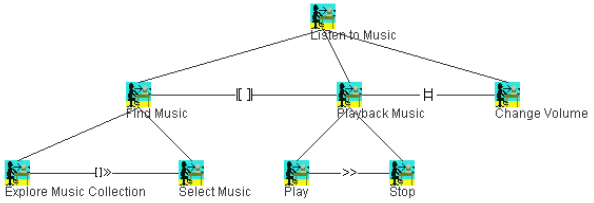

Figure 9. Multimodal Music Player Task Model.

Figure 9 illustrates the task model of our application where the user should be able to navigate through a collection of songs, select a song for playback, control the volume, and stop the playback.

Pipeline Implementation

Table 1 summarizes the *event-action* mapping we will perform to implement our task model.

Events	Actions
B Button pressed	*Start* a song playback
A Button pressed	*Connect* the Wiimote to the CD Shelf
+ Button pressed	*Connect* the Wiimote to the volume level control
- Button pressed	*Disconnect* the Wiimote
Acceleration (X)	*Navigate* through the songs collection or *Increase/Decrease* the volume

Table 1. Event-Action Mapping for the Multimodal MP3 Player

Applying that mapping at runtime, gives the user the option to disconnect the Wiimote from the application or connect it (Button A) to the CDShelf component to navigate through the image or connect it to the volume level control (Button +). The SKEMMI can be used to generate the runtime configuration of our application, as illustrated by Figure 10.

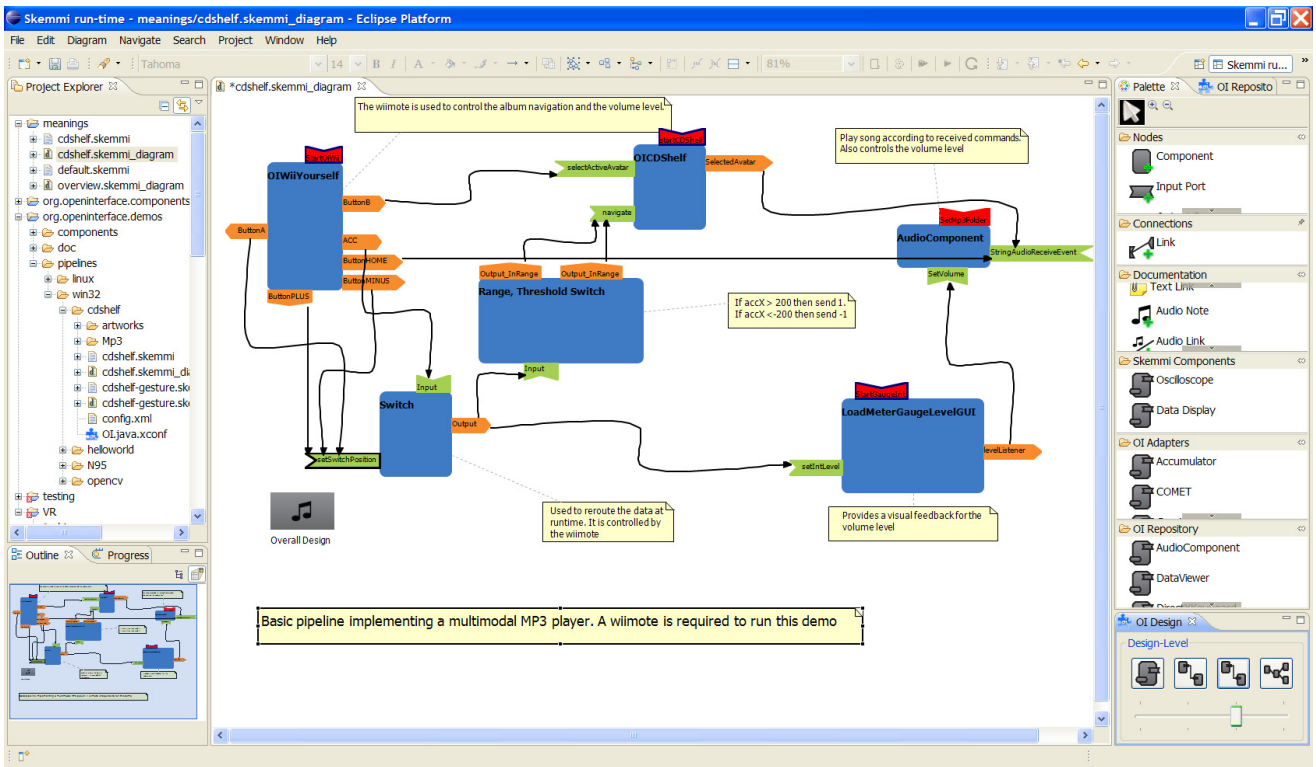

Figure 10. Design of the first iteration within SKEMMI.

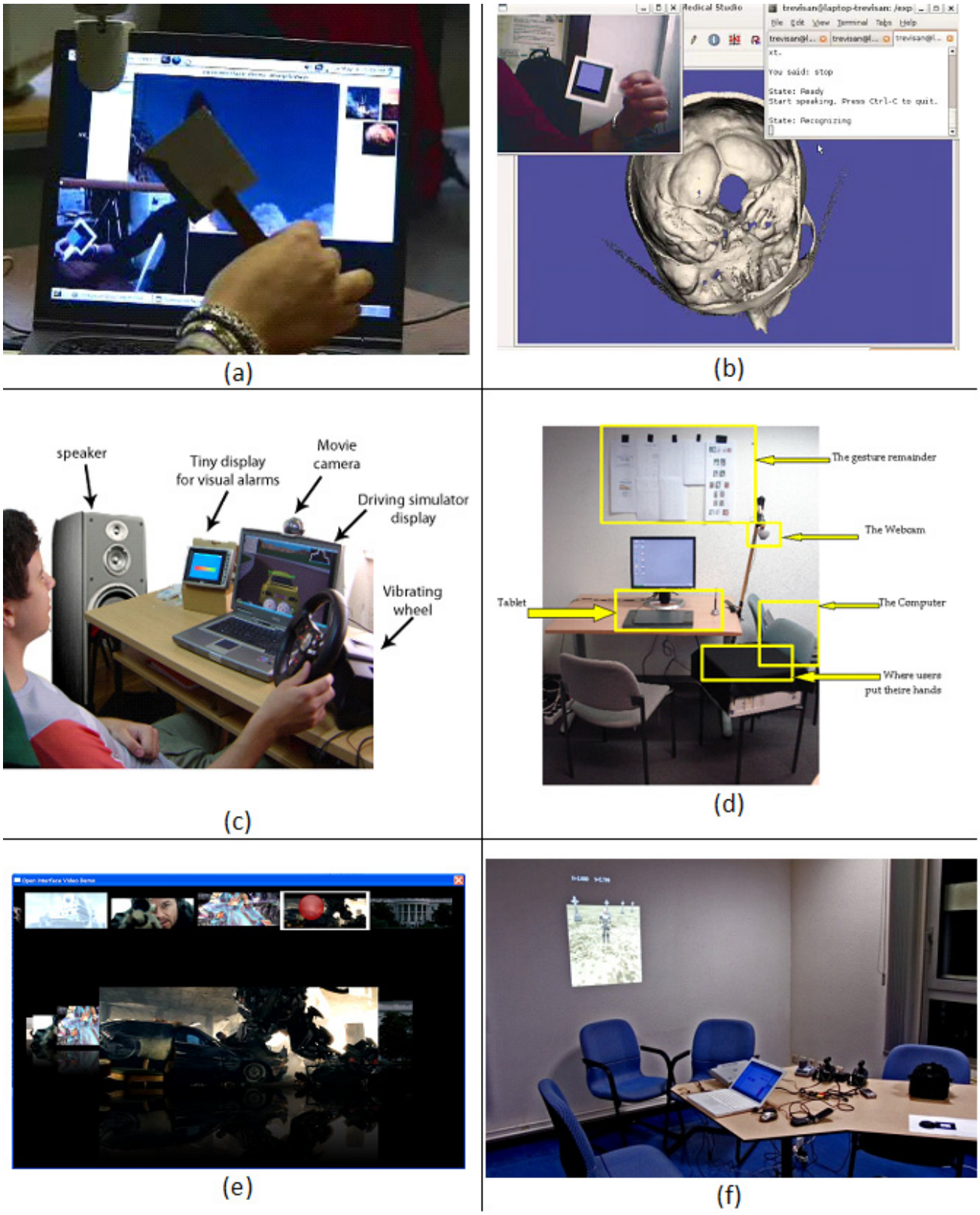

Figure 11. Selection of projects built with OpenInterface.

(a) First prototype, Image navigation; (b) Integration within MedicalStudio; (c) Hypovigilance state detection using a multimodal driver simulator; (d) InterpiXML integration; (f) 3D Game on Nokia N95 testbed.

The application can also be generated on the fly using the runtime API. This last solution is useful when the final prototype has to be a standalone application, therefore not relying on a graphical editor to compose connections.

In this first iteration, the accelerometer is too sensitive and triggers many events. To reduce the noise, we simply use a threshold connector between the Wiimote and the CDShelf component. The filter then only sends an event when its value (acceleration) is outside a predefined range; in this case we use [-200, 200]. This simple example has illustrated how easy it is to integrate modalities and combine them in a single application. The power of the OpenInterface Platform is demonstrated here in the sense that the Wiimote can easily be replaced by any combination of interaction devices and modalities.

In the next section, we describe some applications designed and implemented using the OpenInterface Platform, and give a short overview of the growing and easily extensible component database.

APPLICATIONS

The OpenInterface Platform has been used, and is being used, in several research projects for successfully designing, prototyping, and implementing multimodal applications. Some of them are described below:

Multimodal Driving Simulator

The OpenInterface platform was first used at eNTERFACE05 [14] a four-week workshop organized by the SIMILAR European Network of Excellence, aiming at establishing a tradition of collaborative, localized research and development work by gathering, in a single place, a group of senior project leaders, researchers, and (undergraduate) students, working together on a pre-specified list of challenges.The platform was still in its alpha state, being used as the backbone of a multimodal driver simulator [5], and working as the integration platform for combining the driver state based on video data (e.g., facial expression, head movement, eye tracking) and multimodal input. The setup of this application is illustrated by Figure 11 (c).

MedicalStudio

MedicalStudio [33] is a composable, open source and easily evolvable cross-platform framework that supports surgical planning and intra-operative guidance with augmented interactions. It is designed to integrate the whole computer aided surgery process to which both researchers and clinicians participate. OpenInterface has been used as an integration platform to study speech and marker detection interaction for 3D objects manipulation as illustrated by Figure 11 (b).

UsiXML Interpreter

UsiXML [35] (which stands for USer Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. InterpiXML is a runtime interpreter for UsiXML files and supported a limited number of modalities. The interpreter has been integrated as a component inside within OpenInterface so that it could use the platform as a runtime environment for controlling and combining a larger set of modalities. Figure 11 (d) illustrates a usability case study conducted using the described configuration.

Multimedia Information retrieval

The IRMA project [17] aims at designing and developing a modular, customized, efficient, protected and economically viable interface for multimodal searching and browsing in indexed audiovisual databases. The platform is being used for investigating multimodal interaction for navigating in a large set of multimedia data including annotated video, sound, and text. Figure 11 (e) represents a multimodal video viewer controlled with finger gesture and voice. The finger interaction modality has been implemented using infra red dot tracking.

OpenInterface Project

The OpenInterface Project [27] is an ongoing European IST Framework 6 Specific Targeted Research Project funded by the European Commission. It promises:

1. To ground the development of multimodal interactive systems in a scientific understanding of multimodal interaction. In our case, this will be achieved through reusable software components in the platform that are directly defined from theoretical results on multimodality.
2. To provide a tool for implementing a truly iterative user-centered design process.
3. To turn the results into industrial standards by way of the platform.

Several demonstrators have been implemented, including a multimodal map navigation and a mobile multimodal game. A typical setup for the latter is illustrated by Figure 11(e) where the game runs on a mobile phone (Nokia N95), while the different interaction modalities are running within the platform on a PC. This configuration has allowed Arcadia Design [3] designers to investigate different modalities to control their game: speech commands, 3D gesture commands with the Shake [36] and 3D gesture commands with ARToolkit [1] have been tested so far.

Component Database

Figure 12 synthesizes the current state of the platform component database and Figure 13 presents the database of connectors that can be used to compose components. The presented components are highly reusable and come from the various projects involved in OpenInterface. Tailored components, such as complete applications (games, functional cores, etc.) have not been listed because of their limited reusability in the field of application design.

	Connectors
Communication	•OSC •Bluetooth •TCP •Switch •Multicast
Operation	•Generic Filter •Smooth •Range Filter •Threshold •Rescale •Conditional •Delta
Fusion	•Complementarity •Redundancy/Equivalence

Figure 12. Overview of the OpenInterface Connector Database.

	Input	Output
Gestual	•Sketch Recognizer •Gesture Recognizer •Head Tracker (OpenCV) •Wiimote •Finger Tracking •Data Gloves (5DT, etc.) •TouchPad (Synaptics) •Face Tracker •Infra Red Tracking (OptiTrack) •Stylus •Hand Posture (HandVU) •Mouse, Keyboard •ARToolkit •...	•Wiimote •Shake
Visual	•Generic Pie Menu •Camera(Firewire, USB Webcam)	•VirtualMouse •Virtual Pointer
Vocal	•Speech Recognizer	

Figure 13. Example of components currently integrated and available from OpenInterface component database.

The database is already well furnished in input devices and a large number of combinations could be prototyped for studying new interactions. It also highlights the lack of output devices and interaction techniques; this provides us hints for future work, and directions to explore when designing innovative interaction modalities. This database is available from https://forge.openinterface.org. Specific licensing scheme may apply to each component, depending on the original binary used to perform the integration.

CONCLUSION AND FUTURE WORK

Going from a multimodal application design to a running implementation is not a simple task due to the lack of tools supporting this process. We have presented a runtime platform, the OpenInterface Kernel, which enables easy reuse of existing work to iteratively design and build a multimodal system with minimal programming effort. We have briefly illustrated the process of building a dynamic multimodal application – an MP3 player application controlled with the Wii remote control– by reusing existing open source software. The platform, as presented here, is available for download as an open source platform from https://forge.openinterface.org. Currently, there is one graphical front-end to the kernel, the SKEMMI – OpenInterface Interaction Development Environment – which focuses more on the designer view when developing multimodal application. However, the modular approach followed by the kernel allows it to be interfaced by any kind of front-end application which can output XML description. We have already successfully tested integration PureData and ICARE; in the near future we will investigate the integration of the platform within solutions like CrossWeaver, ICON and with existing open source interactive applications workflow editors such as *EyesWeb* [7] and *Clam Network Editor* [2]. Our focus is also directed toward enriching the platform with a comprehensive set of tools that will allow us to automatically generate multimodal interface prototype from high level description languages like UsiXML [35] or UIML [1] of interactions which is abstract from a particular implementation and allows to integrate basic rules of usability in the iterative design process.

ACKNOWLEDGMENTS

The work on the OpenInterface platform was partly funded by the European FP6 SIMILAR Network of Excellence (www.similar.cc) and is now mainly funded by the European FP6-35182 OpenInterface (www.oi-strep.com). J. Vanderdonckt also acknowledges the support of the Région Wallonne for ITEA2 Call 3 UsiXML project, under reference 2008026.

REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language. *Computer Networks 31*,11-16 (1999), 1695–1708.
2. Amatriain, X., Arumi, P., Garcia, D. 2006. CLAM: a framework for efficient and rapid development of cross-platform audio applications. In *Proc. of ACM Multimedia'2006* (Oct. 23-27, 2006). ACM Press, New York, pp. 951–954.
3. Arcadia Design, www.arcadiadesign.it
4. Beazley, D.M. SWIG: an easy to use tool for integrating scripting languages with C and C++. In *Proc. of the 4th Conf. on USENIX Tcl/Tk Workshop* (July 10 - 13, 1996, Monterey, CA). USENIX Assoc., Berkeley, 1996, 15.

5. Benoit, A., Bonnaud, L., Caplier, A., Damousis, I., Tzovaras, D., Jourde, E., Nigay, L., Serrano, M., Lawson, J-Y. Multimodal Signal Processing and Interaction for a Driving Simulation: Component-based Architecture. *J. on Multimodal User Interfaces 1*, 1 (2007), 49–58.
6. Bouchet, J., Nigay, L. ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces. In *Extended Proc. of CHI'04* (April 2004, Vienna). ACM, NY, 2004, pp. 1325–1328.
7. Camurri, A., Ricchetti, M., Trocca, R. EyesWeb - toward gesture and affect recognition in dance/music interactive systems. In *Proc. of IEEE Multimedia Systems* '99 (June 1999, Firenze). IEEE Comp. Press, Los Alamitos, 1999.
8. CodeProject, www.codeproject.com
9. Cohen, P.R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J. QuickSet: multimodal interaction for distributed applications. In *Proc. of ACM Multimedia'97*, 1997, pp. 31–40.
10. Cohen, P.R., Cheyer, A.J., Wang, M., Baeg, S.C. An Open Agent Architecture. In Proc. of *AAAI Spring Symposium* (March 1994). AAAI (1994), pp. 1–8.
11. Common Object Request Broker Architecture, www.corba.org
12. Coutaz, J. Nigay, L. Salber, D. Blandford, A., May, J. Young, R. Four Easy Pieces for Assessing the Usability of Multimodal in Interaction the CARE Properties. In *Proc. of IFIP Interact'95*. IOS Press (1995) pp. 115–120.
13. Dragicevic P. and Fekete J-D. Input Device Selection and Interaction Configuration with ICON. In A. Blanford, J. Vanderdonkt, P. Gray (eds.), *Joint proceedings of IHM 2001 and HCI 2001 (IHM-HCI'01)*. Springer Verlag, London, 2001, pp 543-558.
14. eNTERFACE Workshops, http://www.enterface.net/
15. Enterprise JavaBeans Technology. http://java.sun.com/products/ejb
16. Gray, P., Ramsay, A., Serrano, M. (2007). A Demonstration of the OpenInterface Interaction Development Environment. In *Adjunct Proc. of UIST'07*. ACM Press, New York, pp. 39-40.
17. IRMA: Interface de Recherche Multimodale en contenu Audiovisuel, http://www.irmaproject.net
18. JLayer, MP3 library for the Java Platform. http://www.javazoom.net/javalayer/javalayer.html
19. Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proc.of the 2nd Int. Workshop on Augmented Reality IWAR'99* (October 1999, San Francisco)
20. Katsurada, K., Sato, K., Adachi, A., Yamada, H. and Nitta, T. 2005. A Rapid Prototyping Tool for Constructing Web-based MMI Applications. In *Proc. of INTERSPEECH2005*, pp.1861-1864.
21. Lawson, J-Y., 2006. Openinterface description languages specification. Technical report, TELE-UCL, 2006. Available online: http://www.openinterface.org/platform/documentation
22. Martin, J. C., TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces. In *Intelligence and Multimodality in Multimedia Interfaces*, AAAI Press, 1997.
23. Max/MSP. Available on-line at: http://www.cycling74.com/products/maxmsp
24. Mehta, N. R., Medvidovic, N., and Phadke, S. 2000. Towards a taxonomy of software connectors. In *Proc. of the 22nd Int. Conf. on Software Engineering ICSE'2000* (Limerick, June 4-11, 2000). ACM Press, New York, pp. 178-187.
25. Music Shelf in Java2D, Available on-line at: http://www.jroller.com/gfx/entry/a_music_shelf_in_java2d
26. OpenInterface Association, www.openinterface.org
27. OpenInterface European project. IST Framework 6 STREP funded by the European Commission (FP6-35182). www.oiproject.org.
28. OSGi Alliance. www.osgi.org
29. Palanque, Ph. and Schyn, A. 2003. A Model-Based Approach for Engineering Multimodal Interactive Systems. In *Proc. of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'03* (Zurich, 2003) p. 543.
30. Puckette, M. 1996. Pure Data: another integrated computer music environment. In *Proc. the 2nd Intercollege Computer Music Concerts*, Tachikawa, pp. 37-41
31. Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
32. Sinha, A.K., Landay, J.A. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *Proc. of the 5th Int. Conf. on Multimodal Interfaces ICMI'2003* (Vancouver, November 5-7, 2003). ACM Press, New York, pp. 117-124.
33. Trevisan, Daniela G.; Nicolas, Vincent; Macq, Benoit; NEDEL, Luciana P. MedicalStudio: a medical component-based framework. In *Proc.of Workshop de Imagens Médicas WIM'2007* (Recife, 2007). Anais do Workshop de Imagens Médicas (em CD), 2007.
34. UIMS Tool Developers Workshop. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1), January 1992, pp32-37.
35. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.
36. Wiiuse, The wiimote C library. www.wiiuse.net
37. Williamson, J., Murray-Smith, R., and Hughes, S. 2007. Shoogle: Excitatory Multimodal Interaction on Mobile Devices. In *Proc CHI '07*. ACM Press, New York, pp. 121-124.