A Three-dimensional Characterization Space of Software Components for Rapidly Developing Multimodal Interfaces

Marcos Serrano, David Juras, Laurence Nigay
Grenoble Informatics Laboratory (LIG), University of Grenoble
38042, Grenoble, France
{Marcos.Serrano, David.Juras, Laurence.Nigay}@imag.fr

ABSTRACT

In this paper we address the problem of the development of multimodal interfaces. We describe a three-dimensional characterization space for software components along with its implementation in a component-based platform for rapidly developing multimodal interfaces. By graphically assembling components, the designer/developer describes the transformation chain from physical devices to tasks and vice-versa. In this context, the key point is to identify generic components that can be reused for different multimodal applications. Nevertheless for flexibility purposes, a mixed approach that enables the designer to use both generic components and tailored components is required. As a consequence, our characterization space includes one axis dedicated to the *reusability* aspect of a component. The two other axes of our characterization space, respectively depict *the role of the component in the data-flow from devices to tasks* and the *level of specification* of the component. We illustrate our three dimensional characterization space as well as the implemented tool based on it using a multimodal map navigator.

Categories and Subject Descriptors

H.5.2 **[Information Interfaces and Presentation]**: User Interfaces – Input devices and strategies, Interaction styles, Prototyping, User interface management systems (UIMS); D.2.2 **[Software Engineering]**: Design Tools and Techniques – User interfaces

General Terms

Algorithms, Design, Human Factors, Standardization, Theory.

Keywords

Multimodal Interaction Model, Component-based Approach, Design and Implementation Tool.

1. INTRODUCTION

The multimodal interaction domain has expanded rapidly and significant achievements have been made in terms of both modalities and real multimodal applications. The advent of new modalities based on a variety of captors and effectors coupled with recognition/synthesis mechanisms, as well as the availability of affordable and commonly used devices, such as webcams and game devices (Nintendo Wii Remote [34]) are rapidly enriching the interaction capabilities of desktop computers. As pointed out by Olsen [28], "systems based on one screen, one keyboard and one mouse are the new equivalent of command-line interfaces".

In addition, going beyond desktop interfaces, multimodal applications have a key role to play on mobile and tabletop systems. Mobile devices embed more and more sensors and interaction techniques, such as the accelerometer and the tactile screen with two-handed interaction of the recent iPhone [20]. Moreover tabletop applications are often based on multimodal interaction by combining speech and gesture [38].

Facing the vast world of possibilities for interaction modalities, models and tools for integrating and combining those modalities become a real challenge that we address in this paper. Our discussion will concentrate on multimodal input, from the user to the system, although the presented approach may hold for output as well.

In this paper we describe a characterization space of software components for rapidly developing multimodal interfaces. Our characterization space is independent from component-based technologies, such as Corba [29], and from services-based approaches, such as OSGi [31]. The characterization space defines high-level characteristics of components, that can be implemented in any of those technologies. In this paper we use the term "component" to refer to any type of software unit (e.g., software component, service). We present an implementation of the characteristics defined in our space in a component-based platform, OpenInterface (OI) [13].

The structure of this paper is as follows: we first motivate our approach according to existing tools for multimodal interaction. We then present an overview of the characterization space as well as its scope. We then explain its implementation within the OpenInterface framework and illustrate our approach using one of the multimodal applications developed within the OI framework, a multimodal map navigator whose main features are presented in the next section.

2. ILLUSTRATIVE EXAMPLE: A MULTIMODAL MAP NAVIGATOR

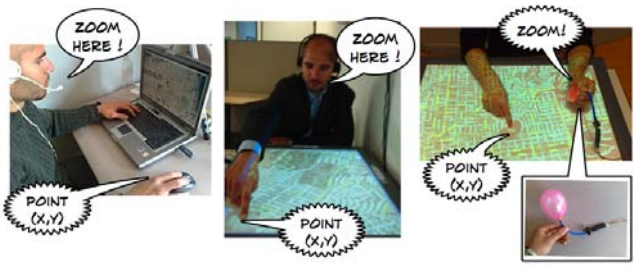
Since the seminal put-that-there [3], many studies have focused on multimodal interaction with a map [7] [38]. We thus consider the classical example of multimodal navigation for a map-based application in order to illustrate and validate our approach.

The multimodal map navigator consists of a map that can be controlled using several interaction modalities. Using a set of devices and the corresponding pure/combined modalities, the user can perform a set of interactive tasks such as panning or zooming. For example, the zoom on a specific point of the map can be specified by combining speech and mouse (Figure 1-a), speech and

ICMI'08, October 20–22, 2008, Chania, Crete, Greece.

pointing gesture (Figure 1-b), or by pressing a balloon along with a pointing gesture (Figure 1-c). A high pressure on the balloon is used to zoom in, and a low pressure to zoom out.

Capture of the finger position is done using the DiamondTouch [9]. Capture of the pressure on the balloon is realized using the interface-Z electronics platform [17], which integrates several innovative sensors such as an atmospheric pressure sensor. Other input devices used with our map-based application include an IR finger tracker [21] and the Wii Remote.



-a- **-b-** **-c-**

Figure 1. Multimodal interaction with a map.

3. RELATED WORK

Nowadays we observe an increasing interest in software tools for multimodal / post-WIMP User Interfaces (UI).

Several recent studies especially in the fields of ubiquitous computing and augmented reality, such as phidgets [14] and the iStuff toolkit [1], focus on innovative and multi-devicesinteraction. Most toolkits are designed for specific devices and/or interaction paradigms, such as camera-based interaction [23], 3D interaction [32], interactive information visualization [15] or multi-person concurrent interfaces [33]. Those toolkits allow implementation and configuration of specific interaction techniques, usually supporting devices efficiently but in ad-hoc ways. Moreover they cover only the device level of interaction or they provide a simple direct mapping between devices and tasks. Moreover they do not consider the combination of modalities.

As opposed to toolkits dedicated to programmers, other tools for post-WIMP UI are dedicated to prototyping such as CrossWeaver [35], BOXES [17] and the CSLU Toolkit [36]. Such tools do not require programming skills.

In this context our goal is to define a tool dedicated to multimodal UI (1) that supports multiple devices, being able to encapsulate existing post-WIMP UI toolkits (2) that enables the description of the complete interaction from devices to tasks (3) that allows us to define multimodal design solutions without programming skills (4) that supports the rapid development of multimodal interaction for quickly exploring different design solutions.

As in Magglite [18] and its related toolkit ICON [10] as well in our previous work ICARE [4], we adopt a data-flow approach that has been shown to be adapted for specifying multimodal interaction. Our contribution is to characterize the building blocks (called here components) that define this data-flow from devices to application tasks. The following section presents this characterization space. We then present a design/development tool based on components with these characteristics.

4. CHARACTERIZATION SPACE OF COMPONENTS

After presenting the scope and an overview of our characterization space of multimodal UI components, we explain how such characteristics will enable us to reach our objectives stated in the previous section and motivated by the limitations of existing tools.

4.1 Scope and overview of the space

Our characterization space for multimodal interaction defines a set of characteristics of components, understanding the term "component" as any type of software unit (e.g., software component, service). As pointed out by Morch [24], component-based software development (CBSD) allows users to tailor existing applications by assembling high-level components. However our characterization space does not define the execution behaviour of the components or the communication mechanisms between components, as do traditional component-based models, such as the Corba Component Model (CCM) [29], the Component Object Model (COM) [8] and JavaBeans [11], or services-based approaches, such as OSGi [31]. Our space can be used to define and implement high-level characteristics of components that can be implemented in any of those technologies.

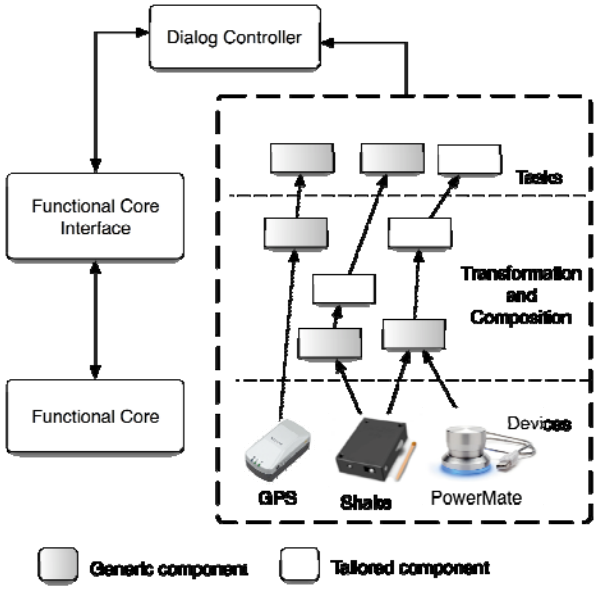


Figure 2. Different types of components within the ARCH software architecture [37].

Concerning the development of an interactive system, to fully understand the scope of our approach, we show in Figure 2 where the corresponding components are located within the complete code of the interactive system structured along the ARCH software architectural model [37]: our approach focuses on the interaction part of an interactive system. For example, in our illustrative example, we consider that we have a functional core of a map application and we focus on the design and implementation of the interactive part of the system. By doing so, for the implementation of our example, we were able to experiment on multimodal interaction with different map applications (Functional Core) including GoogleEarth [13], a map server developed by France Telecom and a stand-alone application in Java. Now that we have

presented the scope of our approach, we give an overview of the characterization space before detailing its key points in the next paragraphs.

Our space is defined along three dimensions as shown in Figure 3. The **first** dimension of the space describes the **genericity** of the components. Our space includes both generic and tailored components. Generic components represent high-level reusable abstractions. Tailored components implement operations for specific devices or for application-dependent tasks. The **second** dimension of the space is related to the **data-flow** from input devices to an interactive application. Along this axis, we identify four types of components: we reuse from ICON [10] the three types of components (Device, Adapters and Application components) corresponding to three levels of abstraction that we respectively name Device, Transformation and Task. From ICARE [4], we reuse the CARE Composition components. The **third** dimension of the space is related to the **component approach**. Along this dimension, we identify two levels of abstraction for specifying a component: Software Component and Interaction Entity.

As shown in Figure 3, this component characterization space is wider than the ones of ICARE and ICON.

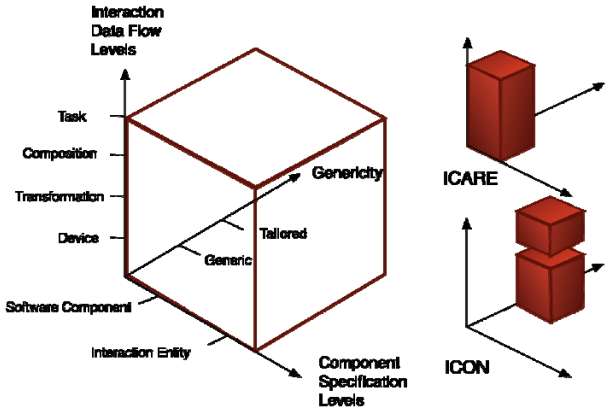


Figure 3. -a- Three axes of our characterization space. -b- Comparison with ICARE and ICON.

In the next paragraphs of this section we describe how those three dimensions respond to some of the limitations identified in Section 3.

4.2 From input devices to interactive tasks

As explained before, existing approaches intend to connect input devices to interactive tasks. While the ICON approach is interesting, using adapters between the input devices and the tasks, it does not treat the multimodal fusion of modalities. For defining the combination of modalities, we reuse the CARE components from the ICARE tool [3]. The CARE components are fully described in [25].

Our space advocates four types of components that will help both designers and developers to fill the gap existing today between input devices and interactive applications. The four types of components are organized into three levels: Task (task components), Transformation Chain (transformation and composition components) and Device (device components).

Input devices define the entry points of data in the transformation chain (at the bottom in Figure 2). The transformation chain is responsible for transforming input values coming from input devices into values adapted to the interactive task under design. The transformation chain is composed of two classes of components, namely transformation and composition components.

At the end of the transformation chain (on top in Figure 2), task components are responsible for the link between the multimodal interaction specified in the assembly of components and the rest of the application. If the application is structured along the ARCH software architectural model (Figure 2), task components will communicate with the Dialog Controller.

4.3 Designing Multimodal Interaction: description levels

Existing approaches are not made for designers, as they require programming skills. As part of our iterative user-centered design process, we would like designers to be able to build multimodal interaction using components. The main issue is the component description level, as designers need different information than developers. For example, for a given device a designer will be more interested by the human sense involved, the weight of the device, its dimensions, etc. The developer will be more interested by the driver library, its programming language, the types of the parameters, etc.

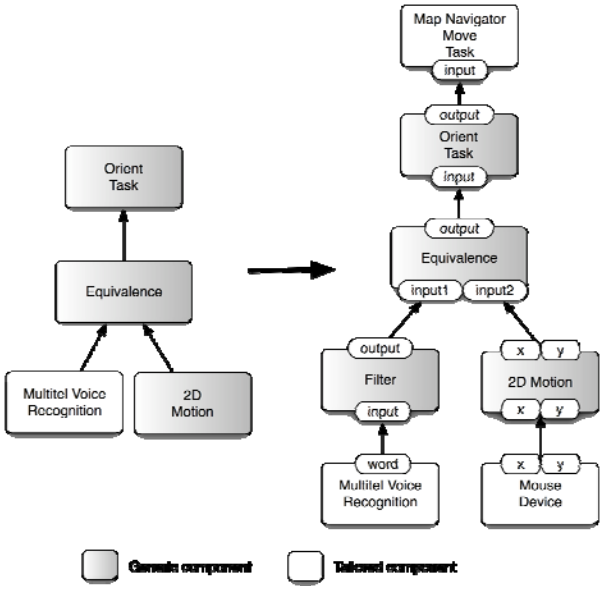


Figure 4. Designer assembly using interaction entities (left) and developer assembly using software components (right).

The component specification axis defines two levels of abstraction for specifying a component. Such levels define different points of view on components that can be manipulated by the designer/developer. The **Interaction Entity** level corresponds to general characteristics of the component, without implying any logical or software structure. The description of an Interaction Entity is therefore independent of the component-based approach. Indeed the description includes attributes related to its nature: HCI and physical attributes for device component, mathematical or physical attributes for transformation components, semantic or temporal attributes for composition components, and Foley's taxonomy-based attributes for task components [12]. An Interaction Entity can then

be implemented in different **Software Components**. Each Software Component belongs to a component technology (such as Corba or OSGi), has a license, a price, and is implemented in a specific language (Java, C++), etc.

Figure 4 illustrates two assemblies of components: on the left, an assembly of interaction entities made by a designer. On the right, the corresponding assembly implemented by a developer using software components.

4.4 Need for genericity

The major goal of our approach is, through the exploration of design solutions, to enable the fast development of multimodal applications, promoting reusability and simplifying the design and development of multimodal interfaces. In order to attain this, the key point is the genericity of the manipulated units. There are two main reasons for looking for genericity: **expressive leverage** and **reusability**. In this section we detail those two points and then present our approach for defining generic components.

4.4.1 Genericity for expressive leverage

Using generic components improves the expressive leverage of our approach, as pointed out by Myers [26] and Olsen [28]. "Expressive leverage is achieved when a tool reduces the total number of choices that a designer must make to express a desired solution" ([28]). In order to improve the expressive leverage of a tool, "generalize and reuse" is one of the solutions.

As seen in Figure 4, the designer (left assembly) can choose a generic component (e.g., 2D Motion) instead of a specific input device (e.g., Mouse). The developer will then select a tailored component. This reduces both the choices for the designer and the developer.

At the same time, tailored components allow both the designer and the developer to have more freedom. The notion of tailored component has been defined in [24] as a component supporting domain-oriented functionalities. In our case, tailored components are components implemented in an ad-hoc way.

4.4.2 Genericity for reusability

In our approach the key point is to identify generic components that can be used for different multimodal applications. Such generic components will enable the rapid development and exploration of design solutions for multimodal interaction.

We have identified two levels of reusability: reusing a component and reusing a set of components. For example, in Figure 4, generic components, such as the Equivalence component, can be reused in another assembly (reuse of one component). At the same time, if the developer wants to use another input device instead of the Mouse, he just has to replace the Mouse component. He is reusing the whole assembly (reuse of set of components), thanks to the generic device 2D Motion, that allows the easy connection of different input devices.

Concerning the reusability of tailored components, they may be reusable, but usually tailored components are implemented for the needs of a specific interactive application. For example, in Figure 4, the Voice Recognition and the Mouse components are tailored, as they use specific drivers. We could say that, as the Mouse component is reusable, it should be generic. However, we define generic devices according to the interaction actions, and not to the captured data. We will detail this in the next section by introducing the notion of generic component in relation to the different data-flow levels (task, transformation, composition, device).

4.4.3 Models for genericity

In order to define generic components at the different levels identified before, i.e. task, composition, transformation and device levels, we use existing and well-known models of human-computer interaction. We will next present the models used for each level.

Generic devices

On the bottom of the data-flow, to define generic device components, we use and extend Buxton's taxonomy [5]. As explained in [22], several taxonomies exist for input devices. Buxton's taxonomy classifies continuous hand-controlled input devices along a classification matrix, using the degree of freedom and the sensed property as main axes. Some studies also provide taxonomies dedicated to a specific class of devices, such as the taxonomy of Hinckley [16] for touch-sensing devices. We base our approach for defining generic device components on Buxton's taxonomy, shown in Figure 5 with some examples of input devices.

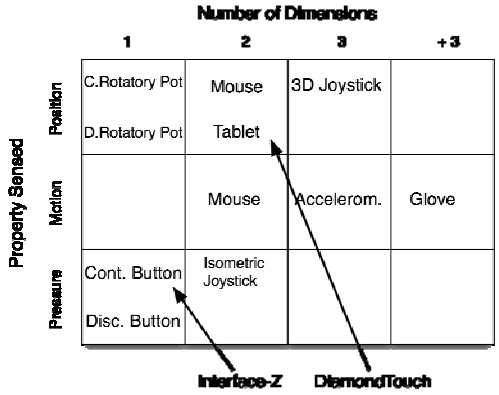


Figure 5. Buxton's taxonomy.

Other types of input devices, such as speech or image analysis, are difficult to classify since several levels of abstraction can be defined for describing them. We do not treat them in the taxonomy. Those non-classified devices can then be described in our model using tailored components. In our future work we will study how to classify and define abstractions of those other input devices.

Generic tasks

On the upper part of the data-flow, generic and reusable tasks components are based on Foley's interactive tasks [12]. For example, the Orient task of Figure 4 is one of Foley's tasks.

Generic transformation

Generic transformations are based on the generic devices as defined before. Generic transformations implement reusable operations that are usually performed on data from a generic device (e.g., the Filter component of Figure 4, that filters input events according to a generic specification).

Generic composition

In order to create generic composition components, from ICARE [4] we reuse the CARE Composition components. For example, in Figure 4 we use an Equivalence composition between the mouse

and the voice recognition. The CARE components are fully described in [25].

Table 1 summarizes the notion of generic component for the different data flow levels presented above.

In this section we have presented our three-dimensional characterization space of components along with its key points and benefits for the rapid development of multimodal applications. In the next section we present one implementation.

	What ?	How ?
Task	Application independent	Foley's task Configurable generic task
Composition	Modality independent	CARE properties
Transformation	Device and application independent	General Transformations: mathematical, physical, mapping operations
Device	Abstraction of device data in terms of interaction actions, Buxton properties	Buxton's properties

Table 1. Genericity according to the different levels of the interaction data-flow.

5. IMPLEMENTATION

The identified characteristics have been implemented in a component-based approach. We remind the reader that instead of component-based platform, we could have used a service-oriented approach. We implemented the characteristics defined in our space in the OpenInterface (OI) framework, a component-based platform. Several generic and tailored components have been implemented in the OI framework. Moreover, using the OI framework, we have implemented the multimodal map navigator of Section 2.

The OI framework is composed of the OpenInterface Interaction Development Environment (OIDE), a graphical environment, and by the OI Kernel, the underlying runtime platform. The OI Kernel is a component-based platform that handles distributed heterogeneous components based on different technologies (Java, C++, Matlab, Python, .NET). The heterogeneity of the platform allows the integration of existing interaction techniques written in different languages. The OI kernel manages the creation and the connection between components at runtime by dynamically loading and interpreting the OI interaction descriptions or "pipelines". Pipelines can be created graphically using the OIDE, built on top of the OI kernel.

The OIDE is a graphical environment that allows direct manipulation and assembling of components in order to specify a "pipeline" defining a multimodal interaction. Such graphical tools have already been used for allowing users to prototype their application, such as the CSLU Toolkit for spoken language interaction ([36]), or the Yahoo! Pipes for web service composition ([39]). Figure 6 gives an example of a pipeline in our tool. The OIDE includes a component repository as well as construction, debugging and logging tools [13].

The OI kernel and the OIDE allow the dynamic assembly of components. Moreover, components are pre-compiled, giving a great flexibility to the tool and making it possible to perform rapid design changes. This flexibility is one of the key properties of UI tools as identified in [28].

Many other multimodal applications have been implemented using OI and its underlying component-based approach, such as a multimodal helicopter game on PC, a 3D multimodal game on a mobile phone and a multimodal slide show. In the next section we present the implementation of the multimodal map navigator presented in Section 3.

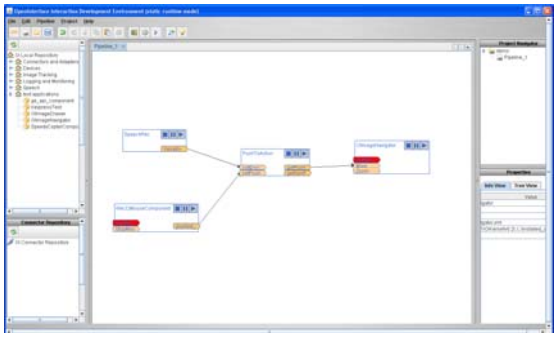


Figure 6. OIDE graphical editor.

6. ITERATIVE DESIGN SCENARIO

Using the software tool presented in the previous section, we have implemented the multimodal map navigator introduced in Section 2. We will focus on one task of the interactive system: zooming on a specific point. This task is quite interesting from a multimodal point of view as it may imply multimodal composition of two complementary modalities: one for specifying the point and one for activating the zoom command.

Step 1. Design of the multimodal interaction

The designer defines the interaction techniques of the different interactive tasks of the multimodal application. For each one of those tasks, the designer creates an assembly of generic and tailored interaction entities.

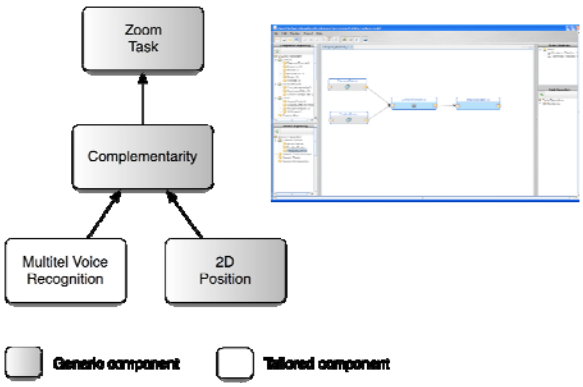


Figure 7. Designer Assembly for specifying a zoom task and screenshot of the same assembly in the OIDE.

Generally, designer assemblies look like the assembly of Figure 4, a composition of two input modalities. In the example of Figure 7, we can see the representation of the assembly for specifying the multimodal interaction for the zoom task along with a screenshot of the same assembly implemented in the OIDE. Notice that assemblies

in the OIDE are done horizontally. In this paper, for clarity purposes, we will show vertical representations of the assemblies rather than screenshots of the OIDE.

Step2. From the designer assembly to the technical assembly

Once the designer has specified the multimodal interaction, the developer creates a technical assembly from the designer's assembly. This generated technical assembly has to be completed by the developer. He will specify the connections between components and will add missing components (such as a specific device).

In our example, the developer wants to implement a first version that can be tested on a laptop computer. For doing so, he chooses a mouse device for specifying the 2D Position. The mouse gives the coordinate of the point to zoom and the speech recognition supports zooming commands such as "zoom here". Figure 8 illustrates this assembly.

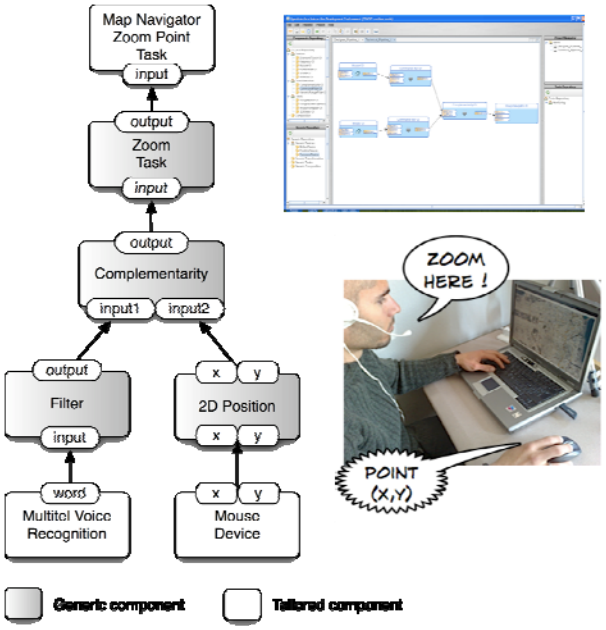


Figure 8. Zoom task using the Mouse and the Speech Recognition.

Step 3: Easily changing devices: from the PC to the tactile surface

Once the PC version is ready, the developer wants to test it on a large display surface using the DiamondTouch for capturing the position of the finger. He just replaces the Mouse component by the DiamondTouch component, as shown in Figure 9.

Step 4. Adding a modality combination

During the informal evaluation of the previous version, we noticed some problems of synchronization between the speech recognition and the pointing on the table. The designer wants to use another modality for the zoom command in order to have a bi-manual multimodal interaction. The designer looks into the available devices and selects the balloon coupled with an Interface-Z captor. A high pressure on the balloon is used to zoom in, and a low pressure to zoom out. He also would like to test the performance of using a double pedal for the zooming command, where each pedal performs a different zoom. He wants to use both the balloon and the pedal in an equivalent way: the user can press the pedal or press the balloon to launch a zoom command. Figure 10 shows the corresponding assembly.

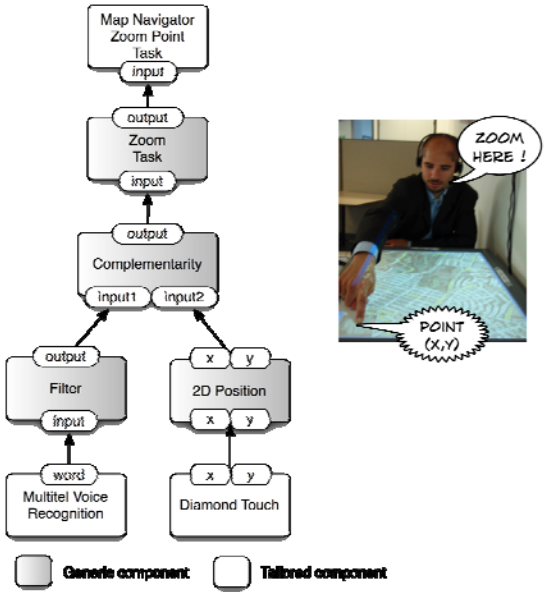


Figure 9. Zoom task using the DiamondTouch and the voice.

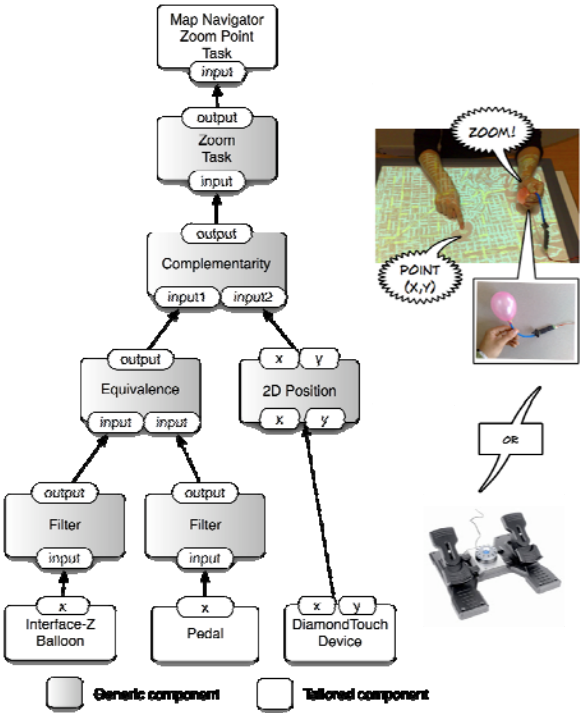


Figure 10. Equivalence composition between the pedal and the balloon.

Step 5: Modifying the modality combination

The evaluation of this equivalent interaction shows that some zoom commands have been launched accidentally due to the high sensibility of the balloon. The designer would like to increase the robustness of the application. For doing so, the developer changes

the Equivalence composition between the pedal and the balloon by a Redundancy composition, that forces the user to use both the pedal and the balloon at the same time. Figure 11 shows the corresponding assembly.

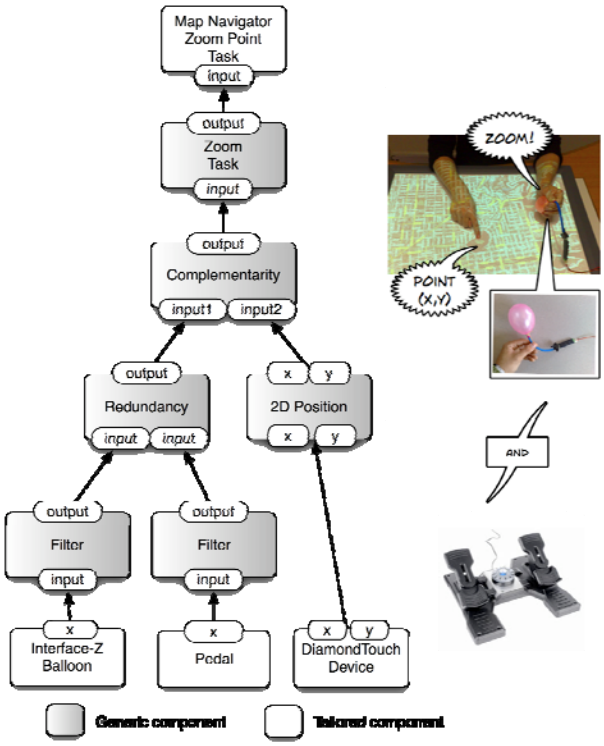


Figure 11. Redundancy composition between the pedal and the balloon.

In this section we have shown an example of iterative design of a multimodal interaction for a given task. The implementation of the characteristics of our space presented here shows how our approach allows the rapid development of multimodal interaction. This example has been implemented by both designers and developers using our tool, and an informal evaluation has been carried out.

7. CONCLUSION AND FUTURE WORK

This article has introduced a new characterization space of software component for rapidly developing multimodal interactions. The space supports generic and tailored components at different levels of abstraction within the transformation chain of raw data acquired from input devices to an application-dependent task. The space also defines two levels of abstraction for specifying components that describe different points of view on components. This flexibility allows users with different technical backgrounds to use a tool implementing these component characteristics. The approach has been implemented in the component-based OpenInterface (OI) framework and several multimodal applications have been implemented using this framework. Such a framework, that includes generic software components, enables the rapid development of multimodal interaction as a central tool for an iterative user-centered design approach. Two test-beds, namely a map-based application and a game, in two different settings (i.e., an augmented environment and a mobile platform), have been developed using the OI framework. Ergonomic evaluations have been performed and we further plan to study participatory design activities using the OIDE (i.e., the graphical environment of the OI framework) and our underlying characterization space.

As further work on the approach, in addition to enriching the framework with new generic components (including logical device components and transformation components), we plan to focus on output multimodal response such as a multiple display set-up and to further study the links between the input and output components. We also started to validate the genericity of our space by using it with a service-based runtime platform.

8. ACKNOWLEDGMENTS

The OpenInterface OI Project is an IST FP6 STREP funded by the European Commission (FP6-35182), www.oi-project.org. The authors thank the OI colleagues for their contribution.

9. REFERENCES

[1] Ballagas, R., Ringel, M., Stone, M. and Borchers, J. (2003). iStuff: a physical user interface toolkit for ubiquitous computing environments. Proc. of Human factors in computing systems 2003, ACM Press, pp. 537-544.

[2] Benoit, A., Bonnaud, L. Caplier, A., Damousis, L., Tzovaras, D., Jourde, F., 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, pp. 49-58.

[3] Bolt, R. A. (1980). Put-that-there: voice and gesture at the graphics interface. SIGGRAPH'80, 14, 3, pp. 262-270.

[4] Bouchet, J., Nigay, L. (2004). ICARE software components for rapidly developing multimodal interfaces. In Proceedings of ICMI 2004, State College, USA, pp. 251-258.

[5] Buxton, W. (1983). Lexical and pragmatic considerations of input structures. ACM SIGGRAPH CG, 17(1), pp. 31-37.

[6] Card, S., MacKinlay, J., and Robertson, G. (1991). A morphological analysis of the design space of input devices. ACM Transactions on Information Systems, 9, pp. 99-122.

[7] Cohen, P., McGee, D., Clow, J. (2000). The efficiency of multimodal interaction for a map-based task. Proc. of the 6th ANLP Conference, ACM Press, pp. 331-338.

[8] COM. www.microsoft.com/com

[9] Dietz, P. and Leigh, D. (2001). DiamondTouch: a multi-user touch technology. Proc. of UIST'01, ACM Press, pp. 219-226.

[10] Dragicevic, P., and Fekete, J. D. (2001). Input device selection and interaction configuration with ICON. Joint Proc. of IHM'01 and HCI'01,Springer Verlag, pp. 543-558.

[11] EJB. http://java.sun.com/products/ejb/.

[12] Foley, J., Wallace, V.L. and Chan, P. (1984) . The human factors of graphics interaction techniques. IEEE Computer Graphics and Applications, 11, pp. 13-48.

[13] Gray, P., Ramsay, A., Serrano, M. (2007). A demonstration of the OpenInterface Interaction Development Environment. UIST'07 Adjunct Proc., ACM Press, pp. 39-40.

[14] Greenberg, S. and Fitchett, C. (2001). Phidgets: easy development of physical interfaces throught physical widgets. In Proc. of UIST'01, ACM Press, pp. 209-218.

[15] Heer, J., Card, S., Landay, J. (2005). Prefuse: a toolkit for interactive information visualization. Proc. of the SIGCHI conference on Human factors in computing systems, ACM press, pp. 421-430.

[16] Hinckley, K., and Sinclair, M. (1999). Touch-sensing input devices. ACM. Proc. of CHI'99, ACM Press, pp. 223-230.

[17] Hudson, S., Mankoff, J. (2006). Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In Proc. of UIST'06, ACM Press, pp. 289-298.

[18] Huot, S., Dumas, C., Dragicevic, P, Fekete, J. and Hégron, G. (2004). The MaggLite post-WIMP toolkit: draw it, connect it and run it. In Proc. of UIST'04, ACM Press, pp. 257-266.

[19] Interface-Z, www.interface-z.com.

[20] iPhone, www.apple.com/iphone.

[21] Letessier, J., Berard, F. (2004). Visual tracking of bare fingers for interactive surfaces. In Proc. of UIST'04, ACM Press, pp. 119-122.

[22] Lingrand, D. and Riveill, M. (2006). Input interactions and context component based modelisations: differences and similarities. Proc. of AVI'06. ACM Press. pp. 19-22.

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

[24] Morch, A. I., et al. (2004). Component-based technologies for end-user development. Comunications of the ACM, Volume 47, Issue 9, pp. 59-62.

[25] Multitel. www.multitel.be

[26] Myers, B., Hudson, S. E., Pausch, R. (2000). Past, present, and future of user interface software tools. Transactions on Computer-Human Interaction, Vol. 7, No. 1, pp. 3-28.

[27] Nigay, L., Coutaz, J. (1997). Multifeature systems: the CARE properties and their impact on software design. intelligence and multimodality in multimedia interfaces, AAAI Press.

[28] Olsen, D. R. Jr. (2007). Evaluating user interface systems research. Proc. of UIST'07, ACM Press, pp. 251-258.

[29] OMG - CORBA. www.corba.org

[30] OpenInterface European project. IST Framework 6 STREP funded by the European Commission (FP6-35182). www.oi-project.org.

[31] OSGi. www.osgi.org

[32] Ray, A., Bowman, D. A., (2007). Towards a system for reusable 3D interaction techniques. Proc. Of the 2007 ACM symposium on Virtual reality software and technology, ACM Press, pp. 187-190.

[33] Shen, C., Vernier, F., Forlines, C., and Ringel, M. (2004). DiamondSpin: an extensible toolkit for around-the-table interaction. Proc. of CHI'04, ACM Press, pp. 167-174.

[34] Shirai, A., Geslin, E., Richir, S. (2007). WiiMedia: motion analysis methods and applications using a consumer video game controller. Proc. of the SIGGRAPH Symposium on Video Games, ACM Press, pp. 133-140.

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

[36] Sutton, S., Cole, R. (1997). The CSLU toolkit: rapid prototyping of spoken language systems. Proc. of UIST'97, ACM Press, pp. 85-86.

[37] The UIMS tool developers workshop, A metamodel for the runtime architecture of an interactive system. SIGCHI Bulletin (1992), pp. 32-37.

[38] Tse, E., Shen, C., Greenberg, S. and Forlines, C. (2006). Enabling interaction with single user applications through speech and gestures on a multi-user tabletop. Proc. of AVI'06, ACM Press, pp. 336-343.

[39] Yahoo! Pipes. http://pipes.yahoo.com