An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-The-Shelf Heterogeneous Components

Jean-Yves Lionel Lawson[1], Ahmad-Amr Al-Akkad[3], 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
{*firstname.lastname*}@uclouvain.be

[3]Fraunhofer FIT
Schloss Birlinghoven,53754 Sankt Augustin,Germany
{*firstname.lastname*}@fit.fraunhofer.de

ABSTRACT

In this paper we present an extensible software workbench for supporting the effective and dynamic prototyping of multimodal interactive systems. We hypothesize the construction of such applications to be 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. Successful realization of advanced interactions can benefit from early prototyping and the iterative implementation of design requires the easy integration, combination, replacement, or upgrade of components. We have designed and implemented a thin integration platform able to manage these key elements, and thus provide the research community a tool to bridge the gap of the current support for multimodal applications implementation. The platform is included within a workbench offering visual editors, non-intrusive tools, components and techniques to assemble various modalities provided in different implementation technologies, while keeping a high level of performance of the integrated system.

Categories and Subjects 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, Verification

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

1. INTRODUCTION

There is currently few ready-to-use software solutions aimed at filling the gap between the design & specification stage and the implementation process of a functional system. Seminal works present tools for the iterative design of multimodal systems [24][4][6], however, they either (1) present a small or hardly extensible number of input devices, (2) they are platform and technology dependent, or (3) they do not provide a flexible prototyping environment for a large and heterogeneous number of research products (such as new device prototypes, new algorithms, etc.). More recent works focus either on hardware prototyping [11] or are designed for targeted interaction techniques [18].

Prototyping is an important phase of the multimodal application development process, as it allows designers to address complex issues in an iterative fashion. A designer can *plug and play* with modalities, i.e. easily combining them, and reusing the work done in previous stages with little low-level programming knowledge.

Our goal is to provide an evolvable software solution implementing a device-independent, interaction technique and programming model-independent multimodal interaction prototyping feature.

In this paper, we first present existing solutions for the design and implementation of multimodal applications by focusing on the heterogeneous and extensible aspects of the presented tools. We then provide a short overview of the *OpenInterface Platform* [13], which is an open-source platform-independent software solution designed to support fast prototyping and implementation of interactive multimodal systems. The dedicated graphical front-end, *SKEMMI*, to the runtime platform is then motivated and presented. We continue giving an overview of applications successfully developed using OpenInterface and we conclude with discussion and future work.

2. RELATED WORK

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

ICON [6] 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.

EICS'09, July15–17, 2009, Pittsburgh, Pennsylvania, USA.

ICARE

ICARE [2] 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

CrossWeaver [24] 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 and Pure Data

Max/MSP [17] is a graphical environment for music, audio, and multimedia; Pure Data [21] 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.

Exemplar

The goal of Exemplar [10] 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.

Our approach 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. To leverage the features of our solution, we also look into the problem of how to support different stakeholders to achieve better collaboration while designing interactive multimodal systems in an ongoing complex design process.

We provide 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.

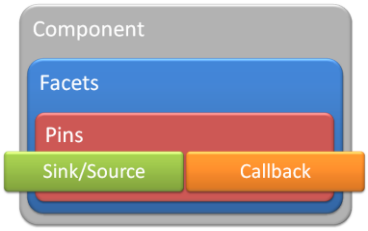

Figure 1: Component, OpenInterface view of any kind of external software.

3. RUNTIME PLATFORM

In this section we give an overview of the design and implementation details of the underlying runtime prototyping platform.

3.1. Heterogeneous Components

The platform adopts an extensible modular architecture in which *components* are the base objects manipulated by the OpenInterface Platform. Components can be implemented in virtually any language, we do not constrain to the use of a component model, and we strive for minimal programming efforts when integrating new components. Therefore, not specifying an explicit model provides additional flexibility, i.e. the ability to implement/support various models for interactive systems (e.g. MVC, PAC, ARCH, etc.). Components are unaware of the platform in which they are running; therefore, programmers can use any preferred programming language and external tools, while only declaring interfaces.

Within the 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 a large number of models when implementing a multimodal system. In other words, components are only specified by the following mandatory attributes:

1. API (Application Programming Interface): 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 dependencies with other components: a component must not make assumptions about the platform or features of other components. All required features must either be declared as imported, or packaged within the component.

Figure 1 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, and graphical interfaces. 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).

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

```
<!DOCTYPE Component PUBLIC "-//OpenInterface//DTD Component XML V0.1//EN"
"component.dtd">
<Component id="openinterface.component.win32.mouse.directx">
		<Name value="DirectXMouseComponent"/>
		<Language value="c++"/>
		<Container>
			<Name value="DirectXMouse"/>
			<Format value="so"/>
		</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 2: OpenInterface CIDL, Simplified Description (bottom) of a C/C++ Mouse Component (top).

We use a code generation technique to provide a non-intrusive integration platform with minimum programming overhead, while achieving good runtime performance. The integration of heterogeneous software is done by (1) describing all components communication interfaces in a language and platform-independent manner, and (2) by generating proxies to encapsulate original components implementation.

Having components declare only their communication interface enforces the requirement of 'independence'. A component exports inputs and outputs to provide functionalities and services (e.g. image display, device status), while it imports inputs/outputs to request features provided by other components. In order to declare interfaces, regardless of their implementation language, we define the XML-based CIDL description language (Component Interface Description Language - see [14] for more details) illustrated in **Figure 2**. CIDL code is semi-automatically generated from source code, and is required by the OpenInterface platform for manipulating components. The architecture of the system allows integrating other description languages such as WSDL [27]. Once the CIDL description of the component has been produced, the platform generates C++ code to encapsulate external binaries into a well defined programming interface, as illustrated in **Figure 3**. 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.

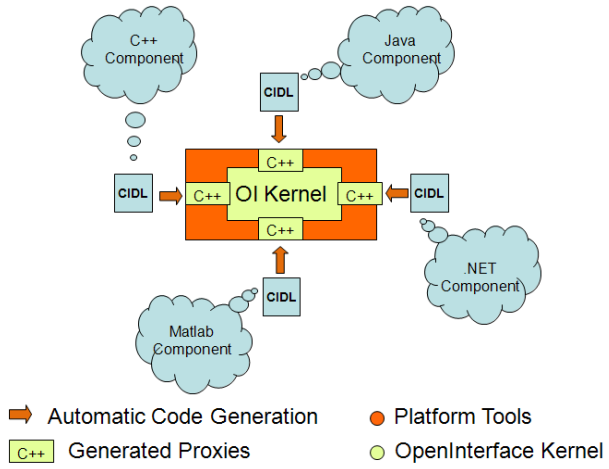

Figure 3: Overview of Heterogeneous Components Integration within OpenInterface

3.2. Pipelines

In order to build a running application, we introduce the concept of *Pipeline* as an *interconnection and configuration of components* as illustrated in **Figure 4**. 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 [14] 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 pipeline's expressiveness in order to simplify intuitive interaction implementation. Currently, advanced flow control, such as multicast (publisher/subscriber), Complementarity and Redundancy/Equivalence modality fusion (temporal synchronization), and data transformation such as range filtering, rescaling, smoothing, thresholding, etc. are distributed within the platform. 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.

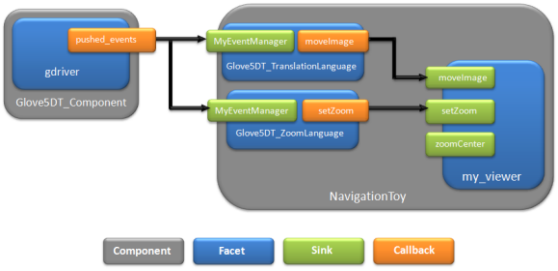

Figure 4: OpenInterface Pipeline

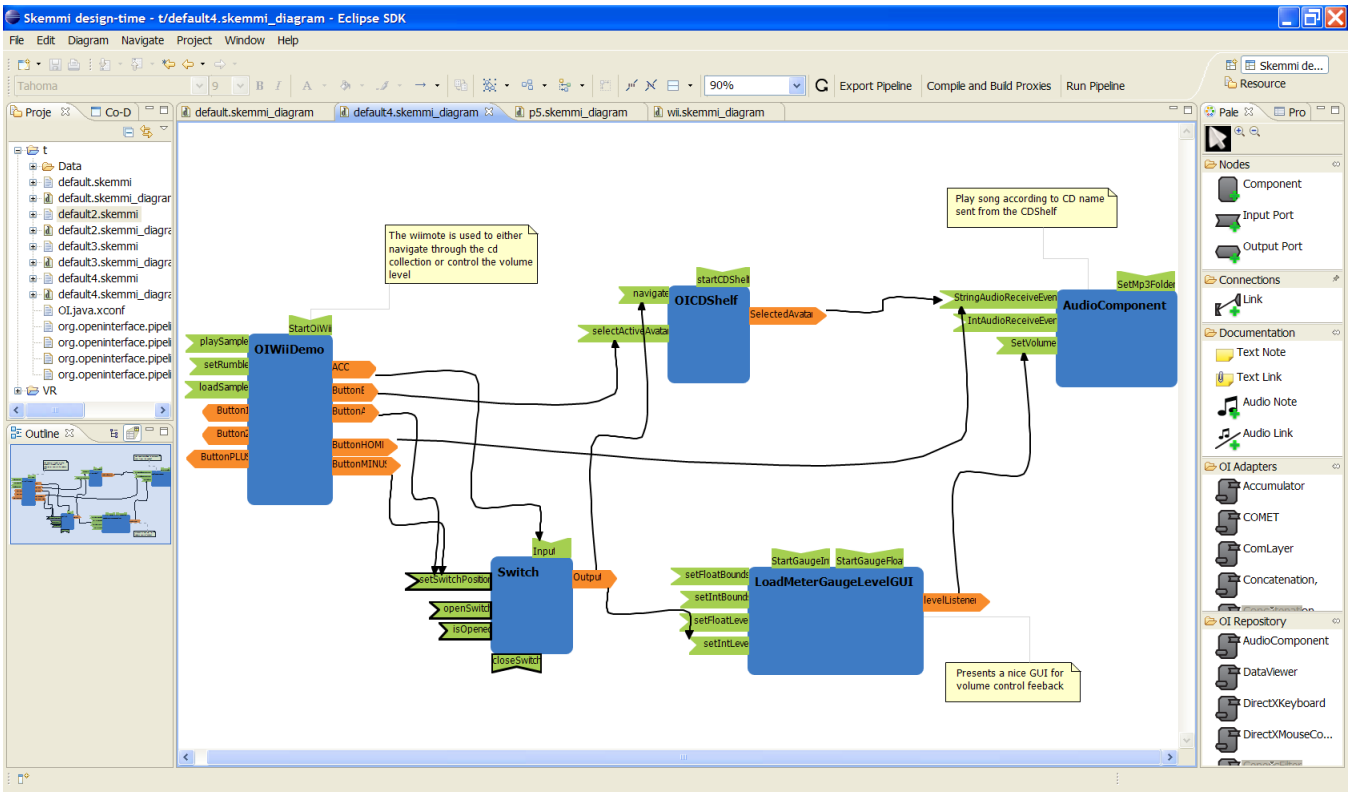

Figure 5: SKEMMI Eclipse Plugin, Simple Multimodal Music Player Design

3.3. 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 required auxiliary mechanism [23]. Four categories fully describe the range of possible component interactions [19]: 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 component regardless of its interfaces, communication (e.g. TCP, RPC) and coordination (e.g. Barrier, Synchronization) connectors fall into that category. Tailored connectors with well defined interfaces are called adapters, as they primarily serve the role of data conversion and facilitation.

They are the basic tools for implementing fusion (at the level defined by [16] and [5]) and communication paradigms (event-based, remote procedure call, pipe, etc.) within the platform. Although basic fusion is supported by the kernel, the mechanisms are not sufficient (without reengineering of involved components) to perform complex high level semantic fusion. In this aim, the platform is being extended with an advanced framework [26] for managing high level data fusion.

In this section, we have briefly presented the design, specification, and implementation of the OpenInterface Kernel. We believe that such a platform enables rapid experimentation with heterogeneous code and used in conjunction with high level graphical interface allows significant implementation refinement using multiple prototypes. In the next part we present the design platform, built on top of the kernel to leverage its runtime features in efficient multimodal application design and implementation.

4. DESIGN PLATFORM

During our preliminary experience with multimodal application development, we have recognized that these applications represent complex artifacts and their development requires the expertise of both interaction design and software engineering domain. In a first approach we evaluated OIDE [8], a design tool presented as a development environment for multimodal interaction built on top of OpenInterface runtime platform. We have encountered a couple of shortcomings while using OIDE and more generally during the whole application design phase:

Limited Focus

The first main difficulty is that component developers rather focus on designing their individual components than the application as a whole. Therefore each implemented application version exposes an inflexible design, i.e. hardly flexible assemblies of components.

Inflexible Design

Components are designed rather statically. While developers implement components they mainly think that they need to provide components which fit into one specific assembly. Hence,

it is hard to exchange, or to extend compositions, as interfaces are being designed too close.

Not enough Influence of Non-Developers

As a matter of the second difficulty, interaction designers are restricted to explore user experience of multimodal interaction, as strongly fixed compositions put constraints on experimenting more intensively alternative application set-ups or multimodal interaction styles respectively. Furthermore, interaction designers and the rest of non-developers are insufficiently equipped for contributing to the software design to make the need for more alternatives, dynamic setup clear to component developers.

Loosing Design Solutions

Last but not least, tracking things is hard. Conceptual solutions are usually kept in textual and graphical documents. The design process is an ongoing stream where all team members collaborate to improve the design. Despite the fact that during the design process a lot of issues are discussed and solutions proposed, it happens the latter are disregarded when translating the design concept into software.

Due to these difficulties, in most cases, the interaction design concept is not properly being implemented, and as a consequence interaction designers cannot easily produce enhanced user experience of multimodal interaction. To narrow down this gap we argue for an approach that balances the needs of both disciplines with the challenge of improving the way an 'interaction design'-concept is actually translated into a set of software components.

4.1. Motivation for SKEMMI

The limitations highlighted above have motivated us in providing an all-in-one prototyping workbench for multimodal applications development. Our approach differs on the following main features not addressed by OIDE:

Support for components development

Developing innovative interactions components calls for the need to be able to implement, refine and package those components. We hypothesize that having an integrated tool that allow to (informally but effectively) specify, develop, and deploy components is a feature that can speed up the learning and development process.

Support of multi-level design

Supporting basic cooperation of users with different backgrounds taking part in the realization of a multimodal application can also be beneficial for the quality of delivered application.

Support for reusability

Plugging off-the-shelf component is not a trivial task because most of the time their interface, behavior and data need to be adapted. When evaluating OIDE, we have encountered the problem of interface incompatibility, thus limiting reusability of components. SKEMMI will leverage the data conversion, synchronization, and communication patterns features provided by the OpenInterface Kernel and also provides syntactic sugar allowing for better abstraction from technical configuration when connecting components.

Support for documentation

Although composing a technical assembly is an important feature, we have found it mandatory to also be able to easily document, export, and distribute composed applications.

Runtime and Debug

SKEMMI will provide the ability to run a designed pipeline and graphically reconfigure the system at runtime. Basic debugging will also be natively supported through visual tools embedded within the GUI.

In the following section we present the implementation of our approach that tries to consider constraints of both stakeholders— interaction designers and software engineers— in a cooperative and designer-centered approach so as to minimize the shortcomings highlighted here above.

4.2. Cooperative Design

We looked into the problem of supporting different stakeholders to achieve better collaboration while designing interactive multimodal systems in an ongoing, complex design process, thus resulting in better application design and implementation. Our aim is not to provide a complex and fully-featured collaborative environment but rather to provide a simple common working base that enables various actors (e.g., various stakeholders having different viewpoints or backgrounds). In our simplistic approach we hypothesize the use of annotations (text and audio), changes tracking, multi-layered view of a single design process and support for informal interactions prototyping to be sufficient to meet our needs, while also assuring an infrastructure for future extension.

Multi-level Design

To help users abstract from implementation details or dive into them at their best convenience, we provide 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 we provide a "workflow" level prospect that shows only components, conceptual links among them, and annotations as illustrated by the top screenshot of **Figure 6**.This level can be further refined to an actual implementation level (center screenshot of **Figure 6**). 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. It also allows redefining a component interface at this stage if it does not suit the designer requirements as explained in the next section.

Component Tailoring

Because of the explorative approach while designing interactions pipelines from off-the-shelf components developed by third-parties, it is quite common that a component interface does not match the designer needs. The editing canvas provides the ability to redesign (change ports attributes and parameters, delete or create port) component interfaces. It's a collaborative activity in the sense that the redesign is done in an informal way by non-programmers and further directly translated by technical users in programming items (i.e. code skeletons and CIDL descriptors) using the code generator.

Changes Tracking

To better support multiple users working on the same design, we define a profile for each actors with an assigned color code. By doing so, changes applied by each participants can be tracked and highlighted. Basic history of designed pipeline will be supported through the use of the Eclipse version control plugin features.

4.3. Interaction Design

Dataflow design

The basic technique for designing interaction is the well-known dataflow approach where an application designer will drop boxes and connect output to input ports.

This task can be cumbersome if exchanged data types are incompatible, external behavior of components is non-documented, and if there is insufficient documentation about components interface. To ease this process, implicit data conversion is performed between compatible data types and documentation of component can be examined directly from 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 [20].

Design-by-example

Components integrated within the platform and editor allow performing a design-by-demonstration technique. 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. Signals inspector such as oscilloscope, image processing library such as OpenCV, pattern matching algorithm (Dynamic Time Warping [9], five degrees-of-freedom gesture recognition) are integrated and ready-to-use to support that informal design technique. Moreover, taking advantage of the underlying platform modularity and heterogeneous components support, further algorithms and tools can be integrated to help in semi-automatically authoring advanced multimodal behavior.

This section has presented an 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 actors –designers and programmers – within the complex iterative design process. The next section illustrates the different stages involved in the use of the proposed multimodal workbench.

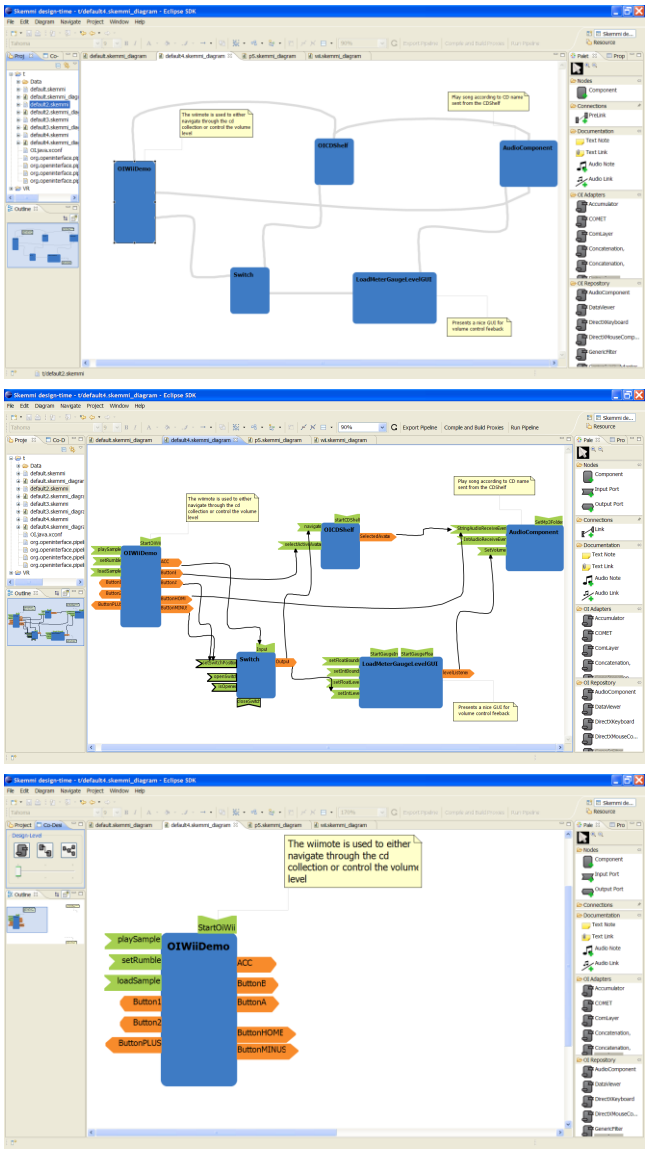

Figure 6: Three-level view of wiimote controlled multimodal music player within SKEMMI. Top: workflow; Middle: Dataflow; Bottom component

5. MULTIMODAL WORKBENCH USAGE SCENARIO

Overview

The OpenInterface Kernel is implemented in C++ to optimize performance, as well as to benefit from existing C++ bindings available for other languages. This allows us to provide a portable platform capable of integrating heterogeneous software in a single application. An overview of how each tools of the workbench interact with others is illustrated by **Figure 7**, where each component is registered into the OpenInterface platform using the CIDL. The platform then automatically generates proxies to perform the actual integration. Using SKEMMI or the kernel API, users can edit the components properties and

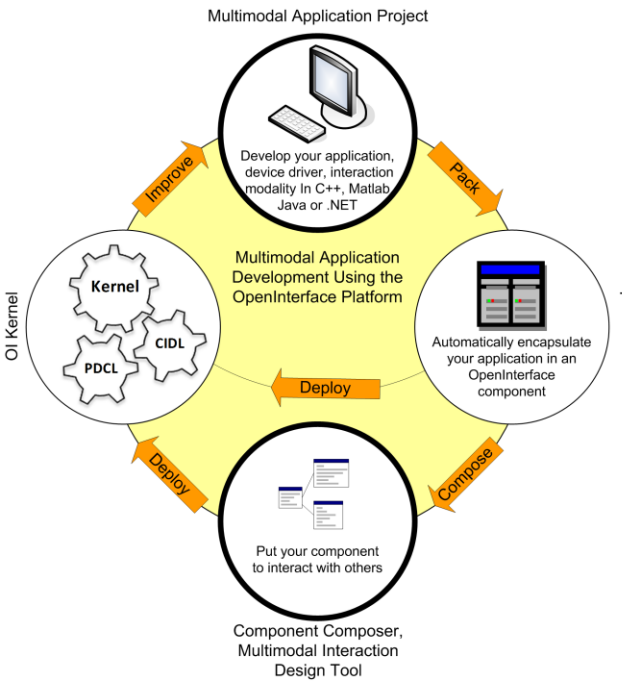

Figure 7: Multimodal Application Development Using the OpenInterface Platform Workbench

compose an execution pipeline of a multimodal application. This execution pipeline is either interpreted at once by the kernel, or dynamically built by the application.

The use of the OpenInterface platform requires the following two steps:

1. 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.
2. 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.

5.1. Integrate Component

The very first step involved in using our software workbench is integrating components within the platform. This process is straightforward, documented, and a dedicated Eclipse plugin is provided to assist the user in achieving this task. One could either integrate existing code or generate new components skeleton.

Integrating Existing code

The Component Builder is the Eclipse platform plugin which provides the following features:

- CIDL generation from source code.
- 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. **Figure 8** illustrates the Component Builder view within Eclipse: Area A displays the projects, area B shows the hierarchical structure of the generated XML and area C presents both the source code and the corresponding CIDL.

Our tests with users, having little to no experience with the platform, have shown that after reading the provided tutorials, an average of one hour is required to manually integrate and test a new component.

Designing New Components

Alternatively, a new component interface can be designed from within the design editor. In that case the user creates a new component and specifies the input/output ports and parameters of the designed component. Documentation can also be attached to the design. The corresponding CIDL descriptors are then generated along with code skeletons for further implementation by programmers.

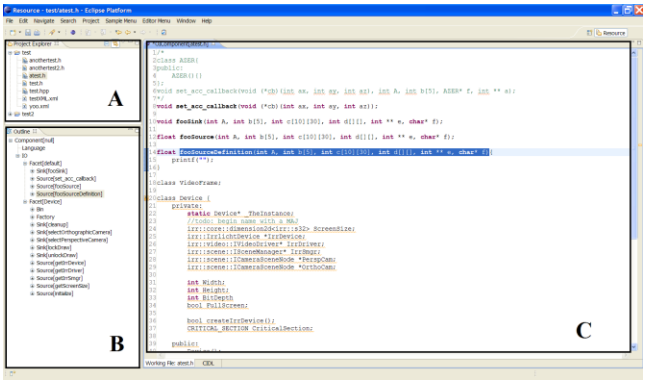

Figure 8: Component Builder Plugin

5.2. Compose Assembly

Figure 5 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 6, top) and further refine the pipeline to actually implement the desired behavior (Figure 6, middle).

The application illustrated in **Figure 6** is a Wiimote-controlled multimodal music player implemented using only open-source software from the web. It provides end users with the ability to navigate through songs and control the volume level in a multimodal way including graphical manipulation and gesture. To implement advanced gesture, we further added a gesture recognizer component to the canvas; connected its input to the Wiimote accelerometer output and its output to the album navigation graphical interface. We then execute the application for learning and recognition phase.

5.3. Run and Debug

Designed applications are directly executed for evaluation from the design environment (or by applications using the runtime API). **Figure 5** shows a full view of the editor, the top toolbar contains buttons to run the pipeline, export the PDCL for manual modification or re-install (generate and compile proxies of) modified components.

Pipelines can be dynamically modified and reconfigured at runtime from either the editor or with the API. From the editor, the user can visually rewire connections, and reconfigure components. Simple debugging is done by wiring selected outputs to data visualizer components embedded in SKEMMI .

6. EVALUATION

In the following parts we present the results of informal evaluations - construction of real applications by third-parties- of the main software tools: OpenInterface Platform and SKEMMI design editor.

6.1. OpenInterface Platform Applications

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

Multimodal Driving Simulator

The runtime platform was first used at eNTERFACE [7], and served as backbone of a multimodal driver simulator [1] for combining the driver state based on video data (e.g., facial expression, head movement, eye tracking) and multimodal input. A first evaluation of the runtime platform was also conducted and promising results such as 5% of memory overhead and 3% of CPU overhead were measured.

MedicalStudio

MedicalStudio [25] 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 in which both researchers and clinicians participate. OpenInterface Kernel has been used as runtime integration platform to study speech and marker detection interaction for 3D object manipulation

UsiXML Interpreter

InterpiXML is a run-time Java rendering engine for UsiXML-specified (www.usixml.org) UIs. It renders a UsiXML UI and incorporates additional interaction modalities such as voice recognition, hand gesture recognition based on video camera, and pen-based interaction. Evaluation experiments, involving a group of 14 users, for comparing the usability of the interpreter with and without OpenInterface has shown equivalent performances (time to perform a task, fulfillment of a task and user evaluation) for both implementations on one hand while the additional flexibility provided by the platform has been demonstrated on the other hand for quickly adding and exchanging modalities with little further programming efforts.

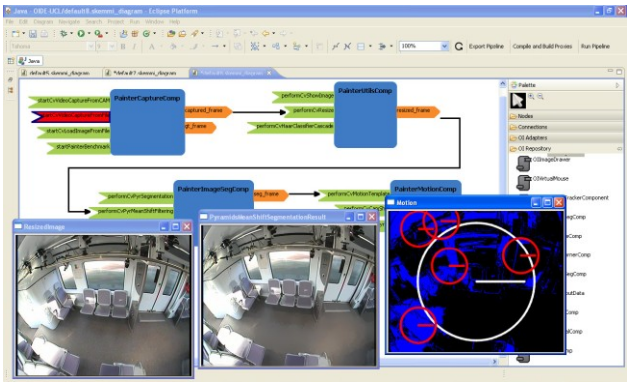

Figure 9: Image Processing Benchmarking with Motion Estimation With Strong Illumination Change .

OpenInterface Project

The OI Project (www.oi-project.org) has developed several demonstrations systems, including multimodal map navigation and a mobile multimodal game. A typical setup for the latter involved a game running on a mobile phone, while the different interaction modalities are running within the platform on a PC. This configuration allowed investigating interaction techniques using: speech commands, 3D gesture commands, and also innovative interaction devices. The runtime kernel has been used with the project dedicated development environment, OIDE.

Image Processing Benchmarking

The platform and SKEMMI editors are been evaluated as tools for benchmarking image processing algorithms (for interaction) in real-time. **Figure 9** illustrates the live application on video frames of a motion templates algorithm to identify pixels motion. OIDE has also been evaluated in this context and didn't meet our requirements due to limited support of the kernel features and usability issues. In this high-computational context, our workbench has displayed realistic performances at runtime (low cpu overhead) and design time (fast composition of pipelines).

6.2. SKEMMI Editor

A first proof-of-concept version of SKEMMI has been evaluated using a group of six experts (programmers, application designers). The outcome allowed us to validate and further refine some features of our approach in bridging the gap between designers and programmers of multimodal interactions: The ability to navigate through three different levels of the same design, basic changes tracking through coloring and the documentation (text, voice) features were widely accepted.

Later informal evaluation of the editor runtime mode emphasized the role of the visual authoring environment (probes, oscilloscope, etc.), especially when designing clutching/declutching mechanisms by combination of signal processing operations. It however highlighted poor support of advanced debugging (breakpoints, step-by-step execution, etc.).

6.3. Component Database

Figure 10 lists a selection of software currently integrated in the platform component database. The components are highly reusable (inside or outside the platform) and come from various projects and experiments.

	Integrated Component
Gestural	•Sketch Recognizer •Gesture Recognizer •Head Tracker (OpenCV) •Wiimote (wiiuse) •Finger Tracking •Data Gloves (5DT, etc.) •Touch Pad (Synaptics) •Face Tracker •Infra-Red dot Tracking (OptiTrack) •Stylus •Hand Posture (HandVU) •Mouse, Keyboard •ARToolkit
Visual	•Generic Pie Menu •Camera(Firewire, USB Webcam) •VirtualMouse •Virtual Pointer
Audio	•Speech Recognizer (Sphinx)

Figure 10: Component Database, a Selection of Components

The database is already well furnished in input devices, and a large number of combinations could be prototyped in order to study new interactions. It also highlights the lack of output devices and interaction techniques. This database is freely available online.

7. CONCLUSION AND FUTURE WORK

The implementation of a multimodal application design suffers from the lack of tools supporting the whole process. There are numerous works targeting different part of the overall complex scheme. We have presented the OpenInterface Kernel runtime platform, which enables reuse of existing components to iteratively design and build a multimodal system with minimal programming effort. We also presented SKEMMI, an extensible design tool built on top of the runtime kernel, which provide feature such as cooperative multi-level application design and component tailoring. Direct manipulation and design-by-demonstration techniques are respectively natively supported by the graphical editor and by dedicated components integrated in the database.

The complete workbench (platform, editors and components) is available for download as open-source software from https://forge.openinterface.org. Moreover, the kernel API and modular approach allows it to be interfaced with any kind of front-end application which can output XML descriptions. We have successfully tested integration with Pure Data; in the near future we will investigate the integration of the runtime platform within solutions such as ICON and *EyesWeb* [3].

Future work will also focus on embedding interaction design techniques like [10][15], usability assessment tools, input devices performance evaluation tools like [22], and on conducting in-depth formal evaluation of the whole workbench.

8. ACKNOWLEDGMENTS

European FP6 SIMILAR Network of Excellence, European FP6-35182 OpenInterface STREP, 3D MEDIA Walloon project. We thank all partners of the OpenInterface STREP Project. We also thank Cyril Carincotte for his early comments, evaluation and feedback on our workbench.

9. REFERENCES

[1] Benoit, A., Bonnaud, L., Caplier, A., Damousis, I., Tzovaras, D., Jourde, E., Nigay, L., Serrano M. and Lawson, J-Y. 2006. Multimodal Signal Processing and Interaction for a Driving Simulation: Component-based Architecture. *Journal on Multimodal User Interfaces*,1, 1,49-58.

[2] Bouchet, J., Nigay, L. (2004). ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces, *Extended Abstracts CHI'04*, ACM, pp. 1325-1328.

[3] Camurri, A., Ricchetti, M., and Trocca, R. 1999. EyesWeb - toward gesture and affect recognition in dance/music interactive systems, *Proc. IEEE Multimedia Systems* '99, Firenze, Italy, June 1999.

[4] 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, *Proc. of* MULTIMEDIA '97. ACM Press, pp. 31-40.

[5] Coutaz, J. Nigay, L. Salber, D. Blandford, A. May, J. and Young, R. (1995), Four Easy Pieces for Assessing the Usability of Multimodal in Interaction the CARE Properties, *Proc.of Interact' 95*, pp. 115-120.

[6] Dragicevic P. and Fekete J-D. Input Device Selection and Interaction Configuration with ICON: *Joint proc. of IHM'01-HCI'01*. Springer Verlag, pp 543-558.

[7] eNTERFACE Workshops, http://www.enterface.net/

[8] Gray, P., Ramsay, A., Serrano, M. A demonstration of the OpenInterface Interaction Development Environment. *UIST'07 Adj. Proc.*

[9] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition. *In IEEE Transactions on Acoustics, Speech, and Signal Processing* 26 (1) (1978) 43-49.

[10] Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. *Proc. of CHI '07*. ACM Press, 145-154.

[11] 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. *Proc. of UIST '06*.

[12] Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. *Proc. of IWAR'99*.

[13] Lawson, J.-Y., Vanderdonckt, J., Macq, B. Rapid Prototyping of Multimodal Interactive Applications Based on Off-The-Shelf Heterogeneous Components. *UIST'08 Adjunct. Proc*.

[14] Lawson, J-Y., 2006. OpenInterface Description Languages Specification. Technical report, 2006: http://www.openinterface.org/platform/documentation

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

[16] Martin, J. C., TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces*, Intelligence and Multimodality in Multimedia Interfaces*, AAAI Press, 1997.

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

[18] Maynes-Aminzade, D., Winograd, T., and Igarashi, T. (2007). Eyepatch: prototyping camera-based interaction through Examples, *Proc. of UIST'07*, ACM Press, pp. 33-42.

[19] Mehta, N. R., Medvidovic, N., and Phadke, S. 2000. Towards a taxonomy of software connectors. *Proc. of* ICSE '00. ACM Press, pp. 178-187.

[20] Merrill, D. and Paradiso, J. A Personalization, Expressivity, and Learnability of an Implicit Mapping Strategy for Physical Interfaces, *Proc. of CHI'05* ACM Press, pp. 2152—2161.

[21] Puckette, M., 1996. "Pure Data: another integrated computer music environment.", *Proc. of the Second Intercollege Computer Music Concerts*, Tachikawa, pp. 37-41

[22] Schedlbauer, M. J. An extensible platform for the interactive exploration of Fitts' Law and related movement time models. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems*. ACM Press, pp. 2633-2638.

[23] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

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

[25] Trevisan, Daniela G.; Nicolas, Vincent; Macq, Benoit; NEDEL, Luciana P. MedicalStudio: a medical component-based framework, *Proc of.WIM '07*.

[26] Vybornova, O., Mendonça, H., Lawson, J.Y., Macq, B. High Level Data Fusion on a Multimodal Interactive Applications Platform. *Proc of ISM'08*.

[27] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl.