I*CHAMELEON:
AN MVC-BASED MIDDLEWARE FRAMEWORK FOR THE SUPPORT OF MULTIMODAL APPLICATION DEVELOPMENT

LO WAI KWAN

M.Phil

The Hong Kong Polytechnic University

2014

The Hong Kong Polytechnic University

Department of Computing

i*Chameleon: An MVC-based Middleware Framework for the Support of Multimodal Application Development

Lo Wai Kwan

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Philosophy

June 2013

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____________________________ (Signed)

LO WAI KWAN (Name of student)

Abstract

Multimodal human computer interactions are becoming increasingly popular, especially in ubiquitous and pervasive computing applications. These applications demand highly responsive and intuitive human control interfaces. Because of their nature and form factor, the traditional keyboard, video and mouse interfaces are often not appropriate or adequate. As a result, there is much current research on developing novel interaction devices and sensors, or algorithms for signal processing. However, there are still challenges when it comes to integrating and customizing different heterogeneous devices into a human-centered multimodal application. In addition, owing to the static binding between user control and the application and the strong coupling between the application programming interface (API) and heterogeneous devices, the development of multimodal applications remains a difficult task.

In this thesis, we introduce *i*Chameleon*, which not only leverages a principled and comprehensive development cycle that systematically captures the principles behind multimodal interaction, but also provides a configurable and extensible multimodal platform to support the development of highly interactive applications. Through the use of an MVC architectural pattern, it enforces the principle of *separation-of-concerns* to facilitate cross-collaboration between device engineers, programmers, modality designers and interaction designers who are working on different aspects of human computer interaction and programming. Collectively, the development efforts are combined, integrated and compiled by the i*Chameleon kernel to create the multimodal interactive application. During the execution, i*Chameleon also supports dynamic adaption across components according to the contextual information from the surrounding environment. For example, if a user is accessing a video via a regular smart phone on i*Chameleon; and a high-resolution display is then discovered, the video can be streamed to the display to take advantage of the higher resolution, without having to modify and re-compile the application, or even having to restart the application. This capability moves the multimodal applications closer to being human-centered rather than device-centered. In the process, usability and flexibility of the applications are enhanced.

To validate the soundness of i*Chameleon, we implemented the platform based on two approaches: web services and publish/subscribe architecture. We carried out two experimental applications, Mobile DJ and interactive robot exhibit. Mobile DJ was implemented over web services to test the support for multimodal interactions over distributed components in real time, regardless of the users' locations. Players browse and search for sound tracks that are currently being worked on by others based on the web services supported, which provides a channel for them to contribute collaboratively. In the second experiment, an interactive robot exhibit was developed using publish/subscribe middleware to demonstrate dynamic adaption. Modalities and devices can be changed according to the users' behavior (e.g., location) and the contextual environment (e.g., level of loudness).

Both experimental applications produce positive results. The experience shows that the use of i*Chameleon can help to decompose the development process into different aspects and each aspect can be developed fairly independently. The overall achievement is that the interaction components become more reusable and the system itself becomes more flexible, validating the design of i*Chameleon.

List of Publications

Journal Article

[1] **Lo, K. W. K.**, Ngai, G., Chan, A. T. S., Leong, H. V., & Chan, S. C. F. (2013). i*Chameleon: An MVC Engineering Approach for developing Pervasive Multimodal Applications. *Software: Practice and Experience*. 2013 (In Preparation)

Conference and Workshop Papers

[1] **Lo, K. W. K.**, Tang, W. W. W., Ngai, G., Chan, S. F., & Tse, J. T. P. (2010). Introduction to a Framework for Multi-modal and Tangible Interaction. *IEEE International Conference on Systems, Man, and Cybernetics - SMC*, 3001–3007. doi:10.1109/ICSMC.2010.5641977

[2] Tang, W. W., **Lo, K. W. K.**, Chan, A. T. S., Chan, S., Leong, H. V., & Ngai, G. (2011). i*Chameleon: a scalable and extensible framework for multimodal interaction. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems* (pp. 305–310). New York, NY, USA: ACM. doi:10.1145/1979742.1979703

[3] **Lo, K. W. K.**, Tang, W. W., Leong, H. V., Chan, A., Chan, S., & Ngai, G. (2012). i * Chameleon : A Unified Web Service Framework for Integrating Multimodal Interaction Devices. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on* (pp. 106–111). Lugano. doi:10.1109/PerComW.2012.6197460

[4] **Lo, K. W. K.**, Tang, W. W. W., Ngai, G., Chan, A. T. S., Leong, H. V., & Chan, S. C. F. (2013). i * Chameleon : A Platform for Developing Multimodal Application with Comprehensive Development Cycle. *Proceedings of the 28th*

Annual ACM Symposium on Applied Computing. Coimbra, Portugal: ACM. doi: 10.1145/2480362.2480570

[5] **Lo, K. W. K.**, Lau, C. K., Huang, X. L. M., Tang, W. W. W., Ngai, G., Chan, S. C. C. F. (2013). Mobile DJ: a Tangible, Mobile Platform for Active and Collaborative Music Listening. *Proceedings of the 13th International Conference on New Interfaces for Musical Expression*. Daejeon and Seoul, Korea Republic.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr Grace Ngai for her invaluable support, advice, insight, and guidance throughout this interesting and challenging research work. It is a great pleasure and honor for me to be her student and to work closely with her. I will never forget her kind encouragement, patient support and tolerance of my last-minute work habit, and even the days she took her valuable time working with me overnight. She enlightens me not only on academic research, but also values and goals in life. She gave me many opportunities to explore the world through novel technologies and community services. Dr Ngai is the best teacher I ever met.

I am also very thankful for the valuable feedback and support of my co-supervisors, Dr Stephen Chan, and the other faculties who generously shared their knowledge and experience with me: Dr Alvin Chan and Dr Hong-va Leong from the Department of Computing. Dr Stephen Chan and Dr Alvin Chan taught me about software engineering. Dr Leong shared his knowledge of signal processing and algorithm design. Without their help, I could not have completed my research.

I would like to thank all members from eToy Laboratory for their support and encouragement. Special thanks to Mr Will Tang for helping to develop my supervisors' ideas into the i*Chameleon platform; Mr Kin Lau, Mr Michael Huang for helping with the hardware development and prototyping.

Finally, I would like to sincerely thank my family for their understanding and tolerance. I know that I have not spent enough time on them over the years.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

In 1988, Mark Weiser coined the term “ubiquitous computing”, and predicted that it would become the third wave in the field of computing science [51]. Ubiquitous or pervasive computing describes a “smart world”, where objects have communication capability and are integrated with human users. Within the smart environment, portable and smart devices are embedded everywhere and computation is invisibly woven into daily life, creating an “All time everywhere computing” [49] reality. Users can, based on their abilities, interests and environment constraints, choose their own way to interact with an object or achieve a task.

At the same time, riding on the success on heterogeneous devices [38][21], computer processing power and storage capabilities [1], human-computer interaction has progressed to the point where it is no longer bound to the conventional keyboard, video and mouse (KVM) interface. Even today, users can control their applications through different human-oriented devices, such as game controllers, motion capture cameras and speech recognition engines. For example, users are allowed to adjust the view angle of the display [62] by Nintendo Wii controllers. In this way, embedded heterogeneous devices have been changing the way that users interact with applications. The result is a trend towards human-computer interaction that is more human-natural, intuitive and robust.

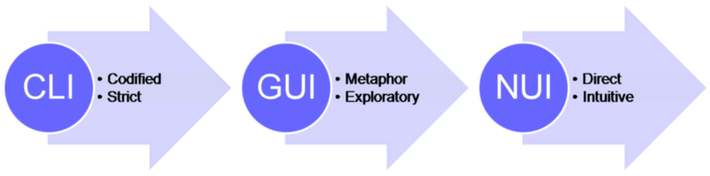


Figure 1-1 Evolution of user interfaces

According to Harbour Research [29] in 2008, 1.75 billion controllers and smart sensors are embedded in our daily environment. The possibility, if these sensors are networked, are that they will be able to provide human-oriented services anywhere, anytime [28], and offer rich context about the user’s state and surroundings. This contextual information would then have the potential to minimize human intervention towards the application, providing a proactive interface that anticipates decisions to meet user needs. This integration between sensors and mobile devices, coupled with the improvement of computation power and enhancement of high-resolution display, has hugely impacted human-computer interaction.

At the same time, mobile computing has also taken off. Modern lifestyles and habits have brought about an increasing need to access information everywhere and at any time, which has changed the interaction style from desktop-based to mobile-based. The total usage time of smartphones now outranks laptops, and it has been shown [46] that there exists a strong relation between user experience and pervasiveness of mobile devices. Users tend to access different information from different channels on a single portable mobile device, such as smartphones or tablet computers. This evolution suggests that adaptation of smart mobile devices to existing applications will become a trend of human-computer interaction in the future.

Mobile interactions normally involve more than one modality. Modality is defined as the type of communication channel used to convey or receive information. It also represents a way of expressing ideas, perceiving views or performing an action [42]. Modalities can be active or passive. An active modality refers to an action which the user issues intentionally while a passive modality represents a command that is not explicitly expressed by the user’s will [8][43]. In mobile platforms, both active and passive modalities are involved. When the user triggers a command, such as taking a photo or sending a message, the smartphone will gather context information, such as location, without notifying the user. These passive modalities increase the information bandwidth available to the command and provide a better interaction experience for the user. Such interactions involving more than one modality are called multimodal human-computer interaction (MMHCI).

MMHCI interacts along different types of communication channels to extract and convey meaning automatically. It have been proven to prevent errors, increase robustness, enrich communication bandwidth and enhance adaptability to different situations and environments [14]. Nowadays, MMHCI has begun to take root in consumer products, as exemplified by the success of commercial products that rely heavily on non-traditional modes of interaction. It has applications in different areas [31], such as ambient spaces. It expands computing beyond desktop and integrates to everyday objects, for example, smart conference room applications [40], mobile computing [26], wearable computing [41], virtual environments and affective computing [55].

Although heterogeneous devices and pervasive modalities exist everywhere, sensors, actuators and hardware resources are still constrained within a single device. It is difficult for users to access different modalities across mobiles and bind sensors from different locations to form customized applications. Also, extra efforts are required to transfer a task from a device to another device. Therefore, existing applications are still tightly coupled and device-oriented but not human-oriented. Figure 1-2 (left) illustrates an example of tight coupling between modalities and devices. The same smartphone contains an accelerometer, light sensor, gyroscope or other actuators, but they are associated with the same device, and the whole pervasive system is self-contained like a black box. It would be difficult to share resources among devices. However, the ultimate goal of pervasive environment is to provide a continuous or uninterrupted user experience when the user moves across devices. Therefore, if the strong coupling can be released and each heterogeneous widget has the ability to discover the capabilities of other devices, a pervasive environment with multimodal interaction can be established. For example, also in Figure 1-2 (right), after decoupling of sensors, actuators and hardware resources, smartphones can bind to suitable components according to the need and context environment. Under the assumption of well-established framework and consummate development process, an “Interaction Cloud” can be formed.

The rationale behind an “Interaction Cloud” is as follows. Cloud computing itself is widely defined by different research areas [34] and it refers to applications delivered as a service over the Internet [4]. Under the context of multimodal interaction, each widget can be defined as a service, for example, an interactive device providing a tilting action can be regarded as “Hardware as a service” (HaaS) while a fusion algorithm can be classified as “Software as a Service” (SaaS). However, most research efforts in cloud computing focus on computational performance, synchronization or resource utilization. There is relatively little effort on the modeling of the various hardware and software devices and widgets that make up the interaction aspect of the application. This means that although different components are well defined and can be combined into multimodal applications, most fusion and fission relationships between devices and modalities are statically bound. This means that if one of the input widgets encounters a problem, the whole application will become unstable or even malfunction. The advantage offered by the “Interaction Cloud” is dynamic discovery and dynamic binding to different services according to the situations or environment.

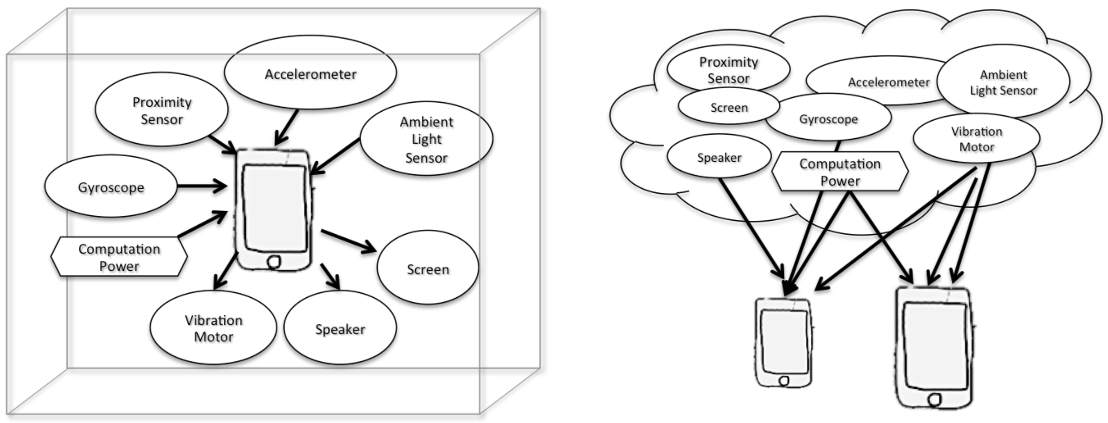


Figure 1-2 Tight Coupling of modalities on a device (Left) Loose Coupling of modalities on different devices (Right)

Figure 1-3 illustrates an example of the concept of “Interaction Cloud” over daily mobile interaction. When a user is preparing a presentation on a tablet during the travel time, he uses gestures and speech to control editing and searching. Once he arrives at the conference room, the tablet auto-discovers available services within the room devices. Connection between devices can then be established and the application can automatically select the mode of interaction. For example, the tablet can hand over the display to a large-screen projector, bind a WiiMote as a presentation controller, connect the microphone to room speakers and unbind unnecessary modalities, like touching. Applications are therefore no longer tied to the user’s device but follow the user himself. This not only enhances the user experience, but also utilizes the available resources.

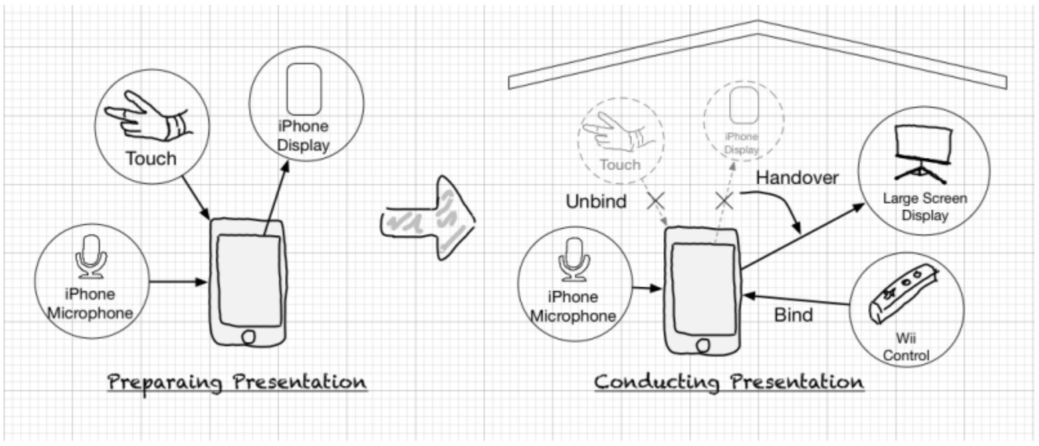


Figure 1-3 Conceptual Idea of “Interaction Cloud”

The existing research in pervasive computing and multimodal human-computer interaction leads us to believe that mobile devices will play a significant role in future human-computer interaction. Therefore, a comprehensive approach to integrate users’ mobile devices as part of a multimodal application is required.

1.1 Challenges in Multimodal Development

The first challenge in integrating mobile devices into pervasive multimodal application is that the static binding between components and the execution platform is usually limited to desktop environments. Although existing frameworks provide predefined interaction widgets, dynamic adaptation of mobile devices has been ignored. Users can only interact with the system through pre-defined commands and cannot customize the controls based on their interests or transfer tasks between devices. Besides, multimodal applications are still mainly designed for desktop environments. It seems that existing multimodal applications are self-contained and do not respond readily to changes in the environmental context. Also, end-user applications are still device-oriented, and extra effort is required to transfer data from one device to another.

We therefore propose that by bringing in the concept of cloud computing (HaaS and SaaS), applications can become user-oriented, and self-described widgets can automatically bind to each other. A middleware is therefore required to provide automatic service discovery and dynamic binding between components in order to resolve the tight coupling problem.

The second challenge is the development cycle of multimodal applications, which involves a broad spectrum of research domains. When developing multimodal applications, developers are required to study the usability of input and output modalities, communication protocols, fusion and fission algorithms and the self-description capability of hardware widgets. Therefore, the development process lies at the crossroads of several research areas including middleware, networking, software engineering, psychology and cognitive science [31]. The multidisciplinary nature of pervasive computing and multimodal interaction brings together different roles of scholars and researchers. Software engineers are interested in building tools and systems to support the development of multimodal interfaces [19], HCI engineers focus on tasks of people using the system and interaction practitioners are interested in how humans use multimodal interfaces. However, existing development artifacts pay little attention to the multidisciplinary nature of MMHCI development. Components are only classified by physical widgets rather than by the nature of the widgets.

We propose that by introducing a MVC-design pattern, components can further be decomposed into model, view or controller, which will provide more support for multidisciplinary collaboration.

The third challenge is the tight coupling between application programming interfaces (API) and the application programming sequences that decode the user's interactions or APIs that are dedicated to specific modalities such as gesture recognition [60], speech recognition [27] or combined usage of speech and gesture [35]. Human interaction with devices is usually made possible through provided services and code libraries. However, multimodal applications are normally constructed from a number of independent and heterogeneous components [54]. Existing multimodal systems [56][12][18] do not provide an extensible and flexible development environment nor a good software engineering approach for integrating large and heterogeneous number of components [36]. As a result, the semantics of the interactive data and the processing of the modality are embedded deeply within the application logic in an entangled manner, rendering it rather difficult for the application to adapt or respond to input from different modalities without explicit modification to the application code. To complicate matters, multimodal interaction involves the combination of multiple interacting modalities that often act in unison to convey a complex human-computer interaction. In a conventional application development platform, adapting existing desktop applications to support multimodal interaction would require significant amounts of programming effort that may make the solution intractable and difficult to maintain. Any change to the modality mix and interaction would require re-programming effort. Even worse, the solution is not easily portable to another platform, nor does it easily accommodate alternative devices [36].

We propose that in order to facilitate the use of multimodal application and develop environment of pervasive systems, component-based software engineering (CBSE) [36][20][8][9][52] should be used. Under such a paradigm, each component is self-contained and serves a specific function. By applying the same communication input and output interface, components can be re-combined and deployed as a new application. This approach emphasizes the concept of separation of concerns, decreases the component dependencies, increases the architectural reusability and finally reduces the production cost. Also, it provides the basis of supporting large scale of heterogeneous devices.

1.2 Contributions

To solve the problems of static binding between components, to facilitate the involvement of a broad spectrum of research domains and to decouple interaction components (applications, devices, modalities and commands) from each other, we have developed the i*Chameleon platform, which provides a complete development cycle and execution platform for multimodal applications. The development platform is targeted to facilitate developers, engineers and designers to collaborate through their respective roles in developing interaction components. The execution middleware supports the discovery and binding of deployed components and finally forms a multimodal application.

The first contribution of this thesis is the classification of interaction components. By taking the concept of separation of concerns from MVC design pattern, we classified the pervasive multimodal interface into three conceptual components: *modalities and interactive devices* (model), *interactive commands* (view) and *fusion and fission algorithms* (controller). It not only provides an isolation layer with a single point of access, but also reduces the number of connections needed and facilitates change management. As a result, a concrete role distribution can be defined at the early state of development and provides a clear structure for each component within the interaction cloud.

The second is a self-described modeling language that is defined to model modalities, devices, data and interaction. This acts as a communication channel between components. Each component is self-contained and it serves a specific purpose. This approach allows logic developers, widget engineers and interaction designers to build the system independently and plug-in to an interaction cloud.

The third is the implementation of multimodal middleware. We applied two approaches: (1) Publish / Subscribe communication paradigm and (2) Web Services. Both of them provide a mechanism for changing the binding between interaction components from static to dynamic according to the execution environment and user preferences. Publish / Subscribe communication paradigm provides a fast prototyping platform and real-time execution environment while web services offers a standardized communication protocol for interaction of components over the Internet.

In this thesis, we present a comprehensive solution to these issues. In chapter 2, we discuss the features of existing modeling languages and system followed by reviewing and comparing the strengths and weaknesses of different well-known multimodal frameworks. In chapter 3, we generalize the design principles towards dynamic binding, services discovery, components modeling and separation of concerns. In chapter 4, the classification and its description language of interaction components is presented according to the MVC design pattern. This chapter also discusses the development cycle in detail. In chapter 5, web services architecture is implemented according to the approach discussed in previous chapters while in chapter 6, the publish/subscribe communication paradigm is applied. In chapter 7, we evaluate the effort of implementing two multimodal application based on the suggested development cycle. Finally, chapter 8 presents the conclusion and suggests future research areas.

Chapter 2. Literature Review

Since the appearance of Bolt’s [7] “Put that there” demonstration in 1980 that presented the first multimodal application, researchers have been studying the topic of multimodal human computer interaction (MMHCI), in particular fusion technologies [5], signal integration and synchronization. In the meantime, new developments in sensors and processing techniques (computer vision, audio and speech), new detection and recognition algorithms and tracking theories have opened up new possibilities in the development of multimodal interaction. In recent years, there have been a number of toolkits and description languages developed for supporting prototype and integration of multimodal interactive applications [59]. In this chapter, we will review the existing approaches in two directions, multimodal modeling language and multimodal platforms.

2.1 Existing Multimodal Modeling Approach

In this section, we evaluate seven multimodal modeling languages. They are either models in UML or XML and each of them have specific aims or target specific applications or modalities. Some of the modeling languages are used for defining and capturing abstract ideas, but do not handle implementation. In the following sections, we will discuss the scope, advantages and limitations of each modeling approach.

2.1.1 Wisdom UML Extensions Modeling

Wisdom architecture is a conceptual model for interaction proposed by N.J. Nunes in 2001 [44]. It separates the concepts between internal functions and user interface. The target of this extension is to bridge the gap between software engineering and human computer interaction and define the roles of task analysis and object models in user interface design.

Figure 2-1 describes the architecture of the Wisdom model. Borrowing the concept from object-oriented software engineering (OOSE) and interaction model from human computer interface (HCI), five dimensions are defined. The analysis model is inter-connected with interaction model through the same information channel.

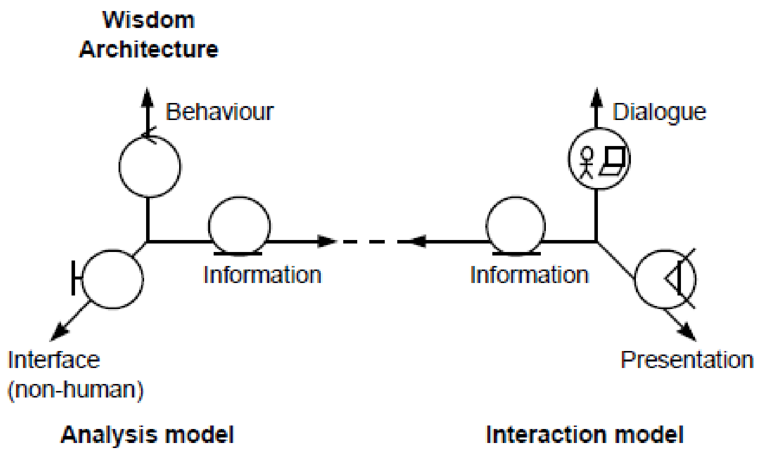


Figure 2-1 The Arch model and the Wisdom architecture

The analysis model includes three types of elements, boundary class (*Interface*), control class (*Behavior*) and entity class (*Information*). The boundary class models the interactions between the system and external systems. Control class represents the business logic. It coordinates and controls the information while the model class model perdurable information.

The Interaction model contains the interaction space class (*Presentation*) and task class (*Dialogue*). The interaction space class models the interaction between the system and users and the task class models the structure of dialogue between the user and the system.

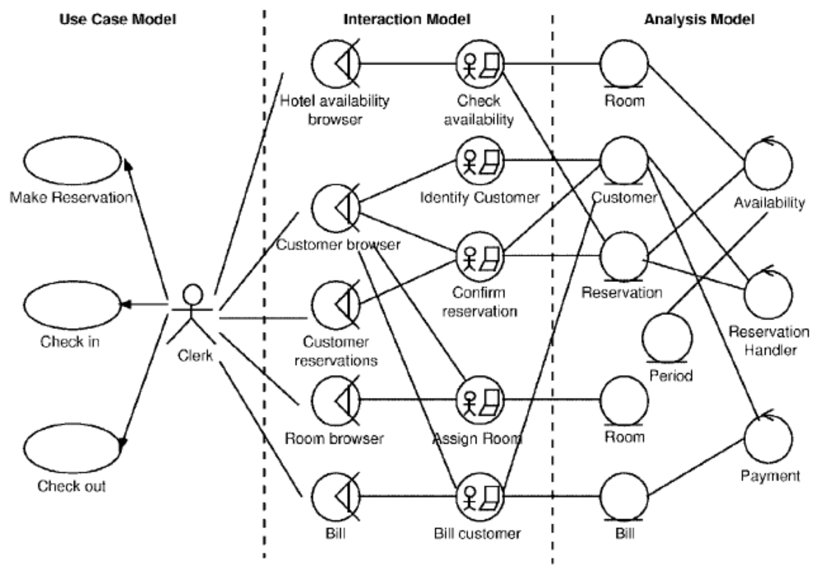


Figure 2-2 Example of a use-case model, user interface architecture and an internal architecture for an simple Hotel Reservation System

Figure 2-2 shows an example of how to model a hotel reservation system using the Wisdom UML integrated with the standard user case model. In this example, the interaction model inter-connects the human aspect with the business logic. However, it only offers the top level of abstraction, which gives an overall description of the system. It does not include the modalities concept or modeling for fusion and fission algorithms.

2.1.2 Zeljko's MMHCI Modeling

In 2004, Zeljko defined a HCI modality meta-model to describe the concept of modalities and to classify multimodal interactions. The motivation of this research is to improve the accessibility of the development of multimodal applications. The idea is that applying UML techniques would serve to facilitate the communication between software engineers and interaction designers.

Figure 2-3 shows the UML model for HCI modalities. It divides modalities into two main categories, input and output. These are further divided into event-based, stream-based, static and dynamic. Following this modeling, developers are less bound to the underlying implementation technique and they can focus on the domain related problems. Modality designers can focus on the higher level concepts, such as the division between simple modality and complex modality while engineers can focus on lower level issues like event-based modality and streaming-based modality. However, the input modality classification and modeling is based on the technical factors of hardware devices. Although this method can clearly categorize current devices, it ignores the properties and the behavior of the devices. Besides, this meta-modeling only handles the traditional video output and lacks support for new types of output manner, such as sensors and tangible objects [45].

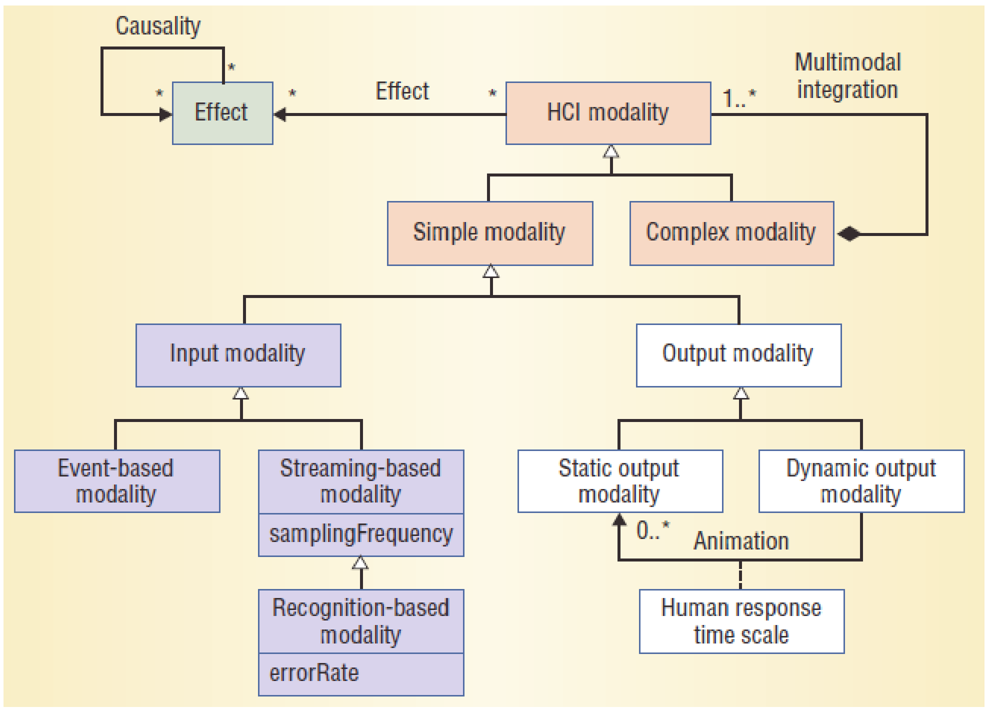


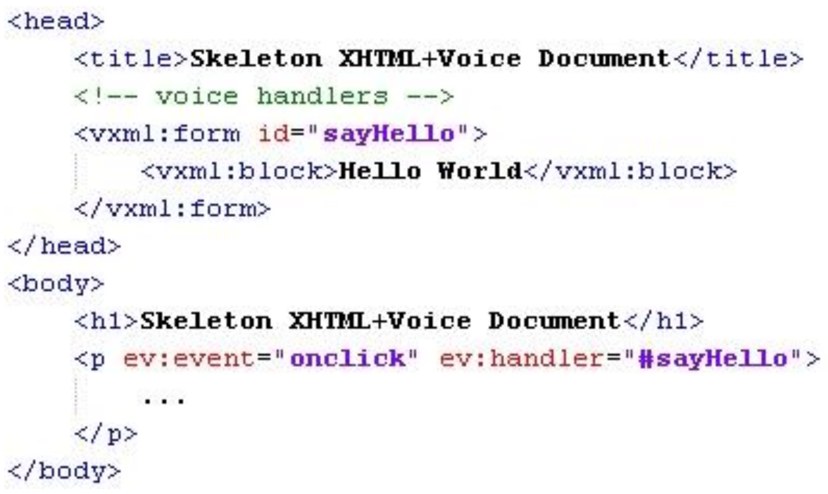
Figure 2-3 Human Computer Interface modalities model [45]

Table 2-1 UML Stereotypes for Zeljko's MMHCI modeling [45]

Simple Modality	Single modality
Complex Modality	Multimodalities combined by at least two simple modality
Input Modality	Human output
Output Modality	Output presented to human
Event-based Modality	Tokenized input signals
Streaming-based modality	Input signals with some resolution and frequency
Static Output Modality	Statically Presents Data, such as image
Dynamic Output Modality	Dynamically Presents Data, such as video

2.1.3 W3C X+V

X+V stands for XHTML and VoiceXML. This technology was finalized by W3C in 2001 [61]. It standardizes and enables the speech interaction over the World Wide Web. The profile includes voice modules that support speech synthesis, speech dialogs, command and control, speech grammars, and the ability to attach Voice handlers for responding to specific DOM events. Figure 2-4 shows an example of how to trigger the sound interface when the user clicks on a paragraph element on a standard HTML document.



```
<head>
    <title>Skeleton XHTML+Voice Document</title>
    <!-- voice handlers -->
    <vxml:form id="sayHello">
        <vxml:block>Hello World</vxml:block>
    </vxml:form>
</head>
<body>
    <h1>Skeleton XHTML+Voice Document</h1>
    <p ev:event="onclick" ev:handler="#sayHello">
        ...
    </p>
</body>
```

Figure 2-4 Example of X+V

One of the advantages of using this technique is both markup languages are proven and guarantees the response time. However, it only focuses on thin-client browsing and therefore, it is hard to implement. Also, this technique can only provide one-way interaction, is not extensible to other modalities and is limited to the Web interface.

2.1.4 Multimodal Markup Language

In 2001, Roessler introduced the application of the Multimodal Markup Language to mobile environments with two input modalities and two output modalities. It is another XML-based modeling language which provides the benefits of a clear separation in content, structure and interaction [30]. It models speech and handwriting as input with graphical and anthropomorphic avatars as output.

This markup language is similar to X + V which also applied the multimodal concept by combining advantages from HTML and VoiceXML. HTML offers authors the possibility to publish documents on graphical devices while VoiceXML allows web-based development and content delivery to interaction voice response systems. Table 2-2 shows the core elements of MML.

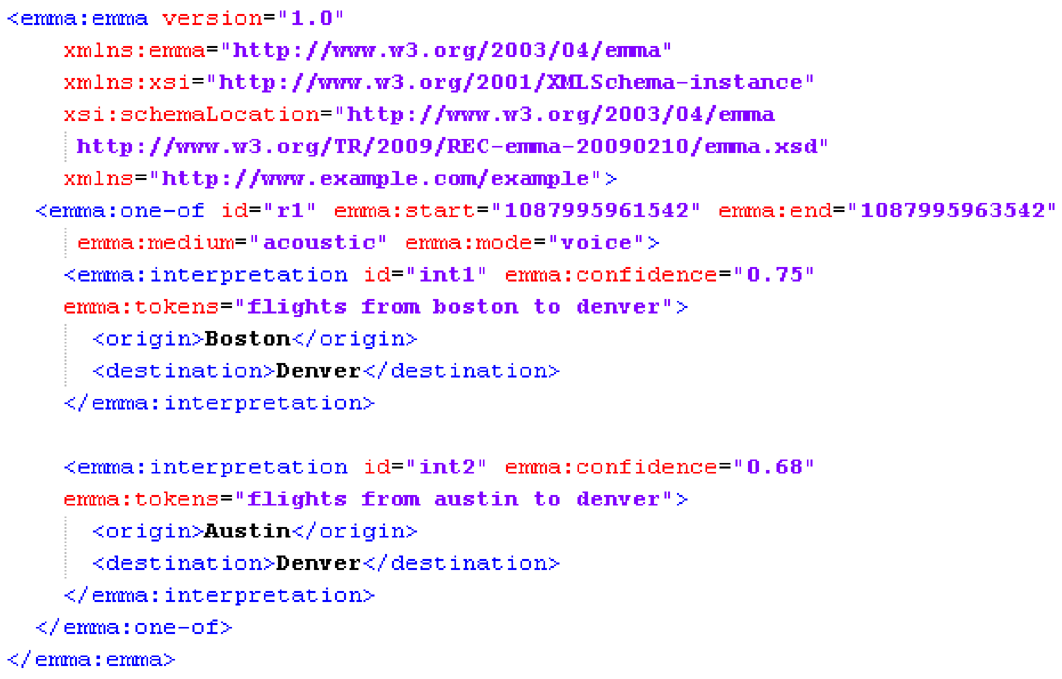
Table 2-2 Specification of MML key element [30]

modalityOut	Define the presentation modalities to user.
modalityIn	Define the use of input modalities including their analyze algorithms. For example, grammars for speech recognition or hand-writing.
production	Control the output elements format.
timing	Specific he synchronization information.
bargeIn	Enables or disables interruption of voice announcements.
initiative	Specific the control flow.

However, MML does not provide extensibility for new upcoming modalities. It also does not provide the clear separation for different scholars and researchers for involving into its development process.

2.1.5 Extensible Multimodal Annotation

Extensible Multimodal Annotation (EMMA) is a well-known multimodal language developed by W3C and the first version was published in 2009. It offers a sufficient syntax for defining multimodal interaction for web applications. The purpose of EMMA is to represent content which is automatically extracted from a user's input by an interpretation component, where input is to be taken in the general sense of a meaningful user input in any modality supported by the platform [22]. Figure 2-5 shows a simple multimodal application which uses voice recognition as an input modality. The XML represents a flight reservation application. Once the engine matches with the token defined in the interpretation element, it will load the corresponding defined data.



```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:one-of id="r1" emma:start="1087995961542" emma:end="1087995963542"
     emma:medium="acoustic" emma:mode="voice">
    <emma:interpretation id="int1" emma:confidence="0.75"
    emma:tokens="flights from boston to denver">
      <origin>Boston</origin>
      <destination>Denver</destination>
    </emma:interpretation>

    <emma:interpretation id="int2" emma:confidence="0.68"
    emma:tokens="flights from austin to denver">
      <origin>Austin</origin>
      <destination>Denver</destination>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

Figure 2-5 EMMA sample markup language [22]

In Figure 2-6, Roman [22] implemented a multimodal mobile application using the EMMA markup language. After semantic interpretation has been performed by different engines, EMMA handles the communication protocol between different widgets and the integration processor.

Although EMMA is a well-defined interaction language, it still poses a high barrier for non-experienced users, as it requires much technical knowledge to create a multimodal application. The application developers are required to do the hardware integration, interaction design and modality fusion. EMMA also focuses less on the standardization of semantic representation of input modalities [32], which means that future modalities cannot be adapted to the language without major changes. EMMA therefore targets on serving the input modalities in a web interface, however, not much attention is paid on the output modalities and non-web applications.

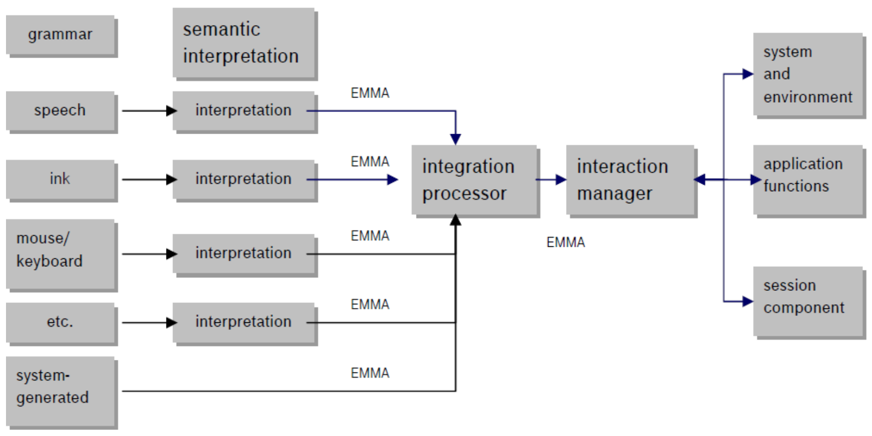


Figure 2-6 Generation of a consistent system task from multiple input channels using EMMA [22]

2.1.6 Multimodal Presentation Markup Language

Multimodal Presentation Markup Language (MPML) is another XML-based modeling language that enables dialogue-based interactions for multimodal user interface that was designed by Ishizuka Laboratories [6] in 1999. Using MPML, developers can develop and deploy HTML-based multimodal presentation application quickly. Figure 2-7 shows a sample application, which performs text-to-speech together with HTML switching. Developers are only required to create suitable agents to handle input and output modalities. For example, users can define an autonomous component that processes the user's speech input as the "Voice Recognition Agent". However, in MPML 3.0, only three agents are supported and it is hard to integrate new agents into the MPML engine, therefore, extensibility is limited. Also, interaction logic and presentation information are mixed together in one XML file, which requires software engineers and interaction designers to understand each other's work.

Meta-model is also not defined in MPML, which only focuses on daily presentation, rather than the differences between generic user interface and multimodal user interface.

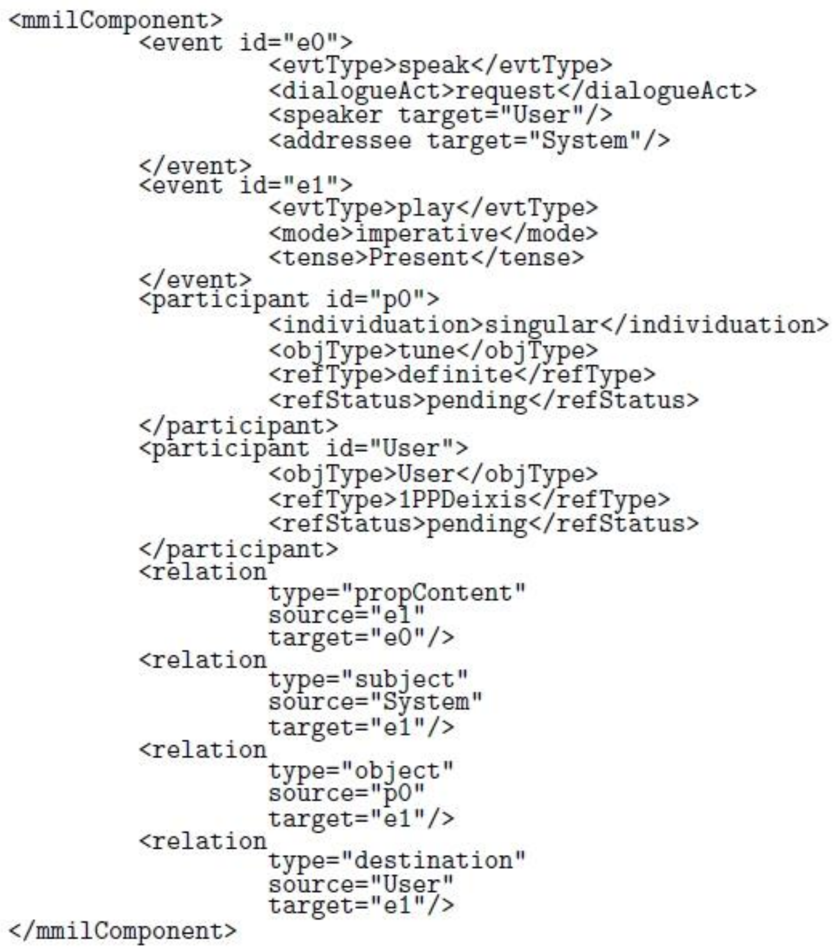
```
<mpml>
  <head>
    <agent id="Genie" character="Genie" system="MSAgent"
        x="800" y="490" agreeableness="100" activity="100"/>
  </head>
  <body>
    <seq>
      <scene agents="Genie">
        <page ref="London-main.html">
          <speak agent="Genie">
            Welcome to London! I am Bruce,
            your tour guide around here.
            <NB pause="500"/>
          </speak>
          <act agent="Genie" act="Announce"/>
          <speak agent="Genie">
            You are currently standing at the east end
            of Covent Garden. Have you ever been here
            before?
          </speak>
        </page>
      </scene>
    </seq>
  </body>
</mpml>
```

Figure 2-7 Sample of MPML

2.1.7 Multimodal Interface Language

Multimodal Interface Language (MMIL), introduced by Romary and Kumar, is yet another type of XML-based multimodal language. It models the interaction between the user and the dialogue as well as the interactions within the system.

The target of MIML is general purpose; therefore, Kumar defined a meta-model, represented in UML. Based on the meta-model, he abstracted six types of information streams, such as word and phoneme lattice, dependency representation forest, dependency representation, word/phoneme sequence, visual-haptic semantic representation, and graphic-haptic layout [32], and translated the salient concepts for XML support. It demonstrates the importance of applying UML meta-model technique in software engineering.



```
<mmilComponent>
        <event id="e0">
                <evtType>speak</evtType>
                <dialogueAct>request</dialogueAct>
                <speaker target="User"/>
                <addressee target="System"/>
        </event>
        <event id="e1">
                <evtType>play</evtType>
                <mode>imperative</mode>
                <tense>Present</tense>
        </event>
        <participant id="p0">
                <individuation>singular</individuation>
                <objType>tune</objType>
                <refType>definite</refType>
                <refStatus>pending</refStatus>
        </participant>
        <participant id="User">
                <objType>User</objType>
                <refType>1PPDeixis</refType>
                <refStatus>pending</refStatus>
        </participant>
        <relation
                type="propContent"
                source="e1"
                target="e0"/>
        <relation
                type="subject"
                source="System"
                target="e1"/>
        <relation
                type="object"
                source="p0"
                target="e1"/>
        <relation
                type="destination"
                source="User"
                target="e1"/>
</mmilComponent>
```

Figure 2-8 Example of MMIL definition

2.2 Existing Multimodal Platform

After the discussion of multimodal modeling languages, existing execution platforms will be evaluated. Multimodal platforms are used to retrieve different input modalities and triggers corresponding output modalities using suitable fission and fusion algorithms. In this section, we evaluate four multimodal platforms and discuss their scope, advantages and limitation.

2.2.1 QuickSet

QuickSet is developed in 1997 and it is a pen and voice based multimodal system for configuring military simulations based on LeatherNet [11], a system used in training platoon leaders and commanders at the US Marine Corps (USMC) based [13]. It runs on a hand-held PC and communicates through a distributed agent architecture based on the Open Agent Architecture (OAA). It requires a central facilitator to process and coordinate different agents, and to route queries to appropriate agents.

Figure 2-9 shows the architecture of QuickSet. Each agent represents a specific function. For example, QuickSet interface provides the geo-referenced map. Speech recognition agents capture the voice command and pass it to the natural language agent to process. Gesture recognition agent recognizes all pen input from a PC screen or tablet. Multimodal integration agent accepts structured meaning representations from other agents, such as gesture agent or speech agent, and produces a unified multimodal interpretation. CommandVu agent is yet another input modality that resides on top of the gesture recognition agent, this enables the user to ask "CommandVu, fly me to this platoon <gesture on the map>".

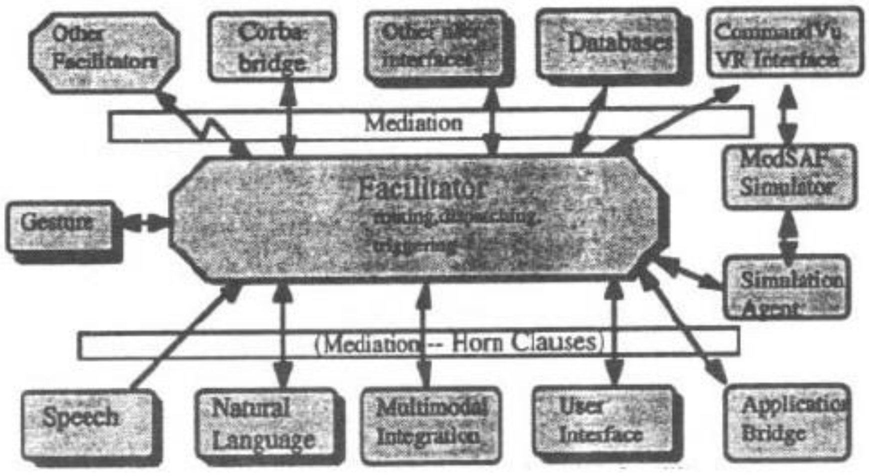


Figure 2-9 Architecture of QuickSet [12]

Holding QuickSet in hand, the user views a map from the ModSAF simulation, and with spoken language coupled with pen gestures, issues commands to ModSAF. In otter to create a unit in QuickSet, the user would hold the pen at the desired location and utter (for instance): "led T72 platoon" resulting in a new platoon of the specified type being created at that location.

However, QuickSet does not involve any modeling technique and is customized for specific application with specific hardware devices. The extensibility is limited. If a developer would like to adapt a new agent into the system, he is required to implement it using the OAA communication protocol and integrate the new logic and rules into the facilitators. If the new agents are required to communicate with other agents, those agents also need to be customized. Interactions between system and human cannot be customized. The fusion algorithms are strongly coupled with the raw signal as well as the agent design. It lowers the flexibility of the command design.

2.2.2 Krahnstoever's Framework

Krahnstoever's framework proposed a method for combining audio and visual interaction with a large screen display [35]. It includes three main components: interaction session, visual and audio components and modality fusion.

One of the major characteristics of this framework is that it includes an initialization and termination phase in the interaction session which normal multimodal applications do not consider. During the initialization phase the interaction dialogue between a new user and the system is established. It is followed by the interaction phase, where the actual communication between the user and the system takes place. Finally, the termination phase is entered when the user (or the system) decides to conclude the dialogue. The state transition model is shown in Figure 2-10.

Visual and audio components are responsible for signal processing, which includes face detection, palm detection, head and hand tracking, speech command recognition and audio feedback. The processed data will be interpreted by another modality fusion component.

In Krahnstoever's platform, the research team focuses on specific input (Speech, Hand Gesture, Head Tracking and Face Detection) and output modalities (Display and TTS). Therefore, the architecture is not designed to flexibly adapt to new modalities. Also, it does not include modality-modeling concept, which limits the communication between different roles of user during the development phase.

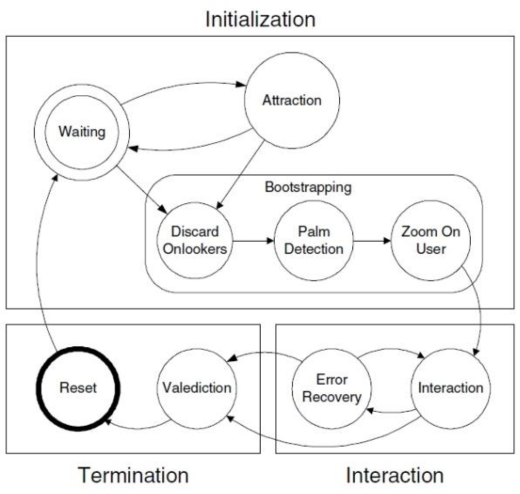


Figure 2-10 State transition model of an interaction session between a user and the system [35]

2.2.3 ICARE

ICARE stands for Interaction-CARE (Complementarily Assignment Redundancy Equivalence). This platform applies the concept of component-based approach for specifying and developing multimodal interfaces [9]. This approach included two types of components: elementary components and composition components. Elementary components are dedicated to interaction modalities, which define the combination of physical device *d* and interaction language *L* <*d*, *L*>. For example, a speech input is described as the coupling of microphone and pseudo natural language. Therefore, the modality definition is dependent on the fusion algorithms as well as the hardware devices. Besides, the fusion conditions are defined by composition components, which include complementarity, redundancy, equivalence and redundancy/equivalence [8].

Complementarity combines all triggered events within a short period of time. When more than one modality conveys redundant pieces of data that were generated at approximately the same time and have the same output, one of the events will be ignored if those modalities are connected by redundancy component. Redundancy/equivalence component is a mix of redundancy and equivalence [20].

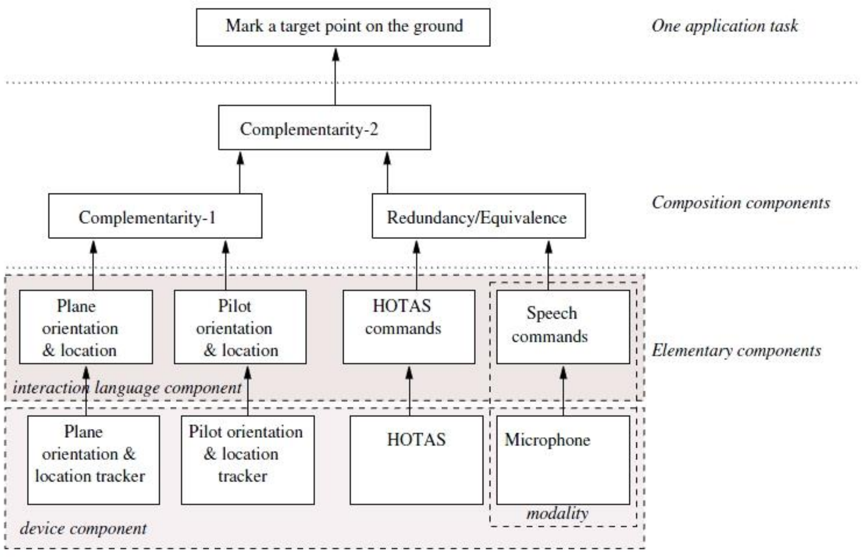

Mark a target point on the ground
One application task
Complementarity-2
Complementarity-1
Redundancy/Equivalence
Composition components
Plane orientation & location
Pilot orientation & location
HOTAS commands
Speech commands
Elementary components
interaction language component
Plane orientation & location tracker
Pilot orientation & location tracker
HOTAS
Microphone
modality
device component

Figure 2-11 ICARE Specification for French military aircraft cockpit [20]

As shown in Figure 2-12, the architecture of ICARE platform requires combining an outside dialog controller to listen to its triggered events. An ICARE component chain defines a pipeline from user's actions to commands or elementary tasks which are useful for the Dialog Controller. The communication protocol is implemented with direct call of methods or by using TCP/IP, UDP, JavaRMI. ICARE enables the designers to graphically manipulate and assemble software components to customize the multimodal application. From the customized result, the code is automatically generated.

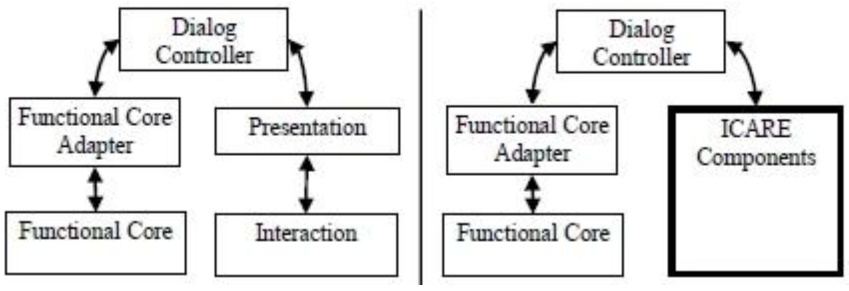


Figure 2-12 The ARCH software architectural model and ICARE components within an ARCH software architecture [9]

Although ICARE provides a graphical user interface for users to customize the fusion logic by establishing the execution condition, user cannot develop flow control, such as looping or store the state of the system. For example, if the user wants to customize a command that is based on previous executed commands, ICARE will not be able to support this. Also, modalities are highly dependent on the physical devices and the underlying algorithms. Therefore, ICARE can only provide a high level of reusability but not at the implementation level. It also does not provide the meta-modeling of the components and there is no concept of separation of concerns.

2.2.4 OpenInterface

OpenInterface is a well-known multimodal application development tool which supports the design and effective implementation based on off-the-shelf heterogeneous components [52]. The motivation of this platform is to provide an evolvable solution implementing a device-independent and interaction language that is technique independent.

OpenInterface is manipulated by components and pipelines. A component is defined as any software or hardware unit that provides a service. Components can do various tasks, ranging from being a driver for input devices, signal processing, networking, produce graphics, etc. They are reusable and independent from other components. It consists of four basic attributes: it must include an API and installation package, it must be documented and self-contained. The access points where data can come inside or outside the components are called pins. There are three types of pins, they are sink pins (receiving data), source pins (receiving and sending data) and callback pins (sending events) [36]. The component will be translated to XML-based component interface description language (CIDL) in order to communicate between the OpenInterface kernel and component driver.

Figure 2-13 discusses an example of defining a mouse component. Within DirectXMouse component, it receives data from two events (startMouseCapture and stopMouseCapture) and mdriver is responsible for interpreting the information and pushing another event (pushed_events) out to other components.

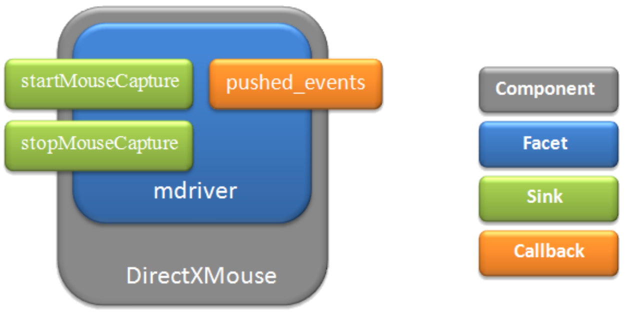


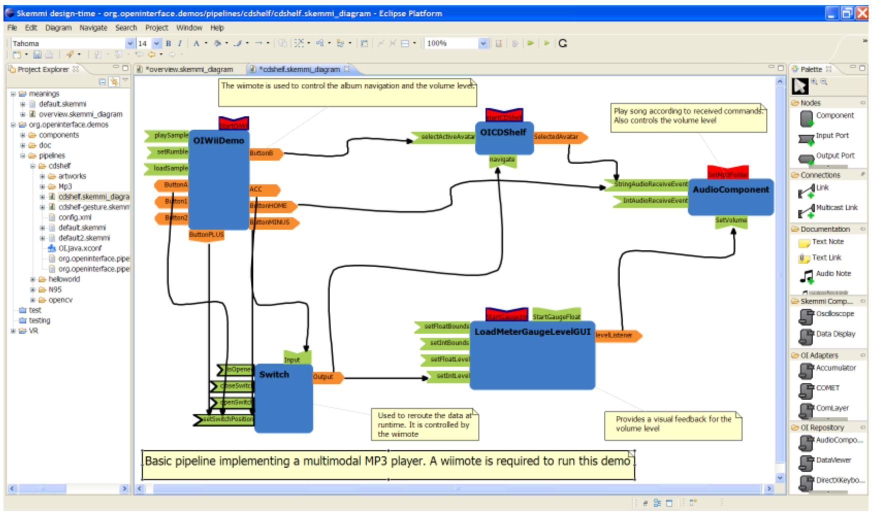
Figure 2-13 Example of a mouse component [15]

The second core object is called pipeline. A pipeline connects different components together in order to manage an advanced task. Components can be composed in pipelines by connecting together sinks and sources of various components. In this way, data produced by callback pins of some component will be sent to sink pins of another component, letting the components exchange data and events. By using this mechanism, one can create complex application pipelines to perform advanced interaction tasks or techniques [15].

In runtime environments, components together with the pipelines would be deployed to the OpenInterface Kernel. The kernel will initialize the components and establish the communication path between them. It unifies components implemented in various technologies (C/C++, Java, Matlab, C#, Python) and follows a dataflow pipe-and-filter architectural style in order to support easy reconfiguration.

Using a component builder called SKEMMI, developers can do the mapping and configuration of the pipeline. This tool provides a graphical user interface (Figure 2-14) to designers to manipulate the component and connect them before deploying to the kernel.

The goal of OpenInterface is to bridge the gap between the design and implementation process of multimodal interactions. Therefore, it introduces the concept of separation of concerns. Different types of users, such as programmers and designers, can contribute their own components in order to develop the final application.



The wiimote is used to control the album navigation and the volume level

Play song according to received commands. Also controls the volume level

Used to reroute the data at runtime. It is controlled by the wiimote

Provides a visual feedback for the volume level

Basic pipeline implementing a multimodal MP3 player. A wiimote is required to run this demo

Figure 2-14 SKEMMI Graphical Editor

Programmers: they can freely use the OpenInterface – CIDL – Components Interface Description Language to share code source and avoid redundancies in common tasks: data format support, usual audio, video processing, etc.

Application Designers: (AD): with minimal efforts, they can build multimodal pipelines using the OpenInterface Design and Development Environment SKEMMI.

End-users: The final OpenInterface multimodal interface provides a natural interaction between the human and the physical or virtual environment.

OpenInterface is a well-developed tool for the development of multimodal application. However, it still requires an experienced designer, with technical knowledge, to customize the application. The barrier is still relatively high when compared with other GUI programming toolkit, such as Alice [16] or Scratch [39]. OpenInterface also does not provide any modeling language and definition language for modalities, devices or data structure.

2.2.5 HCI^2 workbench

HCI^2 workbench [54] is a Publish / Subscribe software platform that aims to simplify the development process of a multimodal application by using a modular programming technique. It provides a graphical environment to support the development of a typical MMHCI system, for example, debugging, module packaging, module management, system configuration and testing. However, each module must be developed with the multi-disciplinary knowledge and all the commands must be defined and configured before execution. It only supports runtime structural changes, such as registration of modules but cannot dynamically change or bind packages during runtime. Besides, it does not provide any conceptual modeling of multimodal interaction.

Chapter 3. Design Guidelines

Owing to the growth of interest in multimodal applications [48], the process of developing such platforms is becoming increasingly important. In order to facilitate the design of multimodal systems from the ground up, design principles and interaction paradigms are necessary. Based on the discussion in chapter 2, different multimodal modeling languages and platforms are presented. For illustrative purposes, the first section presents a comparison and summarization of six modeling languages and four frameworks. Then the design principles on modeling languages and framework design are summarized. Finally, the features of modeling languages and multimodal platform will be discussed.

3.1 Comparison of Multimodal Modeling Languages and Multimodal Platforms

Table 3-1 and Table 3-2 summarize the system conceptual and modalities model for several state-of-the-art frameworks. The tables show that in general, current frameworks usually either focus on the system conceptual model, or the modality model. Due to the multidisciplinary nature, however, a complete MMHCI system needs to include concerns at both the system level and modalities level. Therefore, in order to create a complete multimodal human-computer interaction application, it is necessary to work with different frameworks/platforms. This is suboptimal as it increases the challenges of developing multimodal applications. In addition, within the system model, most modeling languages only address the workflow modeling, ignoring the context model and dataflow model. We therefore propose a single system that seeks to improve the communication between designers and engineers (system level modeling) and engineers and developers (modality level modeling).

Another observation from Table 3-3 is that existing frameworks are often specific to particular input modalities or output modalities. Examples are speech or gestural input. One of the reasons is that often, frameworks are designed and developed to meet specific requirements, and therefore are required to be highly functional and accurate. This, however, results in low flexibility and extensibility. This drawback has been noted in the reviews of ICARE and OpenInterface from chapter 2, which raise concerns regarding the extensibility of multimodal applications.

The extensibility issue has been addressed to a certain point through the components-based software engineering (CBSE) approach. Each modality, algorithm or hardware driver is regarded as a component. In OpenInterface, for example, developers can apply the pipeline concept to connect different components to build a workflow sequence. This minimizes the effort of developers and maximizes the flexibility and reusability of multimodal applications.

Secondly, most existing frameworks still focus on the desktop environment, which limits the portability to other platforms. With the tremendous increase in mobile devices, multimodal platforms need to support distributed development. Multimodal interfaces should adapt to the needs and abilities of different users, as well as different contexts of use. An example might be to allow gestural controls to replace voice input in noisy environments. Therefore, distributed sensory networks and dynamic adaptation among different input / output devices are needed.

Another challenge we discussed in Chapter 1 is the development cycle of multimodal applications. This usually involves a broad spectrum of research domains. Although existing platforms involve the principle of separation of concerns, they still require experienced programmers or developers to customize the interactive commands. This creates a high barrier for new developers.

3.2 Design Principles

Given the challenges discussed in Chapter One and the limitations above, adapting, integrating or extending an interaction modal often requires complex recoding and maintenance. Contemporary markup or modeling languages ignore the multidisciplinary factors and assume a certain level of knowledge sharing between software engineers, interaction designers and fusion engineers. In a real world situation, interaction designers often come from the psychology field while software engineers are computer scientists. They are trained to address different issues pertaining to human computer interactions and are often not aware of each others' concerns or issues, which makes the knowledge sharing assumption invalid.

Tables 3-1, 3-2 and 3-3 thus highlight the desirable principles that enhance the design process for these heterogeneous parties.

Table 3-1 Different multimodal modeling language and their capabilities (a)

	System Level Modeling			**Modality Level Modeling**			**Interaction Language**
	Context Model	**Behavioral Model (Dataflow)**	**Behavioral Model (Workflow)**	**Input Modalities**	**Output Modalities**	**Multi-modalities**	
Wisdom UML Extension	Lack of describing Environment factor Detail distinguish the system by Analysis Model and Interaction Model	Standardize by Information Dimension	Boundary class (System to System) Presentation (System to Human)	No	No	No	UML (integrate with existing User Case Diagram)
Zeljko's MMHCI Modeling	No	No	No	Abstracted interface (1) Event-based (2) Streaming-based	Abstracted interface (1) Static video (2) Dynamic video	Complex modality is defined as more than one input / output modality.	UML
MMIL	No	No	Using relation tag to connect source event and target event.	No	Detailed Speech Modeling (Grammars, sentences structure, etc.)	No	Meta-model with UML and configuration in XML
MPML	No	No	Only support simple one to one request and response event.	No	No	No	XML

Table 3-2 Different multimodal modeling language and their capabilities (b)

	System Level Modeling			Modality Level Modeling			Interaction Language
	Context Model	**Behavioral Model (Dataflow)**	**Behavioral Model (Workflow)**	**Input Modalities**	**Output Modalities**	**Multi-modalities**	
EMMA	No	Token can be defined and passed to other command by emma:token tag	Define event based listener in XML, therefore, each input event will map to a command	No	Detailed Speech Modeling (Grammars, sentences structure, etc.)	No	XML
MML	No	No	Using modalityOut tag to define the output modalities Using modalityIn tag to link with suitable algorithm Using initiative tag to define the control flow	No	No	No	Meta-model with UML and configuration in XML
X+V	No	Follow W3C HTML Protocol No	Follow W3C HTML Protocol	No	Voice	No	XML

Table 3-3 Comparison between different existing multimodal platform

	Platform / Execution Environment	**Supporting Modalities**	**Modalities Adaption and Reusability**	**Communication Protocol**	**Hardware Abstraction Layer**	**Application Command Layer**	**Fusion Algorithm Integration and Customization**	**Interaction Language**	**Separation of Concerns**
QuickSet	Mobile Desktop	*Input:* Speech, Hand Gesture *Output:* Voice, Visual Display	No	OAA	No	No	No	Hard-Coded. Required additional programmin g effort	No. Only design for experienced developers
Krahnstoever's Framework	Desktop	*Input:* Keyboard, Mouse, Palm, Head Tracking, Hand Tracking, Speech *Output:* Voice, Visual Display	No	Direct API Call	No	No	No	Hard-Coded. Required additional programmin g effort	No. Only design for experienced developers
ICARE	Desktop VR Aircraft Maintenance Training	*Input:* Speech, Hand Gesture, Keyboard, Mouse, Sensor *Output:* Handle by other application	No	TCP, UDP, JavaRMI	No	No	Limited. Only basic flow control	Self-defined file with GUI Editor (ICARE Tool)	Programmers and experienced Developers
OpenInterface	Desktop	*Input:* Wii (Head Tracking), Keyboard, etc *Output:* AR, Visual Display	Supported. CIDL & Components	OSC, TCP, UDP, Multicast	Supported. CIDL & Components	Supported. CIDL & Components	Supported. CIDL & Pipeline	CIDL with own GUI Editor (SKEMMI)	Programmers and experienced Developers

(1) Be composed by heterogeneous components. One key theme in pervasive multimodal environments is the integration of different components into a single system. Under this model, heterogeneous components can decouple unnecessary dependencies, and different inputs from different modalities are related but not necessarily integrated. Such a framework would be required to support a wide range of input and output widgets that are discoverable and interoperable across heterogeneous devices.

(2) Components need to be self-described and self-contained. The use of plug-and-play pervasive computing devices makes pervasive multimodal applications more reusable and thus lowers the cost of implementation. To support multi-level and cooperative design, data definition, algorithms and modalities should be componentized as a self-described object that is reusable and discoverable through standardized communication protocols. Dynamic adaptability of new components, low level of interruption, low system down time, and transfer of signals in standard formats are required.

(3) Be integrated with large amount of widgets. The pervasive multimodal system should be able to support large number of widgets of the same modality and/or different modalities, to properly support distributed components and user collaboration. Also, the modality output of one system can become another system's input in a cascaded architecture.

(4) Be extensible and Flexible. Regional factors, time constraints and personal preferences are some parameters that affect the way users interact with applications. The system should allow developers to create the best-fit implementation. It should also be domain-independent, minimizing the changes required to adapt to a new domain.

(5) Components are classified based on the roles of users. The development process involves contributions from different experts. One could identify four major roles when building a multimodal application: device engineers, modality designers,

interaction designers, fusion engineers. Yet the use of most interaction markup languages often results in the entangling of program logic and presentation logic.

(6) Ease of Use. The strong demand on multiple sets of technical skills poses a major difficulty in developing multimodal applications. Developers need to possess good hardware knowledge and theory of cognitive psychology with creativity [47]. However, most psychologists are not trained or equipped with skill sets to program, limiting their creativities. The interaction language must be modeled at a higher-level of abstraction. Code generation from high level modeling language of different components is required.

(7) Dynamic adaption of components. In pervasive computing, processing power and devices are everywhere. Multimodal interactive components must bind to suitable input or output and detach unnecessary components according to the surroundings context.

3.3 Features of i*Chameleon Multimodal Platform

Based on the discussed design principles, we apply different aspects of software engineering techniques to ensure the fulfillment of the principles. Table 3-4 provides a summary. In general, we first need to consider the conceptual system modeling in order to provide the abstracted system overview to different stakeholders, such as device engineers, fusion developers, interaction designers and modality designers. Modeling the modalities concept would also provide the structure of the interaction. This not only narrows the gap between the interaction designers and the hardware developers, but also provides a high-level of reusability to the system.

Multimodal modeling includes three aspects – conceptual system modeling, modalities concept modeling and modeling language:

1. Conceptual system modeling - the overview of the execution environment,
2. Modalities concept modeling –the behaviors and nature of particular input or output modality
3. Modeling language – the representation method for above modeling.

Conceptual system modeling represents the functions and parties involved in a clear manner. This offers an effective communication channel between the engineers and designers. Within this model, we should not take into consideration design or implementation criteria but focus on the required elements and workflow of the system. We can divide this model into two sections: context modeling and behavioral modeling.

The context model distinguishes between the system scope and system environment. Behavioral models are used to describe the overall behavior of the system. This model can be divided into data-flow models, which model the data processing in the system, and state machine models, which model how the system reacts to events.

Modalities concept modeling defines a meta-model for modeling input and output modalities. It provides a standard structure for unified and generic modality concept that is much less bound to the underlying implementation technology and much closer to the problem domain than conventional programming environments [45]. Therefore, this modeling is independent of the conceptual system modeling.

Modeling language describes the human-computer interaction, integrated with the rich modeling semantics of UML or XML. This language syntactically models the modalities concept, captures the requirements and expresses solutions for the design. This allows us to apply third-party interpreters to interpret the concept and even generate the underlying code.

Besides the modeling language, techniques from component-based software engineering are applied to offer self-described and self-contained heterogeneity components. In i*Chameleon, we apply the model-view-controller (MVC) design principle on component implementation. The MVC design pattern can help to classify the components according to their nature by separating user interaction, data representation and logic. Finally, device engineers, modality designers, signal processing and fusion engineers (fusion engineers) and interaction designers are assigned well-defined roles. More detail about the MVC components will be discussed in the next chapter.

In order to support the dynamic adaption of components, the i*Chameleon platform is implemented separately using a publish/subscribe approach and web services approach. Web services provide distance interaction and binding between components, and provide the advantage of well-developed technologies and guidelines. Developers can deploy their components to the web services server based on the existing infrastructure.

The publish/subscribe communication paradigm is another option for the i*Chameleon kernel. It supports the plug-and-play of different MVC-based components to the i*Chameleon kernel. The components can register their interest in an event or publish their events to a defined interface. Mechanisms and rules are also defined to provide dynamic binding to other components. The details of web services and publish/subscribe communication paradigm will be discussed in chapter 5 and chapter 6 respectively.

Table 3-4 Techniques applied to fulfill the multimodal interface design principles

	Metamodeling and Multimodal Description Language (XML)	Separation of Concern (MVC)	Dynamic Binding Between Modalities (Pub/Sub or WebServices)
Heterogeneity		Y	Y
Self-described and self-contained	Y		Y
Components integration	Y	Y	
Extensible and Flexible	Y		
Components Classification		Y	
Ease of use	Y	Y	
Dynamic adaption of components			Y

Chapter 4. MVC-based Development Cycle

In 2009, Dumas generalized the common practice of designing a multimodal system [19]. He suggested that a multimodal system should contain a fusion engine, a fission module, a dialog manager and a context manager in order to interpret the incoming modalities and process the output modalities, and signals should be processed through four phases of undergo, perception, interpretation, computation and action.

Figure 4-1.illustrates the process. Different recognizers are required to process input modalities and the results are output to the fusion engine, which is in charge of giving a common interpretation to the inputs. After the dialog management helps to identify the state, the modalities are handled by the fission engine, which triggers different output modalities.

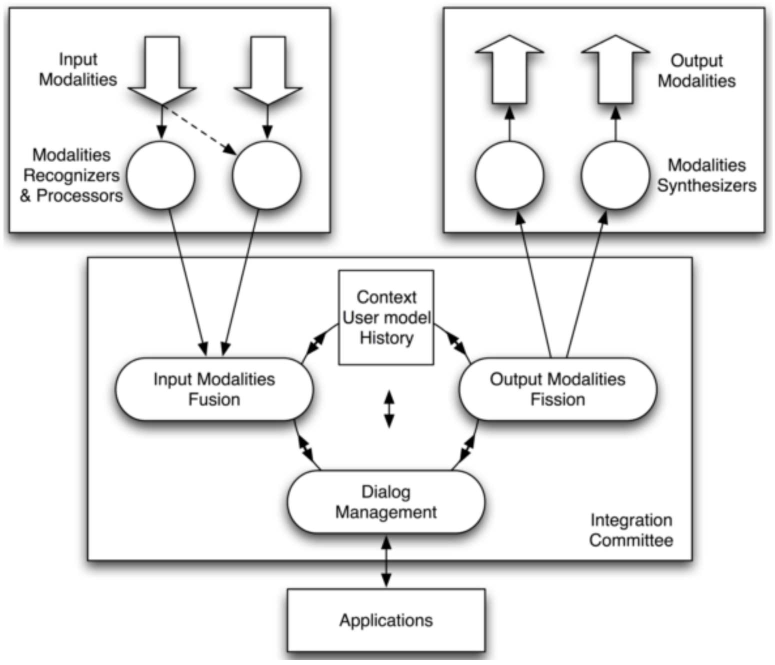


Figure 4-1 Suggested architecture of a multimodal system, with the integration committee and its major software components. [19]

These four phases are independent and can be modeled by the model-view-controller architectural pattern. This architectural pattern separates the design space along three aspects, with each focusing on a specific area and role in the overall design:

1. The *Model* defines the behavior and nature of the input and output modalities. It acts as a standardized protocol for facilitating the communication among the system.
2. The *View* serves as a presentation layer and models the encapsulation of abstract devices. It handles the communication channel from the multimodal system to the output devices.
3. The *Controller* is responsible for retrieving the input data from the input devices, analyzing them, translating them into modalities (*Model*) and triggering the commands (*View*).

Thus, the *Model* encapsulates the collection of functional methods for multimodal system, and manages the definition and behaviors of data definition, modalities and fusion workflow, including the interpretation of the description of the nature of the modalities. The *View* is defined as the part that directly interacts with the users: in other words: the action. The computation and perception handling the analyzing process is then defined as *Controller*.

The i*Chameleon platform adopts the MVC design pattern, which allows developers to develop and deploy interaction components and create multimodal applications, which end-users can execute and customize based on their profile.

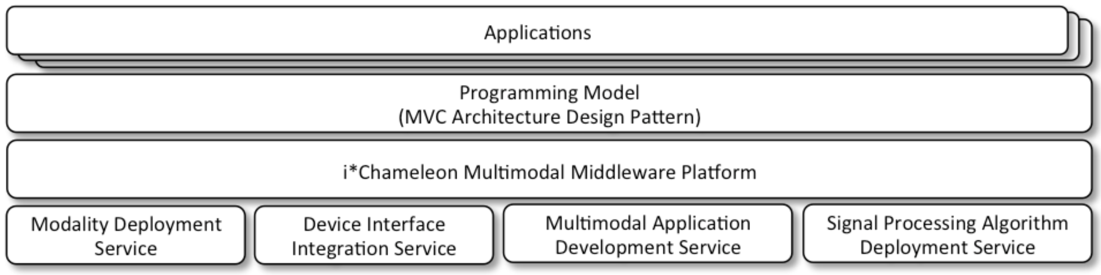

Applications
Programming Model
(MVC Architecture Design Pattern)
i*Chameleon Multimodal Middleware Platform
Modality Deployment Service
Device Interface Integration Service
Multimodal Application Development Service
Signal Processing Algorithm Deployment Service

Figure 4-2 The System Architecture of i*Chameleon platform, with the services provided for different stakeholders to develop or integrate different components to the system

Figure 4-2 shows the system architecture of the i*Chameleon platform. Four services are provided to facilitate the development process:

(i) the device interface integration service allows device engineers to integrate input and output interaction devices to the middleware;

(ii) the modality deployment service offers modality designers the possibility of defining input and output modalities at different abstraction levels;

(iii) the signal processing algorithm deployment service allows programmers to implement algorithms for translating incoming signals to outgoing modalities, and

(iv) the multimodal application development service allows non-expert users to associate different modalities with each other to customize the final multimodal application.

On top of the services, the i*Chameleon middleware coordinates different interaction components. Components will be automatically bound together according to the contextual environment and user profile in order to select the most suitable components. For example, if the user watching a movie on a smartphone walks into a home theatre with a larger screen, i*Chameleon will unbind the existing output screen and transfer the output signal to the large screen. The middleware is implemented with web services (discussed in chapter 5) and publish / subscribe communication paradigm (discussed in chapter 6).

In the rest of this chapter, we will first discuss the process of developing a multimodal application under the i*Chameleon platform. We will describe four different aspects of the development process, as shown in Figure 4-2. We will then describe the classification of the interaction components, according to the MVC (Model-View-Controller) paradigm.

4.1 Development Process

In multimodal interaction, broad spectrums of research domains are involved in the development process. The development process lies at the crossroads of several research areas including middleware, networking, software engineering, psychology and cognitive science [31], and the multidisciplinary nature of pervasive computing and multimodal interaction brings different roles of researchers and developers into this domain. As a result, the complexity of developing a multimodal application increases. For example, developers are required to be knowledgeable on diverse topics such as the usability of input and output modalities, communication protocols, fusion algorithms and self-description capability of hardware widgets. This brings together a wide range of expertise: Software engineers are interested in building tools and systems to support the development of multimodal interfaces [19], while interaction designers focus on tasks of people using the multimodal interface and algorithm developers implement fusion techniques.

Bouchet [8] proposed to apply component-based engineering approach to the development of multimodal applications. In 2007, a survey from International Data Corporation showed that more than 50% of software developers had applied the concept of components based software engineering (CBSE) during their project [3]. CBSE not only lowers the complexity of developing multimodal applications and thus delivers better software, but also increases the reusability and extensibility of the multimodal applications. For example, CBSE defines a multi-sensor device as a collection of components, thus allowing the description for each component to be reused for other devices. However, existing frameworks classify components according to their physical capabilities rather than their functional nature, which decreases the applications' flexibility and reusability.

MVC has been widely used since the 1980s after the concept was published by Xerox PARC for Smalltalk-80 [10]. It is a design pattern, which separates the business logic, presentation to end-user and information collection into three sections. To this end, an application is divided into three core components: the model, the view, and the controller. Each of these components handles a discrete set of tasks. As a result, the MVC design pattern can ensure a clear task division as well as the interrelationship between the different tasks. As each component within the MVC is self-contained and separated from the other two components, it also ensure fast prototyping and high reusabilty. When it comes to coding multidisciplinary and large scale heterogeneous widgets, MVC offers the advantages of decoupling the presentation logic, program logic and modalities modeling, which lowers the barriers required of non-technical users, shortens the development cost and increases the flexibility and extensibility of multimodal applications.

Conceptually, the Model is responsible for handling the behavior and properties of specific domain related problems, such as the definition of the modalities or the categorization of user profiles. The View acts as a presentation layer between the components. It is defined as the interface that provides feedback to the end-user and encapsulates abstract output devices. The Controller receives events from users, models the input devices, and extracts high-level semantic meaning of the interactions from incoming signals. Figure 4-3 shows the distribution of interactive components between four development processes. It can be seen that the development of the components for each process falls under the expertise of a different class of developers.

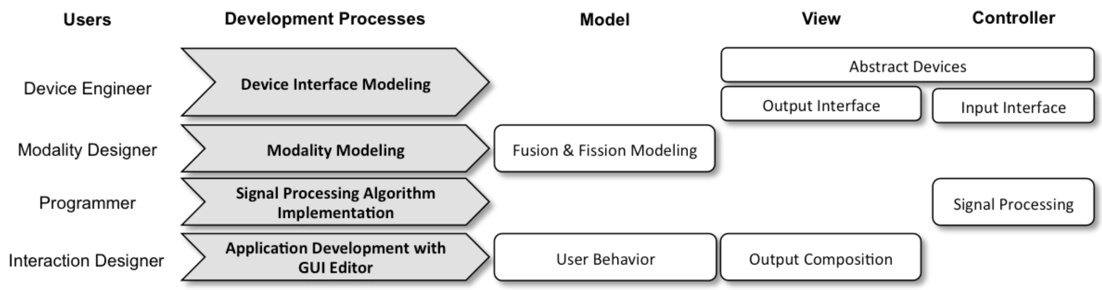


Figure 4-3 Development Process of multimodal applications and Classification of interaction components according to MVC design pattern

Developing multimodal applications on the i*Chameleon platform involves four different situations: device interface modeling, modality modeling, implementation of signal processing algorithms and application development. Each scenario has a specific focus and well-defined steps. Based on the defined structure from i*Chameleon, developers can collect the requirements of the application to model or implement the components. Finally, developers are required to deploy the components to the platform by using specific services provided by the middleware.

4.2 Device Interface Modeling

Device engineers are responsible for widget related adaptation, such as creating a new driver for existing widgets or integrating new widgets to the system. To decouple the sensors, actuators and device under our model, three types of devices exist: input interface components (e.g. a light sensor, or an accelerometer), output interface components (e.g. display, sound, etc), and abstract device components (which function as a general profile of a particular set of input or output interface components and describe the properties of the devices).

In this way, the abstract device component only captures the device-dependent properties while the input and output interface describes the interaction-dependent behaviors. Therefore, each interaction device can be associated with a number of input and output interfaces, which makes it possible to model heterogeneous widgets over a single device. Figure 4-4 shows the relationship between the three components. For example, rather than modeling a game controller as one component, it is modeled as an abstract device component (describing the operational environment and Bluetooth protocol) together with input interface component (buttons, accelerometer) and output interface component (vibration, LED).

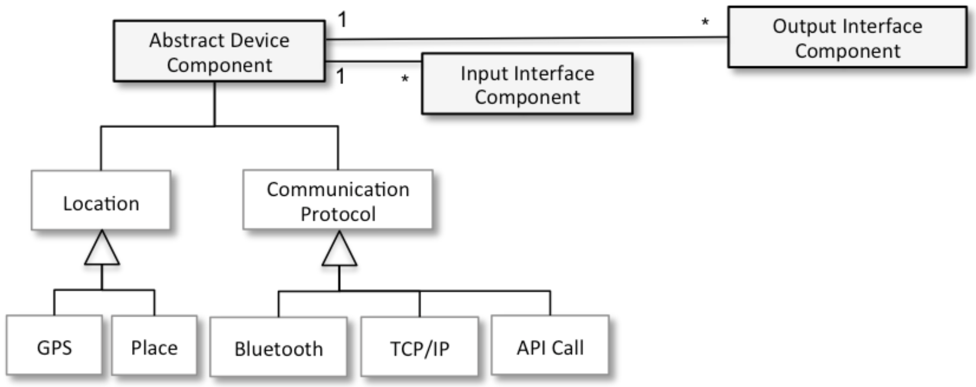

Abstract Device Component
1
1
*
*
Output Interface Component
Input Interface Component
Location
Communication Protocol
GPS
Place
Bluetooth
TCP/IP
API Call

Figure 4-4 Meta-model of abstract device component and its relationship to other device interface components

The idea of the abstract device component is similar to the hardware abstraction layer [38] and Figure 4-4 presents a meta-model. It captures the widget's nature and properties in two dimensions: location and communication protocol. The first dimension is location-awareness, for instance, based on global positioning or localized token. This can enrich the context information and awareness of the physical environment surrounding. A communication protocol is also captured to abstractize the adapters for runtime communication. Our current communication protocol models the basic TCP messaging routines, the shared memory data transport protocol and the xBee and Bluetooth transmission protocols. The required communication package is generated automatically when the model is deployed to the device interface integration service. This allows programmers to create other communication protocols and deploy them independently.

In summary, the abstract device component provides a profile of the static or interaction-independent mechanisms of the interaction devices and also relates together heterogeneous components according to their nature and functionality. This allows on-the-fly switching between components of the same nature to optimize for the best performance of the system. For example, given an application that needs to capture speech for transcription, the system may automatically choose to use a close-range microphone, if one is available, over a tabletop microphone, if the context is a meeting, as the former provides a better signal in such an environment.

The abstract device components are modeled by the device engineers, according to the physical capabilities and functionalities of the associated input/output interface components.

The model for the input/output interface components captures the dependencies between the components and the operating environment, as shown in Figure 4-5 and Figure 4-6. This dimension models influences from physical factors, for example, loudness is an indicator of accuracy for certain widgets that are affected in noisy conditions. This dimension can be extended to other properties by modifying the XML schema in the kernel. Therefore, by positioning the widget in different working environments, i*Chameleon can offer a handover mechanism according to the contextual information in order to enhance the user experience, or to offer the most reliable service. Taking our example of multiple available microphones, the operational environment can provide information about the performance of the device given the noise level of the operating environment. Given this information, the kernel can then make a decision about which microphone to use to provide the most reliable service.

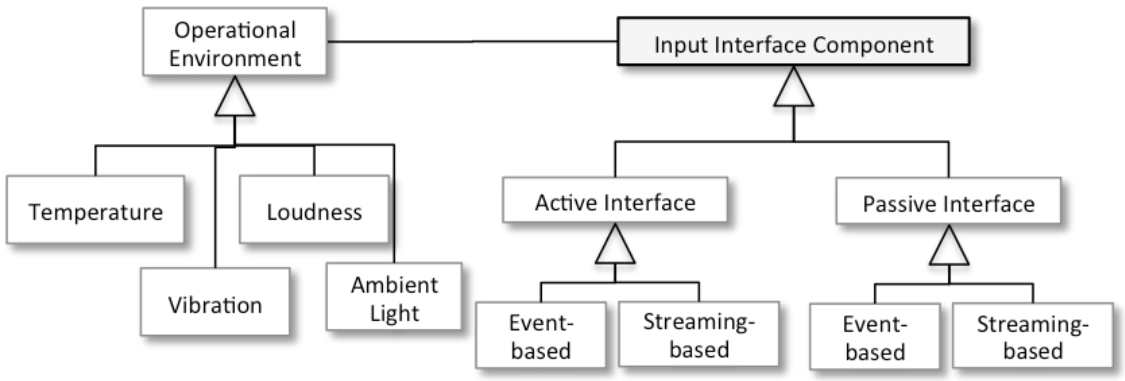


Figure 4-5 Meta-model of input interface component

The input interface component is the only component to capture human output, such as body moment, or sensory input such as light sensors embedded in the surroundings. As it directly generates the events for the i*Chameleon platform, it belongs to the controller aspect of the MVC design pattern. It is divided into actively involved or passively involved and requires the device to translate human output into a form suitable for computer processing. Active modalities refer to actions which users issue intentionally, while passive modalities represent commands that are not explicitly expressed by the user [8][43]. The categorization of the input signal offers a way to prioritize and select the output modality. Furthermore, input signals are classified into event-based or streaming-based. Event-based input modalities react to user interactions that produce discrete events while streaming-based modalities sample continuous input signals from the user and produce a time-stamped array as input to the kernel. For example, the accelerometer from game controllers generates streaming events in an active interface.

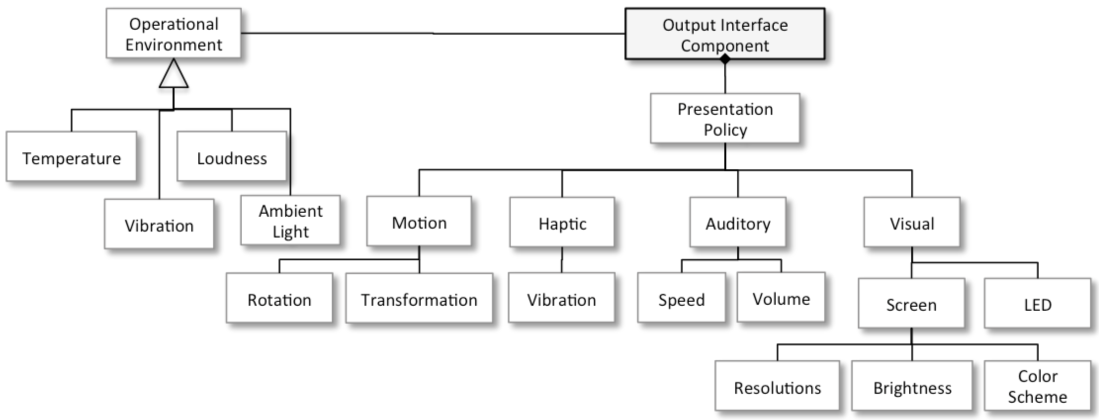


Figure 4-6 Meta-model of output interface component

All feedback to the users is handled by the output interface component, which is classified as a view component in the MVC pattern. It not only models the traditional screen display, but also models human perception according to human senses and motion. Sight, hearing and touch are modeled while smell input and taste input are not taken into consideration as they are not currently commonly used in human-computer interaction. Motion provides another aspect for describing the physical movement of output devices, such as robots and other actuators. The main concept of this meta-model is that it generalizes the common features of output widgets. This not only provides an abstraction interface that allows device engineers to reuse the common components to facilitate the development process, but also allows the platform to enhance the selection mechanism of the output devices. For example, in vision output, the screen is defined and engineers can model the screen by specifying resolutions, brightness and color scheme. As a result, each screen display must be modeled according to the defined specification. During the execution, the kernel binds the output device to a screen with a high brightness if the detected environment is dark.

To summarize, based on this meta-modeling, a single device is no longer defined as one and only one component. It can be divided into view (output interface component) and controller (input interface component) by association with the behavior of the devices (abstract device component). By separating hardware dependent components and interaction dependent components, it not only avoids unnecessary dependencies, but also provides the ability to discover the capabilities of other devices. This way, pervasive environments with multimodal interaction can be established.

4.3 Modality Modeling

Modality refers to the way an idea is expressed or perceived, or the manner in which an action is performed [42]. For example, human interaction modalities include speech, vision, gestures, facial expressions or body movement. It describes the manner of interaction and is independent of the hardware definition or implementation algorithm. According to Zeljko, modality modeling focuses on the notion of an abstract modality, which generalizes the common characteristics of HCI modalities regardless of their specific manifestations [45]. Zeljko specified the fundamental classification of input modalities, which were based on the nature of the input signal, such as event-based or streaming-based. Bruno [19] summarized human-machine interaction with four states: decision state, action state, perception state and interpretation state. Each state involves different levels of data abstraction, such as sensory level, feature level or decision level. Sensory level data is also called raw data, which is acquired from an individual widget. This type of data may be noisy and requires specific filtering techniques to extract the semantic meaning that is representative of feature-level data. Finally, based on the extracted features and the addition of the time dimension considerations, the decision-level data provides a response through the output modality.

Multimodality modeling is not a simple task. It has to consider the combination of single modalities at different data abstraction levels as well as the coordination between them. For example, some feature level modalities extract semantic meaning from multiple sensory level modalities. Also, an output modality may need to be coordinated with a number of possible devices. In summary, when defining a meta-model for multimodality interaction, we need to consider the following:

- The receiving pattern: e.g. Are the input signals event-based or streaming-based.
- The data abstraction level: e.g. Is the incoming data at the sensory level, or has the device internally already translated it to feature or even decision level?
- The fusion pattern: e.g. Should signals be combined in a parallel or sequential manner?
- The dependency between the modalities: e.g. Do certain modalities combine or depend on others to generate higher-level multi-modal signals?

We first classify the modality component according to its nature, in other words, whether it is an input modality or output modality. An input modality receives abstracted data patterns from input interface components while output modality sends abstracted commands to output interface components. Figure 4-7 shows the modeling of the output modality. Uni-modal output is described by the receiving pattern, which can be event-based or streaming-based. For example, an event-based output modality might be the playing of a video clip file. A streaming-based output modality models real time processing feedback, such as displaying the signal from a camera.

Simultaneously, we also apply the CARE model [50] to model multi-modal output modality. The CARE model provides a methodology to describe and model user-machine interaction. CARE stands for complementarity, assignment, redundancy and equivalence.

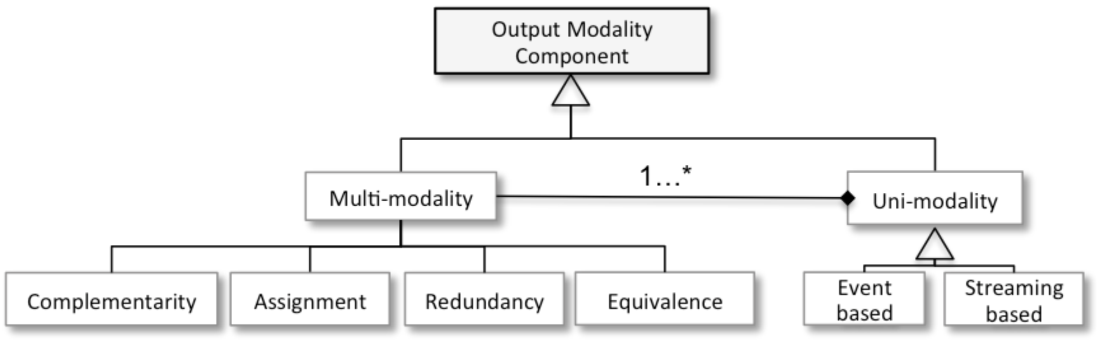


Figure 4-7 Meta-model of output modality component

Complementarity is involved when multiple complementary output modalities are needed to produce the feedback within a temporal window, and when all of them are required for the interaction to result in success. An example is sending an instruction to a robot car to move forward, which requires the turning on of both left and right motors. Assignment indicates that one and only one output modality can lead to another output modality. If the previous state cannot be reached, the next state will not be executed. For example, instructing a robot car to turn right requires both motors to stop first, then the right motor is started. Equivalence output modality implies that multiple output modalities can be used to produce the desired feedback, however, only one would be used at any one time. For example, it may be possible to produce feedback to the user through either sound, or displaying the text on the screen. However, if the external constraints are added, the system may not choose to use voice while in a noisy environment, or the screen device when the user is walking down the street with the device in his pocket. Lastly, redundancy indicates multiple output modalities, which even if used simultaneously, can be used individually to produce the desired feedback, which mean that either one of the output modality can be used to express the same meaning. For example, pushing a notification to the mobile phone and sending an email can both send the user a message. However, either one can also achieve the same outcome. Therefore, by using CARE properties, relationships can be added between different output modality components.

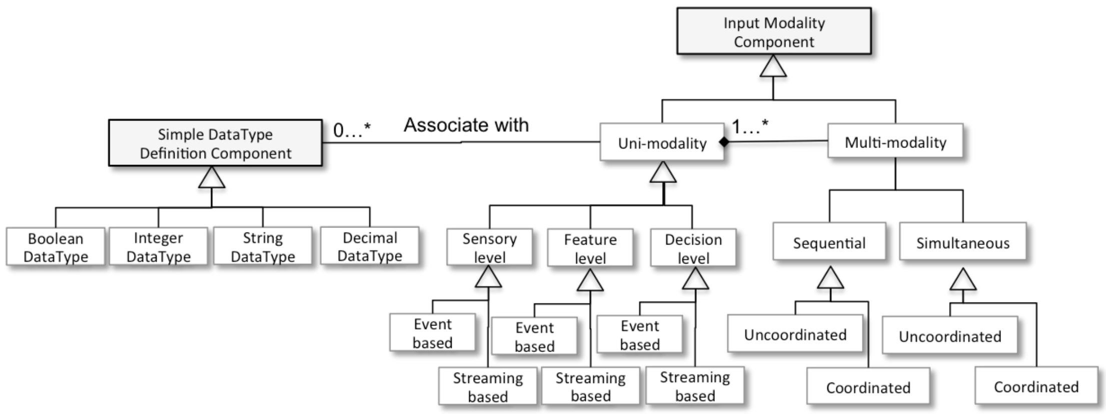


Figure 4-8 Meta-model of input modality component

Similar to output modalities, input modalities may be uni-modal or mulit-modal, as shown in Figure 4-8. Input modality components describe the commands, which are organically triggered by the human(s). A uni-modal input represents a singular form of interaction while multi-modal input integrates several modalities sequentially or simultaneously. A uni-modal input is classified into three different data abstraction levels and associated with simple data type definition components, which use four pre-defined different data types (Boolean, integer, decimal and string). Data abstraction levels include sensory-level, feature-level and decision-level. Sensory-level data is also known as the raw data from devices (for example, a set of coordinates). Feature-level data represents semi-processed information, generated from sensory-level data or directly retrieved from devices (for example, a set of hand movements). Finally, decision-level data illustrates the meaning behind the input modality, which can be the analyzed result from feature-level data or directly triggered by devices (for example, a zoom-in gesture). Each data abstraction is divided into event-based input modalities or streaming-based input modalities. Event-based modalities react to user actions by producing discrete events while streaming-based modalities produce a time-stamped array for particular signals. For example, a recognized voice token is an event-based sensory-level input modality and a point-set would be a streaming-based sensory-level input modality.

The CASE model is applied to model multiple-input modalities. It describes the means by which modalities are combined at the integration engine level and it focuses on different possibilities of modality combinations [42] according to "use of modalities" and "fusion of modalities" (Figure 4-9 left).

“Use of modalities” expresses the temporal availability of input modalities which describe the receiving pattern. If the modalities are classified as “simultaneous”, this means that the system may employ multiple input modalities in parallel. Conversely, when “sequential” modalities are concerned, this means that the system processes the modalities one after another. “Fusion of modalities” considers the coordination of incoming modalities, which may be coordinated or uncoordinated. In our meta-modeling scheme, coordinated modalities require both modalities to be triggered to generate another modality. For example, when a “single click” is combined with another “single click”, a “double click” modality is generated. In contrast, uncoordinated modalities refer to modalities that are independent from each other, where either one of them, when triggered, would lead to the resulting modality. For example, when a user performs a “drag-and-drop” on a virtual object using the mouse and executes a “put-that-there” voice command in parallel, only one command will be enforced.

With the discussed classification methods, CASE model introduces four properties: concurrent, alternate, synergistic and exclusive.

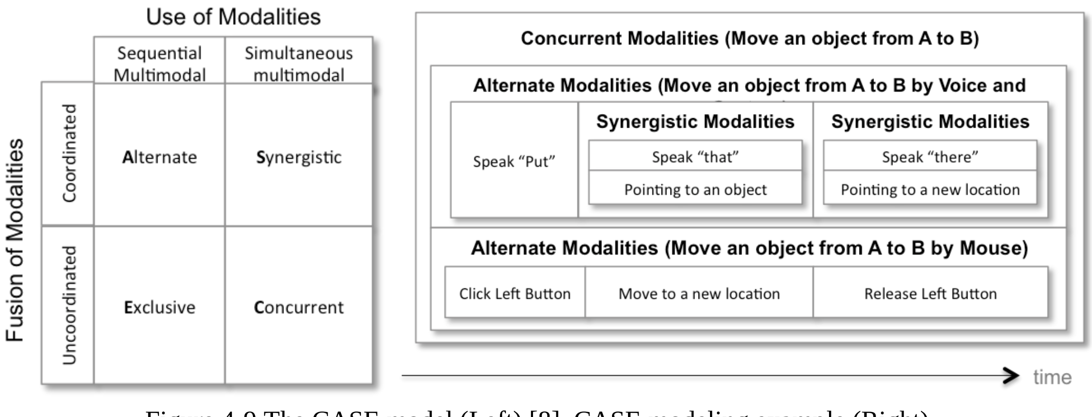


Figure 4-9 The CASE model (Left) [8]. CASE modeling example (Right) .

Figure 4-9 (right) illustrates an example of modeling a multimodal modality component with different combinations of modalities. A user can move a virtual object from A to B using two different modality channels, either by voice and gesture or mouse. Therefore, at the highest abstraction level, this modality component is would be modeled as *concurrent*. For the “Put that there” command, the user is required to speak the keywords in the correct sequence, so, it is classified as *alternate*. This is the same situation as a mouse drag-and-drop modality. Lastly, during the "Put that there" command, user is required to point to an object while saying the keyword "that" and point to a new location while saying the keyword "there". As these actions are coordinated and need to be triggered simultaneously, they are modeled as *synergistic*. With the use of CASE model, a modality repository can be built. It not only can define the definition of each modality, but also store the dependency between them. This meta-model can model the modalities at different abstraction levels.

Table 4-1 Fulfilling the design requirements: The Meta Model

	Input Modalities	**Output Modalities**
Receiving Pattern	Uni-modality	Uni-modality
Data abstraction level	Simple DataType Definition Component	-
Fusion pattern	Multi-modality with CASE model	-
Dependency between the modalities	Multi-modality with CASE model	Multi-modality with CARE model

4.4 Signal Processing Algorithm Implementation

Signal processing algorithms implementation refers to the development of a new analyzing algorithm with specific input data that can be deployed to the i*Chameleon kernel to improve or enhance the performance or accuracy of the application. Each algorithm is wrapped as a self-contained component which needs to have a self-descriptive profile, follow the proper communication protocol, associate with specific modality component(s) and generate particular modality components. This kind of components belongs to the "Controller" pattern of the MVC design model.

A self-descriptive profile is used to define the requirement of the algorithm (processing power or frames per second) and provide performance information to the kernel (accuracy or turnover time). Therefore, if there exists more than one component that fulfills the requirements of the situation, the kernel can select the most suitable algorithm. For example, if a number of algorithm components are distributed around the network, the kernel can automatically select the appropriate component according to the round trip time based on the communication protocol and accuracy defined in the profile.

Similar to the abstract device component, the communication protocols are wrapped in an abstraction layer that currently models the TCP/IP, Bluetooth and xBee protocols. Programmers are therefore only required to define an XML document to handle the communication protocol when a new algorithm component is developed.

Another feature is the integration of publish/subscribe architectural pattern. Once the algorithm component is registered to the modality components, the web services protocol (discussed in chapter 5) or publish/subscribe mechanism (discussed in chapter 6) will automatically route the modality to the suitable algorithm, or the analyzed result will notify the corresponding components.

We categorize those generic algorithms for implementing fusion algorithms into three types: filters, parsers and analyzers.

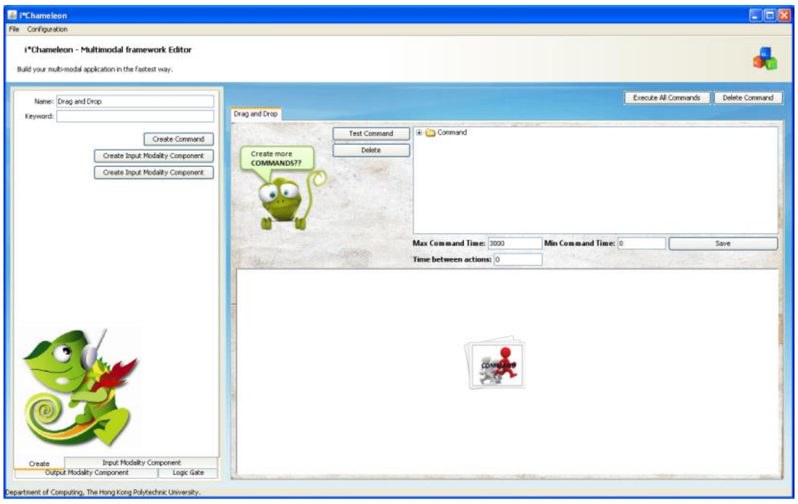
- Filters (Sensory-level modality) handle data pre-processing. They receive the raw data from the sensors or other hardware, and based on the hardware definition, translate the raw data into well-defined sensory-level modalities. For example, a filter will is used to normalize the coordinates for a 2D point object from a video camera to ensure all input data is consistent for analyzers or parsers.
- Analyzers (Feature-level modality) are responsible for computation and extracting the meaning from a stream of sensory-level modality or feature-level modality for tokenization. For example, after 2D point modalities have been obtained from filters, analyzers will compute it and extract the meaning from it, such as a zoom-in hand gesture command.
- Parsers (Decision-level modality) determine the output modalities from the different tokens extracted from the analyzers. For example, after the zoom-in gesture is detected, parsers will gather the contents from the existing execution state. Based on the execution conditions, the parsers will notify the “View” components to trigger the commands.

Therefore, developers can design and deploy suitable algorithms without affecting other components based on the requirement of the application.

4.5 Application Development

Creating a new application based on a set of implemented modalities and devices requires the interaction designers to have a certain level of programming knowledge. Since we have separated the hardware-dependent tasks from the modality definition and algorithms and also applied the concept of component-based software engineering, the interaction designers now only need to map the input modalities to the output modalities, without having to know the internal details and specifications of the hardware. This not only results in a high degree of reusability and fast development, but also lowers the barriers of multimodal interaction development.

We extended from previous work [57] to develop the i*Chameleon interaction editor according to the "View" aspect of the MVC design pattern. It is designed to allow non-computing experts to create high-level modality components and link them together to form a command. We applied the concept of logic gates, but limited the user's choice to AND and OR gates. Therefore, with a graphical user interface (Figure 4-10), non-technical users can easily drag-and-drop modeled input modality components and map them to output modality components to form a multimodal application.


i*Chameleon
File Configuration
i*Chameleon - Multimodal framework Editor
Build your multi-modal application in the fastest way.
Create Command
Create Input Modality Component
Create Input Modality Component
Execute All Commands
Delete Command
Drag and Drop
Test Command
Delete
Command
Create more COMMANDS??
Save
COMMAND
Create
Input Modality Component
Output Modality Component
Logic Gate
Department of Computing, The Hong Kong Polytechnic University.

Name: Drag and Drop
Keyword:
Max Command Time: 3000
Min Command Time: 0
Time between actions: 0

Figure 4-10 i*Chameleon Modality Component and Command Editor

In addition, users can define certain preferences with an XML file. These preferences model the user's habits and preferences by setting priorities to available modalities. For example, if a user prefers using voice input rather than gesture control, once the priority is configured, when both inputs are detected, the kernel will default to voice input whenever possible.

In summary, there are a number of advantages of applying the MVC (model-view-controller) design pattern to multimodal interaction development. Its enforcing of the principle of *separation-of-concerns* in the i*Chameleon framework and the resulting four different aspects of the development process (device interface modeling, modality modeling, signal processing algorithm implementation and application development) enables cross-disciplinary collaboration between device engineers, modality designers, programmers and interaction designers. This simplifies and accelerates the application development process.

Chapter 5. Web Service Architecture

A web service is a self-contained, self-describing and modular application that can be published, located, and invoked across the Internet [25]. Together with contemporary networking infrastructure, web services provide high compatibility with different components. Once the components are deployed to the i*Chameleon platform, they can discover and interact with the deployed components dynamically.

Web services are supported by a set of XML-based protocols such as Universal Description, Discovery and Integration (UDDI), Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP). They are platform-independent, conducive to heterogeneity and are supported by all current IT infrastructures, with libraries in nearly all major programming languages. Developers interact with SOAP, which is a specification protocol for exchanging uniform information, by passing XML-encoded data, bound to HTTP as the underlying communication protocol, from one endpoint to another. It uses XML messaging over plain HTTP, thus avoiding networking issues, such as firewall problems, allowing for remote procedure calls via simple request/reply. WSDL describes the functionality offered by a web service. It offers a machine-readable description of the usage of the web service, the way the function is triggered, what parameters it needs, and what data structures it returns. Meanwhile, UDDI provides directory services for services to list themselves over the Internet, which is beyond the scope of this thesis.

Figure 5-1 (left) shows a typical web service architecture, which involves a web service broker, web service provider and a number of clients. A web service broker acts as a repository for all available web services described by WSDL. When a client requests a service, the broker will offer the most suitable one for the client, and finally the client can directly interact with the service provided by SOAP.

The architecture of the i*Chameleon platform follows a similar model. The application service consists of the broker and provider. The broker generates the necessary packages for the clients while service requests are received from the provider. The input widgets, such as touch screens, and output widgets, such as vibration motors, make up the clients.

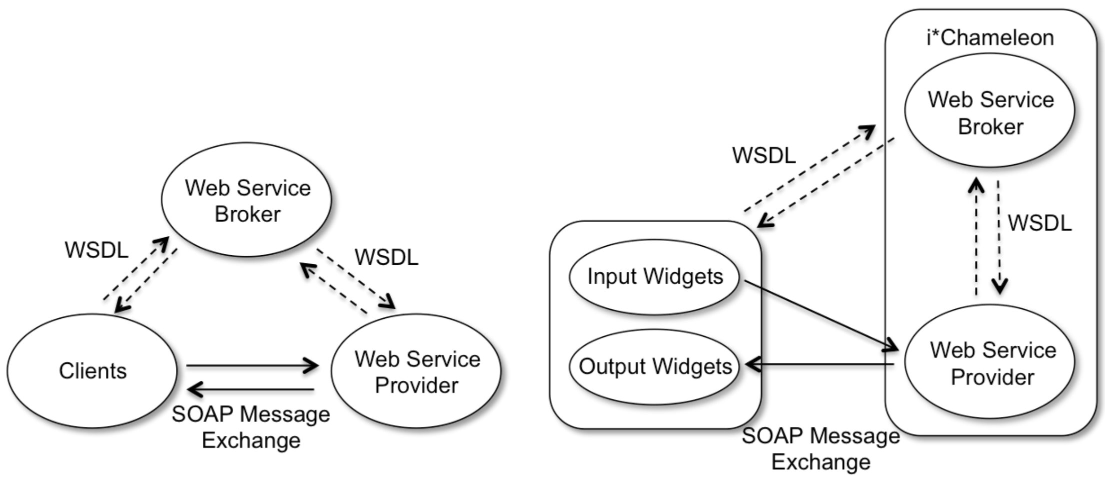


Figure 5-1 (Left) Web service architecture (Right) i*Chameleon Web service architecture

One of the biggest problems faced by multimodal interaction is that hardware devices often come with their own language-dependent or platform-dependent libraries. This makes it difficult to integrate multiple modes of interaction into the same application. Using web services can solve the problem of incompatibility between the programming languages required by the various hardware devices. Freed from the programming language constraints, developers only need to call the corresponding web services. Besides, web services also provide a standard protocol for communication over the Internet, which allows distributed devices to interact with each other in remote areas. An example might be controlling a team of robots in different locations. Therefore, the concept of “Interaction Cloud” can be achieved.

In this chapter, we will first discuss the detailed architecture and workflow of the web service-based i*Chameleon followed by the MVC classification over this platform. In section 3, four development processes will be studied.

5.1 Web-Service Architecture

Web service-based i*Chameleon involves two core kernels, the application server and the co-processor. Figure 5-2 shows the detailed platform architecture. Linked with the application server is the co-processor, which receives and processes input modalities, such as visual and auditory information. The purpose of the co-processor is to offload some processing tasks from the application server. It receives data from the application server, analyzing and recognizing events and sending notifications back to the application server with the intended action, i.e., the command to the target application. Communications between them are handled by a socket channel with TCP/IP connection implemented via object serialization, allowing us to create reusable objects and transfer them through standard sockets with common modeling definitions. It manages sensory and application input separately in order to allow each widget to be reused and extended [24]. A modeling and definition language has been developed for i*Chameleon to support the sharing of information among different components.

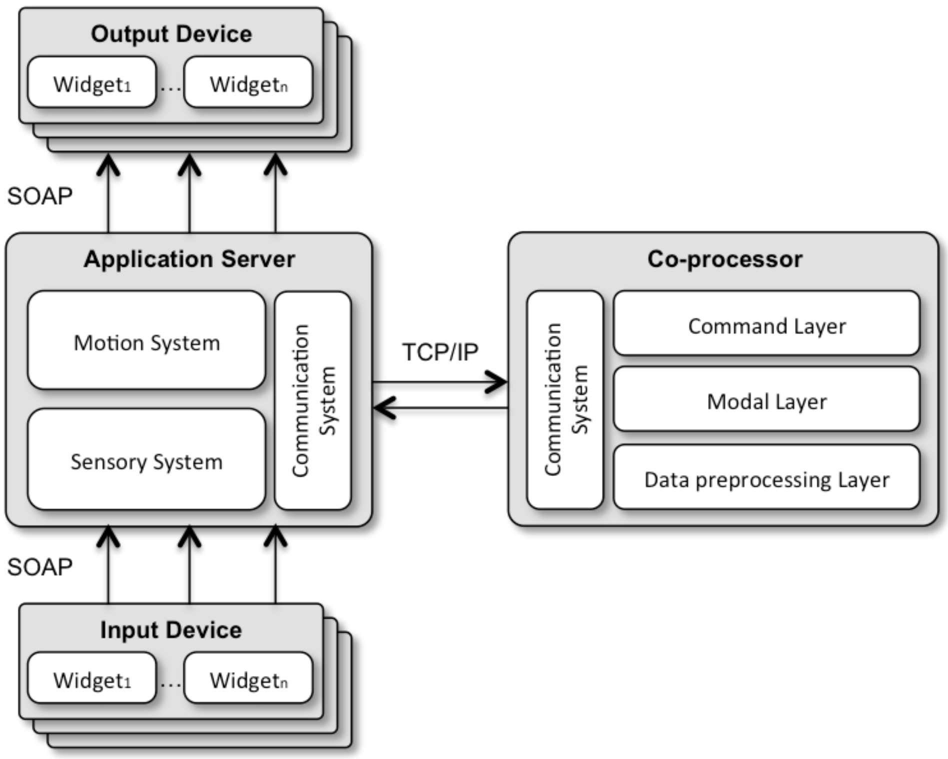


Figure 5-2 i*Chameleon includes two core sub-systems, a web services application server and a co-processor. The application server is responsible for receiving signals from the Input device, communicating with the co-processor and coordinating with the output devices. The co-processor analyzes the input data and sends notifications back with the analyzed command to the application server.

5.1.1 Application Server

The application server consists of three modules: Sensory System, Communication System and Motor System. The sensory system is responsible for handling all input signals from input widgets and communications using SOAP. It provides three important functions. First, it manages all sensory input devices at the workstation level. Each device can have one or more widgets associated with it; for example, an iPhone can be defined as a device with three input widgets: voice, touch and tilt. At the beginning, each workstation declares itself as an input device, which is a client of the sensory system. After declaration, this input device can be deployed to i*Chameleon, which would return a device identifier to the client. Hence, the application server manages the list of connected widgets which enables i*Chameleon to distinguish the senders of events.

Secondly, the sensory system handles all input events from connected input devices. Each device can be dynamically associated with the sensory system by invoking web services, and be given an identifier as shown in Figure 5-3. This method has the advantage of adding new devices without stopping the multimodal application. For example, after a device is initiated, a developer can append a new speech recognizer (widget). Thereafter, once incoming verbal input is detected and recognized, it can trigger an event to notify the sensory system.

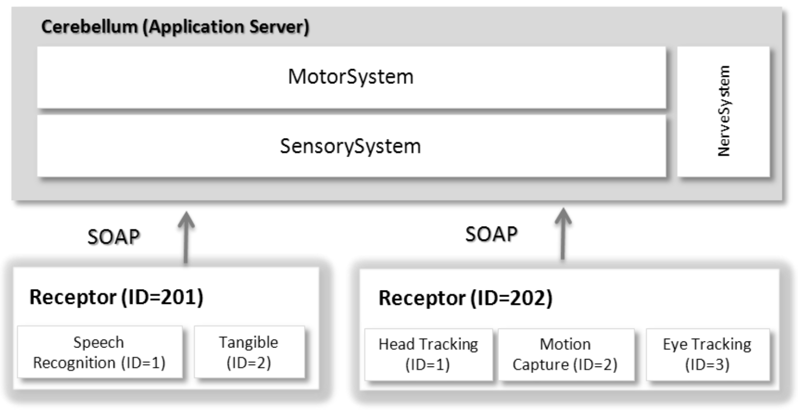


Figure 5-3 Two sensory receptors associated with Application server with specific identifier and each receptor consists of different modalities.

Thirdly, the sensory system also acts as a hardware abstraction layer, which is responsible for handling the data communication between application server and co-processor, including object standardization and creation. Based on the received raw data, it translates them into specific data objects according to the data definition stored in the communication system. Also, based on the modeling technique, i*Chameleon defines a set of abstract device classes which provide the structure of the hardware configuration. According to the device’s nature, we classify them into four categories as shown in Figure 5-4, with associated examples depicted in Table 5-1.

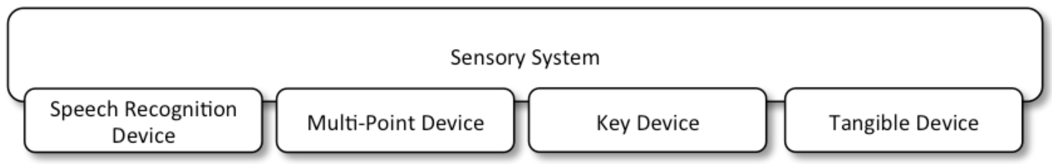


Figure 5-4 Defined devices under sensory system

Table 5-1 Devices and associated modality

Widget	Signal Object	Modality
Speech Recognition	Recognized Word	Voice
Multi-Point	A Set of Points	Hand Gesture
Key	Pressed Key	Key
Tangible	Orientation	Tilt

The communication system is responsible for storing the modalities’ definitions, transmitting the incoming modalities and analyzing modalities between application server and co-processors. This is implemented with object serialization, allowing objects to be transmitted over the network via TCP/IP. The co-processor connects by making a connection request to the application server, establishing a communication channel for the transferring of objects.

The motor system is responsible for handling the analyzed events from co-processors and forwarding the commands to the corresponding output devices for execution. The process orchestrations are similar to those of the sensory system. The motor system thus acts as the communicator between the application server and command executors. When the motor system receives the command from the co-processors, it triggers a particular output device on a specific workstation to execute the command. Command executors are platform-dependent, and each workstation owns its executors. When an executor receives a notification from the application server, it triggers the output device to perform the action.

Besides, the motor system also handles the selection of output devices. It stores the device information and configuration. Therefore, it automatically binds the commands to the best-fit devices for execution. For example, if a screen with better resolution is available, the motor system will bind the display with this device.

Using the web services declared in the motor system, developers could customize or create commands dynamically. A command is responsible for storing the execution conditions, which include events, actions (which are inputs to user programs, e.g., games) and time constraints. An event, an invoker in the command design pattern, is created by the end-user, which describes when the associated action should be triggered. Actions are the instructions that need to be executed when a command is triggered. For example, an action can be "Open File Explorer", "Make the WiiMote Vibrate" or "Right Click the Mouse".

5.1.2 Co-processor

The co-processor handles all the computation tasks involved in translating data from receptors into meaningful commands, such as mouse moves or key clicks. It is organized into three layers: Data Preprocessing Layer, Modal Layer and Command Layer. Each layer is independent of the other two. Hence changing one layer will not affect the other layers.

After receiving the structured data from the communication system, the objects will first arrive at the data-preprocessing layer. After passing through appropriate filters, such as noise filter, point tracking filter or transformation filter, data integrity and accuracy can be ensured.

The most complex part of the co-processor dwells in the modal layer, which accepts packaged data from the hardware abstraction layer, analyzes the input and recognizes the events. These events are then passed to the command layer, which maps it to a corresponding action. This modal layer can be subdivided into two major components. The first component consists of a number of analyzers, which handle data processing and computation in order to translate the raw resources into tokens using pre-defined rules. The second component is a parser that parses the tokens into parse trees of different modalities, such as gesture, voice or others using an interaction grammar and finally maps the parse trees to events. Events are then passed to the command layer to be executed. Due to the layered design, developers can attach or detach interaction devices without any modification to the other layers. When data is received from the hardware abstraction layer, the relevant modality is triggered. For example, a series of 2D Points triggers the Gesture modality. As we defined four categories of widgets, four corresponding modalities are implemented, as illustrated in Table 5-1 and Figure 5-5.

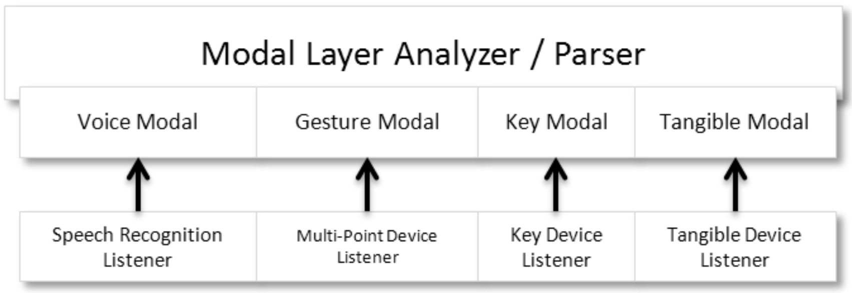


Figure 5-5 Modal Layer

The final layer is the command layer. As the name implies, this layer is responsible for determining which command needs to be executed. After the modal layer recognizes the events, these events still appear independent from one another. It is only known that at a certain moment, say, t_1, two events, e_1 and e_2, are triggered, but it is unclear which command needs to be executed. The command layer gathers the execution conditions of all application inputs. Once an event is triggered, it consolidates the triggered events with all previously triggered events to determine whether an associated command is to be executed. It then sends back the interpreted signal for the associated command to the motor system in the application server.

To illustrate the operation of i*Chameleon, suppose a multi-touch table-top screen is used to display and manipulate a digital mapping application. A user puts four fingers on the table and pinches them together, to indicate a zoom-in action. The machine managing the table sends the coordinates of the corresponding points to i*Chameleon via web services. The data are handed over to the hardware abstraction layer for preprocessing and then passed to the modal layer for analysis. The modal layer interprets the incoming Points as a Gesture, which is passed to the command layer, to be mapped to the appropriate command for notification to output widgets through the motor system.

5.2 Platform Workflow

In i*Chameleon, each input device corresponds to a particular input in the web service. Figure 5-6 illustrates how an event is triggered, recognized and an action is executed. An application input is fired by one or more events, while an input device triggers an event.

The sensory system captures raw data and notifies the application server. Data together with the device information will be passed to the sensory system, which translates the raw data into a generalized data object, which stores information related to the event. Finally, the object is sent to the co-processor for analysis.

The co-processor is responsible for listening to the communication system event handler, receiving a signal, and extracting raw data from the signal. The representation of data is device-dependent. The translated object will be passed to a set of data-preprocessing filters, such as noise filter or normalizing filter, to ensure that the data is valid and accurate.

Then, the output data from the hardware abstraction layer will processed by the proper algorithm for the modality, e.g., applying a gesture recognition algorithm on a set of points. An analyzer will eventually parse the input data and map it to a command based on a set of grammar rules, paving the way to trigger the corresponding action (command for target application).

Finally, the co-processor notifies the application server of the action intended by the user, and the motor system in the application server gathers the intended actions and notifies the application input devices to execute the commands by invoking the corresponding web services.

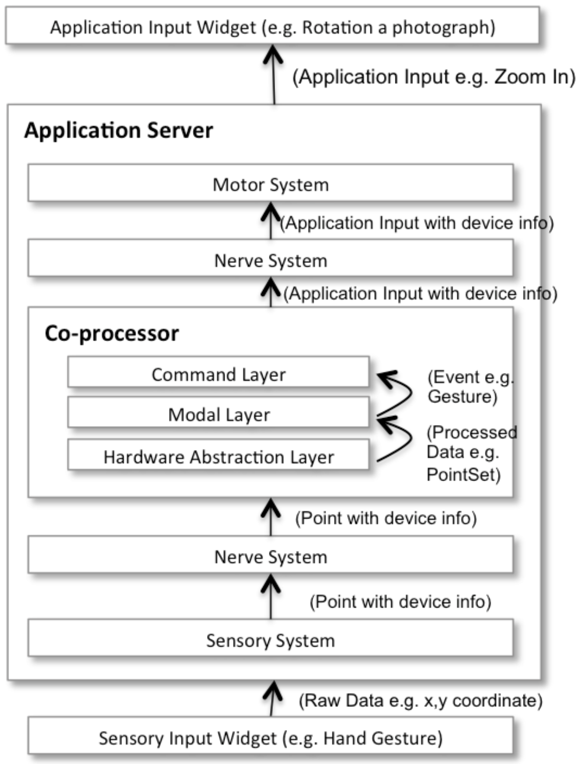
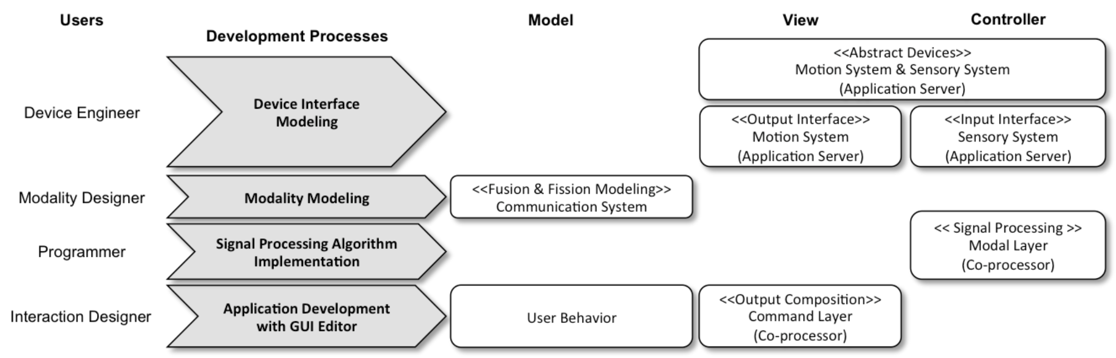

Application Input Widget (e.g. Rotation a photograph)
(Application Input e.g. Zoom In)
Application Server
Motor System
(Application Input with device info)
Nerve System
(Application Input with device info)
Co-processor
Command Layer
Modal Layer
Hardware Abstraction Layer
(Event e.g. Gesture)
(Processed Data e.g. PointSet)
(Point with device info)
Nerve System
(Point with device info)
Sensory System
(Raw Data e.g. x,y coordinate)
Sensory Input Widget (e.g. Hand Gesture)

Figure 5-6 i*Chameleon workflow. Hand gesture processing starts with the camera capturing coordinates. Generalized point object sends the coordinates to Sensory System via web services. Co-processor translates the raw data into gestures and notifies Motor System to trigger corresponding application.

5.3 MVC Components Classification and Development Cycle over Web Service Architecture

Our suggested web service architecture involves two core systems with a number of components. Each component can be classified into Model, View or Controller according to their nature and functionally. Figure 5-7 illustrates the classification of the MVC components and the associated development process based on the model discussed in chapter 4 (Figure 4-3).



Users
Development Processes
Model
View
Controller
Device Engineer
Device Interface Modeling
<<Abstract Devices>> Motion System & Sensory System (Application Server)
<<Output Interface>> Motion System (Application Server)
<<Input Interface>> Sensory System (Application Server)
Modality Designer
Modality Modeling
<<Fusion & Fission Modeling>> Communication System
Programmer
Signal Processing Algorithm Implementation
<< Signal Processing >> Modal Layer (Co-processor)
Interaction Designer
Application Development with GUI Editor
User Behavior
<<Output Composition>> Command Layer (Co-processor)

Figure 5-7 i*Chameleon components. MVC Classification and Development Process over Web Service Architecture

Device engineers are involved in the process of device interface modeling, which involves the controller model (input interface) and view model (output interface). The abstracted hardware configuration (discussed in section 4.2, Figure 4-4) is required for deployment to both motion and sensory system. If the device engineer wants to deploy a new input device, he is required to:

1. Model the abstracted device configuration XML based on the 4 services provided by sensory system
2. Model the device, e.g. Desktop computer
3. Generate the device driver by the code generator and the driver included
    a. Declaration of the device as an input device
    b. Communication protocol
    c. Widget listeners
4. Append new sensory input widget, e.g. Wii controller that is connected to the declared desktop computer
    a. Integrate the device dependent API for getting the signal, if any
5. Once the device driver has been executed, the device arrival event will send it to the i*Chameleon platform via web services call, and a communication channel will be set up to listen to the incoming signal

Similarly, if the device engineers need to deploy a new output device, they are required to:

1. Model the abstracted device configuration XML
2. Model the device, e.g. Desktop computer
3. Generate the device driver using the code generator and the included driver
    a. Declare of the device as an output device
    b. Declare the communication protocol
    c. Implement the widget listeners
4. Append new sensory input widgets, e.g. a screen that is connected to the declared desktop computer

    a. Integrate the device dependent API for triggering the command, if any
5. Once the device driver is executed, device arrival event will send it to the i*Chameleon platform via web services call, and a communication channel will be set up for sending commands

In addition, modality designers define the input (Figure 4-8) and output modalities (Figure 4-7) according to the modeling we discussed in chapter 4.3. This component is classified as a model pattern as it is related to the behavior modeling. After the designer defined the modality by XML, the kernel can generate the modalities classes. The communication system acts as a repository that manages all modalities. Finally, sensory system or motion system can, based on the modality definitions, translate and structure the raw input to modalities they require.

Programmers work with the kernel of i*Chameleon directly and they are required to develop and deploy signal processing components to the modal layer. With the standardized modalities defined in communication system, programmers may need to reuse the modalities as input parameters and output objects of the algorithm components. For example, if point (x, y coordinates) modality M_1, and hand gesture (zoom in) modality M_2, is defined in the communication system, programmers can deploy their algorithms for retrieving M_2 from M_1. Creating an algorithm component only requires defining the XML configuration, which is shown in Figure 5-8, passing the configuration to i*Chameleon kernel to generate necessary packages and finally implementing the algorithm.

```
<?xml version="1.0"?>

<iChameleon xmlns="http://etoy1.comp.polyu.edu.hk/ichameleon/controller">
        <communication type="TCP/IP" >
                <port>8001</port>
                <hostname>localhost</hostname>
        </communication>

        <algorithm type="analyzer" paradigm="frame">
                <description>Analyze zoom-in from point.</description>

                <inputModalities>
                        <add ref="iChameleon.model.Point" />
                </inputModalities>

                <outputModality ref="iChameleon.model.handGesture.ZoomIn" />

                <feature>
                        <add type="int" value="30" name="fps" />
                        <add type="double" value="0.8" name="accuracy" />
                </feature>
        </algorithm>
</iChameleon>
```

Figure 5-8 XML Configuration of creating a algorithm component. This example illustrates an analyzer receiving a point set that results in a zoom in hand gesture.

Interaction designers can customize the commands using the graphical user interface discussed in [58], which is classified into View pattern according to the MVC definition. With a drag-and-drop editor, non-computing experts can also customize their own interactive commands and execute them with the i*Chameleon web services architecture. When the editor is started, it will retrieve the modalities and algorithm component definitions from the kernel. Therefore, the users can manipulate the deployed modalities and customize the desired commands. After customization, an XML-based file will be generated and deployed to the kernel by specific web services.

This chapter covers the implementation of i*Chameleon platform using the approach of web services. Fundamentally, the components in web services have a clear MVC classification. It also allows interaction components, e.g. the input and output widgets and the algorithm components, to be distributed across different hosts connected by the web service. Based on the suggested modeling approach, different developers can model the components independently and deploy them to the same i*Chameleon kernel, which will then generate the necessary component packages, manage the input data and handle the output commands. This facilitates and speeds up the development procedures.

Chapter 6. Publish/Subscribe Architecture

The publish/subscribe paradigm of communication has recently received increased attention [23]. Different researchers [24, [17, 23]59] have claimed that the publish/subscribe paradigm provides decoupling of distributed interaction in large-scale applications and improves the flexibility of adapting the system architecture.

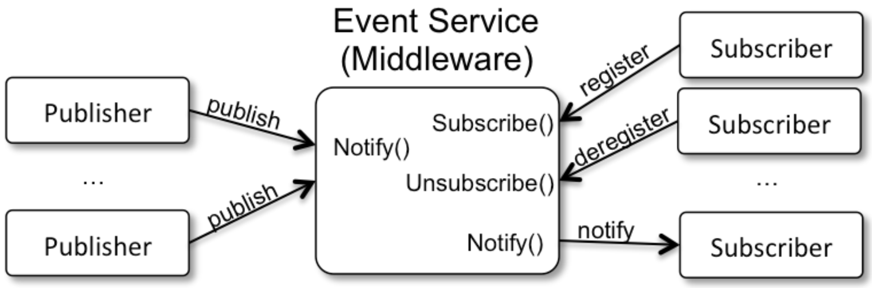


Figure 6-1 A simple publish/subscribe architecture

Figure 6-1 shows a simple architecture for a publish/subscribe middleware. Basically, subscribers express their interest in an event, so as to be notified when matching registered events are generated by a publisher. The publishers do not record references to the subscribers and do not know the number or identity of subscribers participating in the interaction. Similarly, subscribers do not store the list of publishers, neither do they know the number of publishers involved. Therefore, this paradigm offers the ability to decouple with respect to the dimension of space. The event service handles the registration of all the publishers and subscribers and the broadcasting of events to them. This allows the components to be self-contained and self-described. Similar to the web-service architecture discussed in the last chapter, each interaction widget acts as a component. Therefore, these components can be self-contained and independent of other publish components or subscribe components, which fulfills the design principles of “Heterogeneity” and “Self-described and self-contained” suggested in Chapter Three.

In addition to space decoupling, the publish/subscribe paradigm also provides decoupling along time and synchronization dimensions [23]. Time decoupling benefits the interacting components as they do not need to be actively participating in the event services at the same time. In other words, publisher components might publish events even when no subscribers are connected. In multimodal applications, most existing frameworks focus on real time feedback based on absolute space and time frame. However, in a context-aware pervasive system where multimodal interaction is involved, external factors such as the situation of the current working environment need to be taken in account. It may not be possible to capture such data instantaneously, or the capture of such data requires backtracking and linking to previous events or events from disconnected widgets. Time decoupling helps to facilitate the process. Synchronization decoupling ensures components are not blocked while publishers are producing events or subscribers are consuming events and it helps to support concurrent activities. The context-aware environment also provides additional information for decision-making and contributes to the dynamic adaptation of the publishers and subscribers.

In the following sections, we will first discuss the architecture of implementing a multimodal middleware for supporting a publish/subscribe paradigm followed by the workflow of the execution. Finally, the MVC design pattern and the development cycle over this publish/subscribe architecture will be presented.

6.1 Publish/Subscribe Architecture

In order to achieve space, time and synchronization decoupling over a publish/subscribe paradigm, three schemes are defined according to the level of expressive power: namely, topic-based, content-based and type-based [23]. These schemes provide the methodology for publishing events and the different approaches result in different performances. In the following section, we will first discuss the existing three variants in use for designing publish/subscribe systems, followed by the i*Chameleon architecture.

The first scheme is topic-based and it has been widely used in different applications [2]. Basically, topics are represented by keywords. It extends the notion of channels with methods to characterize and classify events. The events are grouped under a specific topic T, every event will map to one or more topics, and subscribers can subscribe to one or more topics.

The topic-based scheme has the advantage of being easy to implement. However, since topics are represented by string keywords, it lacks the descriptive power of the content-based scheme, which augments the topics with additional information about the event, such as the data type of the signal or the magnitude of the data. Implementation under this paradigm is more complex and query languages, such as SQL, may be needed. However, as the content filtering has no information about the relationship between different topics (such as Topic A is a type of Topic B), this approach may increase the risk of redundant events.

Finally, the type-based publish/subscribe paradigm adopts an object-oriented approach and groups events into structured hierarchies. This allows for more powerful filtering of the events, and provides a closer integration of the language and the middleware.

Chapter 4 described modality modeling as based on the signal type, augmented with optional data. For example, a signal may be described as a point with (x, y, z) coordinates, at 30 fps (optional). Therefore, we clearly cannot rely solely on the topic-based approach. As many of our events are related (e.g. different kinds of gestures, etc), the content-based paradigm does not fit our needs. Given all these constraints and requirements, the i*Chameleon platform uses the type-based approach.

In terms of the system architecture, as presented in Figure 4-2 (Chapter 4), the i*Chameleon platform consists of a middleware that coordinates different modalities from different devices. Components can act as publishers, subscribers or both and interact with the kernel. The system architecture and interaction between components and kernel is shown in Figure 6-2. Under a publish/subscribe communication paradigm, different MVC-based components are classified, which allows plug-and-play to the kernel. Modality and device properties are defined as the model; output interface components and descriptions of interactions are classified as the view; and the controller consists of input interface components and algorithm components.

Details about the classification will be discussed in section 6.3. The system is mainly divided into three parts, i*Chameleon kernel (middleware), multimodal application (publishers and subscribers) and algorithm component (publishers and subscribers).

The core middleware is responsible for managing all publishers and subscribers, routing the message to suitable components and deciding on the interactive commands. The kernel involves four managers: device interface manager, modality manager, controller manager and command manager.

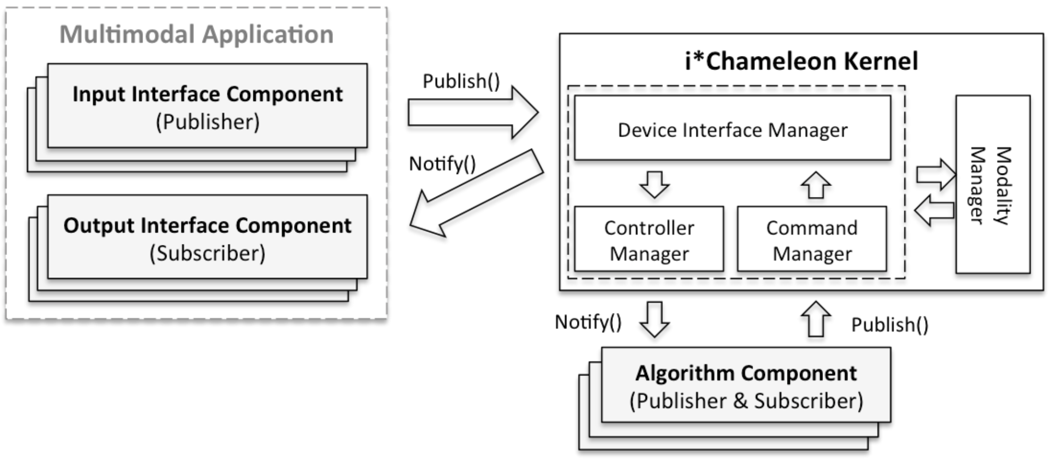


Figure 6-2 Interaction between components (publisher / subscribe) and i*Chameleon kernel

- The device interface manger offers "Device Interface Integration Service", which handles registration and deregistration of input interface components (publishers) and output interface components (subscribers). These components can register their interest in a particular event or publish their events through the services provided by the kernel.
- The modality manager provides "Modality Deployment Service". It acts as a repository for all deployed modalities and handles input and output modalities.
- The controller manager offers "Signal Processing Algorithm Deployment Service". After the programmer implements the algorithm component, it can be deployed to the kernel via this manager.
- The command manager provides an interface for users to create and deploy their interaction files. It also manages the fusion of modalities.

Secondly, input and output devices are modeled as publishers and subscribers respectively. The input interface components are integrated with device dependent drivers that retrieve the sensory data and translate it to the corresponding data structure defined in the modality manager. Once connected to the kernel, the component generates events by calling a publish() operation. Output interface components are subscribers and they express their interest in an output modality, such as “Zoom In”. Therefore, once the kernel receives the required output modalities, which need to be executed, it can notify the relevant output interface components.

The last part of the kernel consists of the algorithm components, which handle signal processing and translate the sensory level data to feature level or others. These components are similar to operators or functions of other programming language, which require input parameters and return value. Therefore, algorithm components are defined both as publishers and subscribers.

6.2 Platform Workflow

In the last section, we provided a detailed description design concept of the publish/subscribe system and the architecture of the i*Chameleon platform. In this section, we focus on the component interaction models.

Registering a publisher, retrieving an event and triggering a command all require a direct and continuous interaction between the different managers and components. The sequence diagram in Figure 6-3 shows the main communication and interaction patterns when a new input device and output device are connected and trigger a command. For example, a smart phone can register as an input device while a robot car can register as an output device. The smart phone can send a tilting event to trigger the robot car to move forward [37].

After the kernel is started, all publishers and subscribers can register or unregister without the time constraints. Once a publisher or subscriber has completed the handshake protocol, they are ready to transfer sensory data or receive notification. The sensory data first publishes to the device interface manager. This manager verifies the data structure of the incoming signal with the modality manager. During the verification process, the modality manager standardizes and normalizes the data with the support of device profile stored in the device interface manager. For example, a point retrieved from an infrared camera (x_1, y_1) is normalized with the devices' resolution. In addition, a timestamp will be added to all sensory data.

Once that is done, the pre-processed sensory data is published to the corresponding algorithm components. According to the subscribed interest of the event, the controller manager forwards it to registered components based on the type of data. If there exist additional algorithm components that subscribe to the same type of sensory data, the controller manager requests the device interface manager for the device profile and compares the accuracy or available resources to decide which component(s) should be notified.

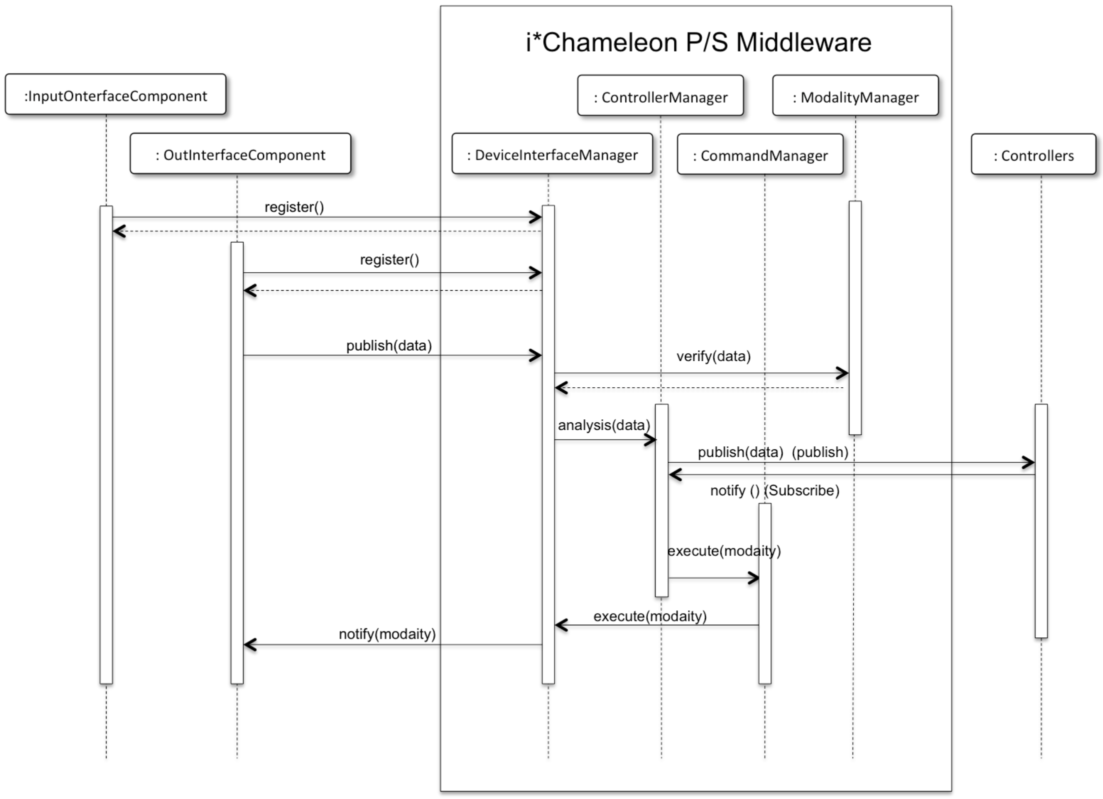


Figure 6-3 Interaction between publishers, subscribes and the i*Chameleon publish/subscribe multimodal middleware

After the sensory data has been published to the algorithm component, the data is analyzed and translated into feature level data or decision level data. During this interpretation process, the algorithm components publish the result back to the controller manager, and the controller manager publishes the data again until no more algorithm components are subscribing to the data. This allows the sensory data to be continuously translated to higher-level data.

The command manager also listens to the events published to the controller manager; once an event is matched, the event would publish to the corresponding device interface manager. Finally, the device interface manager will trigger the command by notifying the output interface components.

6.3 MVC Components Classification and Development Cycle over Publish/Subscribe Paradigm

According to the development cycle discussed in Chapter 4, four processes are required to design and develop the multimodal applications. For each process, the components are defined based on the MVC architecture pattern. Figure 6-4 illustrates the classification of publish/subscribe managers along with the development cycle.

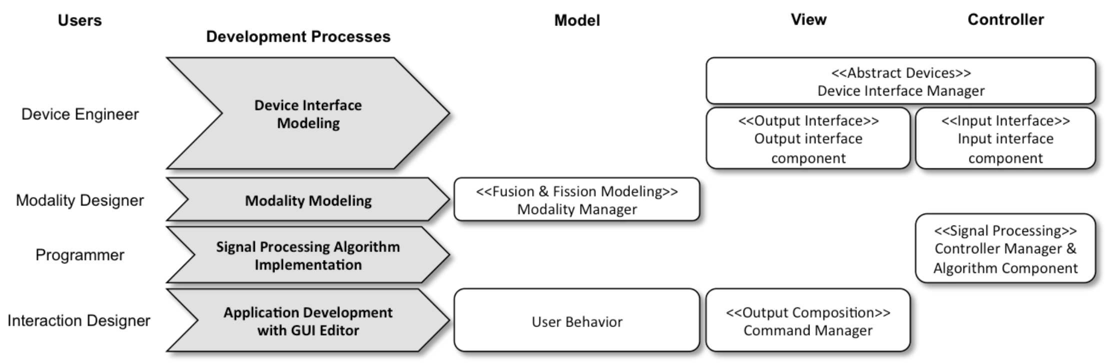

Users
Development Processes
Model
View
Controller
Device Engineer
Device Interface Modeling
<<Abstract Devices>> Device Interface Manager
<<Output Interface>> Output interface component
<<Input Interface>> Input interface component
Modality Designer
Modality Modeling
<<Fusion & Fission Modeling>> Modality Manager
Programmer
Signal Processing Algorithm Implementation
<<Signal Processing>> Controller Manager & Algorithm Component
Interaction Designer
Application Development with GUI Editor
User Behavior
<<Output Composition>> Command Manager

Figure 6-4 i*Chameleon components MVC Classification and Development Process over Publish / Subscribe Architecture

The device engineer can deploy input and output interface components to the kernel. Input interface components are classified as Controller (because they gather and model the user input information), while output interface components are classified as View (since it models the information that it needs to generate an output representation to the user). Similar to the web service architecture, device engineers are required to:

1. Model the abstract device
2. Model the input or output interface components

3. Select the modality that the input components will publish or the output components will subscribe to
4. Generate the publishers or subscribers
5. Once the publisher or subscriber is executed, a handshake protocol will be sent to the i*Chameleon platform and the components will be registered to the device interface manager.

The second process is modality modeling, which involves the modality designer and is independent of the kernel implementation. This enables the modality modeled for web services architecture to be reused. The same set of generated code will then be deployed to the modality manager as the Model component under the MVC pattern. This component is responsible for representing the behavior and status of the modalities.

In addition, programmers can reuse the XML configuration from the web services (as discussed in the previous chapter) to generate the communication layers for both publishers and subscribers. After the communication layer has been generated, the programmers can implement their own algorithms and deploy to the fusion and fission managers. These components are classified under the Controller pattern, which is responsible for receiving low-level signals and translating them to high-level data.

Finally, interaction designers can make use of the graphical user interface to customize the commands they desired. As the modeling technique is also independent of the kernel implementation, the customization methods by the non-computing experts are the same as in the web service. The users can manipulate the input and output modalities using the drag-and-drop editor and finally deploy the commands to the command manager from the i*Chameleon kernel. By definition, the commands support interaction to the system or provide feedback to the users. Therefore, these components are grouped under the View pattern.

In this chapter, we have presented the contribution of the publish/subscribe middleware to the i*Chameleon platform. This communication paradigm allows components to be dynamically plugged into and detached from the kernel. This allows the kernel to bind to different components according to the task or the content environment in order to provide the best experience for end-users in the run-time environment. We have also discussed how the different components of the publish/subscribe middleware fit into the MVC (model-view-controller) design pattern, and the classification of the components into different categories, based on their functionality, interaction style and clear delineation of development roles.

Chapter 7. Multimodal Applications

In this section, we illustrate the use of the i*Chameleon platform in two scenarios. They are: (1) Web services-based i*Chameleon for an application called Mobile DJ, which is a tangible and mobile platform for active music listening [33] and (2) publish/subscribe-based i*Chameleon for robot control in an interactive exhibition area. In particular, we demonstrate how the discussed platform facilitates the development process and support the runtime execution. For each scenario, we present the requirements of multimodal interaction application, as well as the division of labor and separation of concerns with respect to the four development processes. Finally, the runtime support and handover mechanism between components are discussed.

7.1 Mobile DJ (Web Service)

Mobile DJ is a music-listening system that allows multiple users to interact and collaboratively contribute to a single song over a social network. Our platform enables single or multiple users to actively modify music content or manipulate sound effects via a physical interface device, which supports multiple modes of interaction, and encourages user immersion into the music through exaggerated physical motions.

This system also allows collaborative and social interaction in real time, regardless of the users' location. It also allows players to browse and search sound tracks that are currently being worked on by others, and provides a channel for them to collaboratively contribute. Such collaboration requires synchronization of actions, even when the users are in different physical locations, in order to achieve pleasant effects.

In this system, it consists of a tangible musical control interface that is connected to a mobile device for signal processing and social interaction. The overview of the system architecture of *Mobile DJ* is shown in Figure 7-1. The system consists of three components: tangible musical control interface, active music listening application and i*Chameleon web services application server.

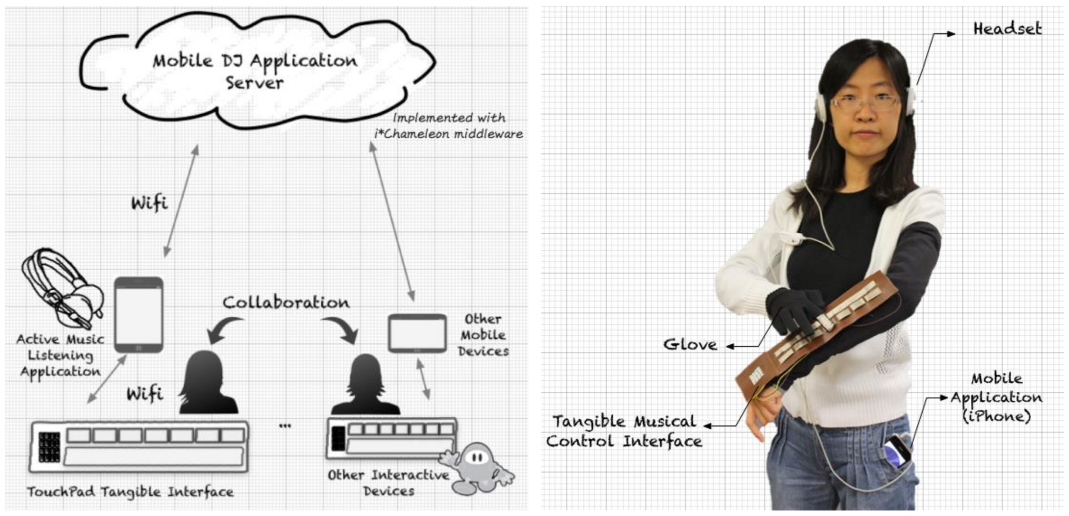

Mobile DJ Application Server
Implemented with i*Chameleon middleware
Wifi
Collaboration
Active Music Listening Application
Other Mobile Devices
Wifi
TouchPad Tangible Interface
Other Interactive Devices
Headset
Glove
Tangible Musical Control Interface
Mobile Application (iPhone)

Figure 7-1 *Mobile DJ* System Architecture (Left) *Mobile DJ* user (Right)

The tangible musical control interface (*TouchPad*) is responsible for capturing the user's interactions, which is regarded as an input widget of the multimodal application. Together with the mobile device, it forms a self-contained digital musical mixing platform. Active music listening on a single-user basis is enabled when a user connects the Musical Control Interface to a device and registers it with the application. Secondly, the active music listening application was implemented to run on iOS in Objective-C, and the BASS audio library was used to implement special sound effects such as scratching. This application is defined at the output interface in term of sound. Finally, i*Chameleon web services application server was used to integrate the tangible control interface and the music listening application together by providing a comprehensive protocol to model and handle the communication between them. Portable music players, such as an Apple iPhone, that is running an active music listening application, can be connected by standardized web services.

In the following section, we will first discus how to model the tangible musical interface and listening application following by the modality modeling. The model of algorithm and modality mapping will be discussed as well. Finally, the runtime environment and workflow are presented.

Table 7-1 Development Cycle with corresponding MVC components over Mobile DJ multimodal application

Process	Roles	MVC	Description
Device Interface Modeling	Device Engineer	View & Controller	1. Abstracted Device Component - *TouchPad* - Active Music Listening Application
		View	1. *TouchPad* (Screen) 2. Active Music Listening Application (Audio)
		Controller	1. *TouchPad* (Slidebar, potentiometer) 2. *TouchPad* (Accelerometer)
Modality Modeling	Modality Designer	Modal	1. Input Modality Modeling - Sliding - Swing (SwingUp / SwingDown) - Pressing (ShortPress / LongPress) 2. Output Modality Modeling - LED - Audio
Signal Processing Algorithm Implementation	Programmer	Controller	1. Translate continues point set into orientation (for detecting swing events) 2. Retrieving swing events from continuously pressing
Application Development	Interaction Designer	View	1. Mapping between modalities

7.1.1 Device Interface Modeling

In this multimodal application, each user uses two devices, *TouchPad* for input and *Active Music Listening Application* for output. Figure 7-2 shows the hardware implementation of the *TouchPad* interface. We require the interface to be intuitive and to support different modes of interaction with the music. From a wearability standpoint, the form factor of the interface as an armband implies that users should be able to interact with the music through movements of the other hand on the arm, or by swinging the arm. Besides, *TouchPad* also provides visual feedback. The major output device would be on the mobile application. Therefore, within the device interface modeling, we need to model two pieces of physical hardware, *TouchPad* and smart phone.

From a technical point of view, the *TouchPad* is equipped with a potentiometer on the surface of the armband. The slidebar consists of two tracks. The lower track is in one piece, while the upper track consists of segments connected with constant-valued resistors. When a user touches both the upper and lower tracks at the same time, the resistance between the tracks changes in proportion to the number of resistors that have been bypassed, thus allowing us to determine the upper track segment that is currently being touched. For aesthetics and usability, the tracks in the potentiometer are constructed from conductive fabric, which makes them congruent to textiles and garments. Besides, an accelerometer is equipped for detecting the motion of *TouchPad* and a LED matrix is provided for visual feedback. Therefore, when the device engineer models the *TouchPad*, he needs to model two input interface components and one output interface component and associated them to the same abstract device component.

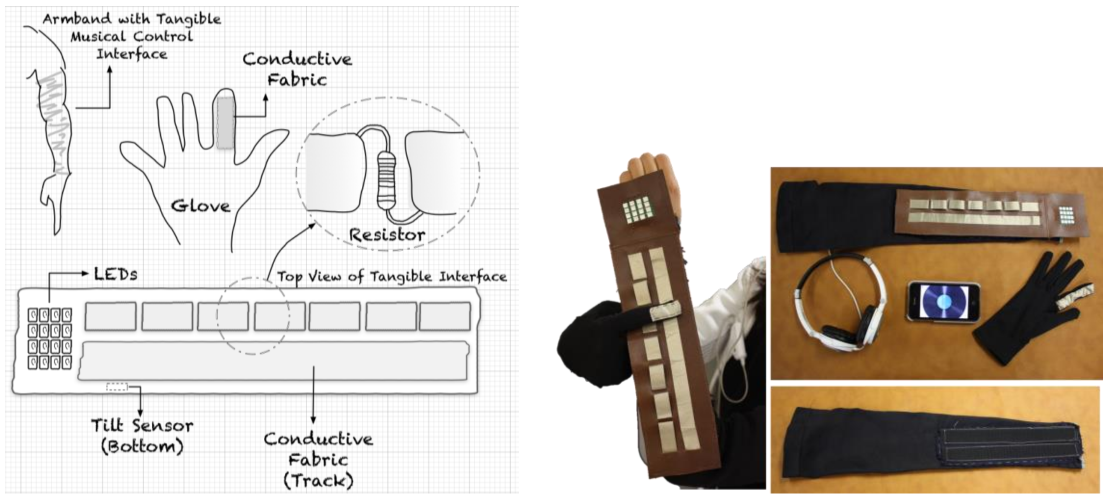

Armband with Tangible Musical Control Interface
Conductive Fabric
Glove
Resistor
LEDs
Top View of Tangible Interface
Tilt Sensor (Bottom)
Conductive Fabric (Track)

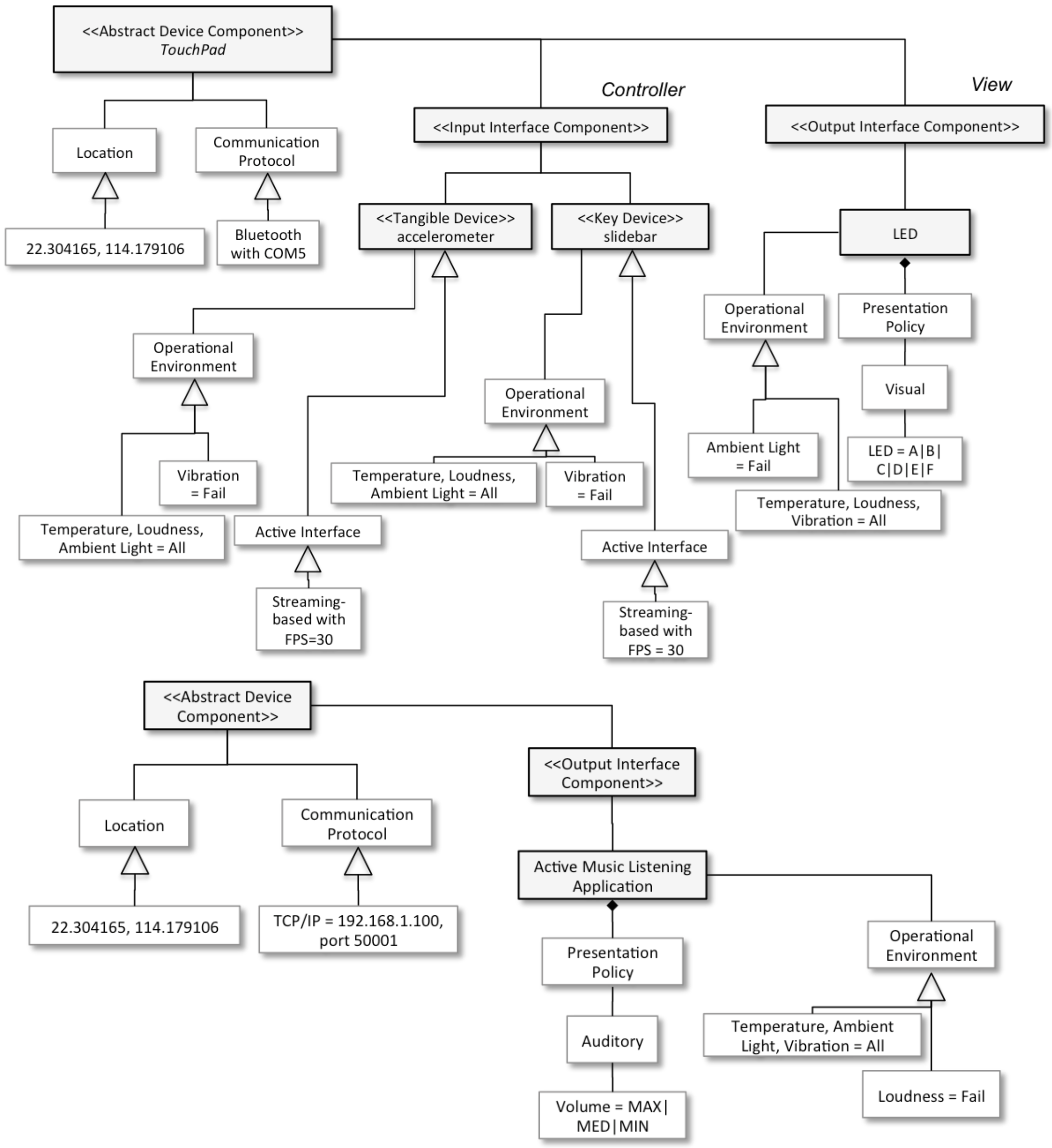
Figure 7-2 Hardware implementation of *TouchPad*

Figure 7-3 shows the modeling of both input device and output device. With reference to the modeling; two abstract device components are defined. It generalizes the common information for all input or output devices equipped in *TouchPad*, for example, the configuration of Bluetooth connection. Therefore, even if the engineers design and install more sensors on the *TouchPad*, this modeling can still be reused. Also, if the *TouchPad* is moved to another location or used by another user, the engineers are only required to change the modeling of the abstract component.

The abstract device component is associated with two input interface components and one output interface component.

(1) Input interface component for *TouchPad*: Accelerometer

- As the accelerometer will be affected by vibration, this attribute will be set to "Fail".
- The signal is defined as streaming-based with FPS 30. Therefore, the i*Chameleon kernel will expect the signal (points with x-,y-,z-coordinates) to continuously fit into the web services handler.



<<Abstract Device Component>> TouchPad
Controller
View
<<Input Interface Component>>
<<Output Interface Component>>
Location
Communication Protocol
22.304165, 114.179106
Bluetooth with COM5
<<Tangible Device>> accelerometer
<<Key Device>> slidebar
LED
Operational Environment
Presentation Policy
Visual
LED = A|B|C|D|E|F
Ambient Light = Fail
Temperature, Loudness, Vibration = All
Vibration = Fail
Temperature, Loudness, Ambient Light = All
Active Interface
Streaming-based with FPS=30
Streaming-based with FPS = 30
<<Abstract Device Component>>
<<Output Interface Component>>
Active Music Listening Application
TCP/IP = 192.168.1.100, port 50001
Auditory
Volume = MAX|MED|MIN
Temperature, Ambient Light, Vibration = All
Loudness = Fail

Figure 7-3 Input device modeling (Upper) Output device modeling (Lower)

(2) Input interface component for *TouchPad*: Slidebar

- Similar to the accelerometer, the slidebar will be affected by vibration; therefore, this attribute will be set to "Fail".

- The signal is defined as streaming-based with FPS 30.

(3) Output interface component for *TouchPad*: LED

- The operation environment is limited to dark area. So, if the context sensors reporting the users are exposed to bright lighting, the application may opt to use other feedback devices.
- The only presentation policy is configured to be visual, which provides 6 patterns for display. Those patterns are represented in token format (*A*,*B*,*C*,*D*,*E*,*F*).

(4) Output interface component for Active Music Listening Application: Audio

- Being an audio output, it cannot work well in a noisy environment while the rest of the factors would not be affected.
- The audio will be accepting streaming data from the kernel by default and the kernel can also control the volume.

Device engineers can make use of the modeling to define the XML-based properties files. Two abstract device drivers for *TouchPad* and application are generated with communication handler and widget listeners. Then each input or output components will be created as notifiers. This allows engineers to deploy all drivers to the kernel with SOAP protocol and start the widget anytime.

7.1.2 Modality Modeling

In this application, three input modalities (swinging, sliding and pressing) and two output modalities (visual and audio) are involved. Swinging, swaying and bouncing along to the beat are a common response to music, and certainly very commonly seen in active music listening experiences. The armband form factor of the tangible interface also suggested interaction through swinging or waving of the arm. Swinging is supported via the incorporation of an accelerometer in the armband, which detects movements of the user’s body. The potentiometer on the tangible interface supports a pressing interaction. The pressing is further divided into short press and long press. When an algorithm detects a number of short presses, sliding has occurred.

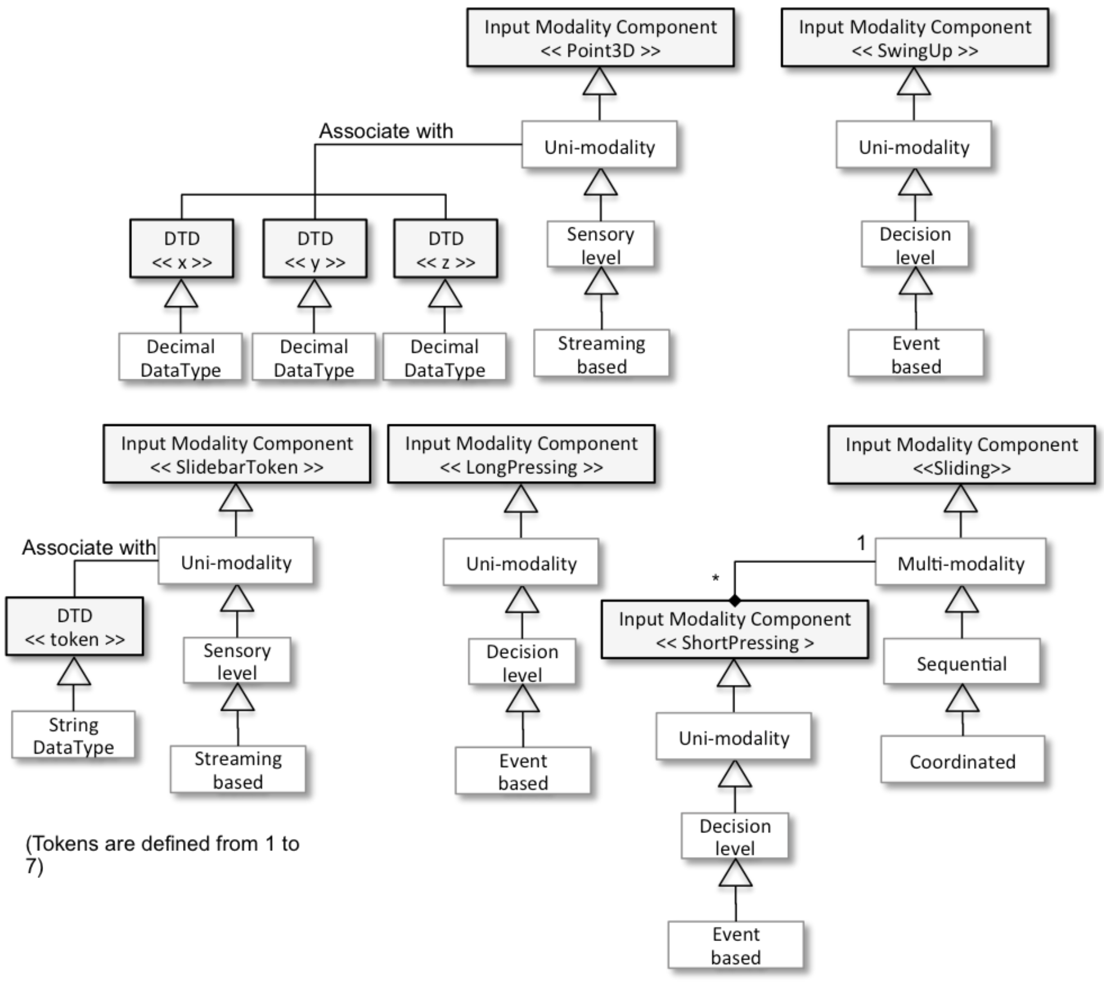


Figure 7-4 Input modalities modeling of Mobile DJ Application

On the other hand, *Mobile DJ* also includes two major output modalities, visual feedback and audio feedback. In general, in active listening, the user's interactions will receive audio feedback through the playing of the rearranged or modified music track. However, since one of the objectives of *Mobile DJ* is to support remote multi-user interaction, it was also deemed necessary to incorporate visual feedback to enhance the interaction between users. The visual feedback is supported by a matrix of multicolored LEDs that is incorporated into the tangible interface above the slidebar potentiometer. The LEDs flash in different colors to signal interactions from the collaborating partner, which provides a channel that allows a certain degree of communication and signaling, but without interfering with the experience or imposing upon the center of attention of the user. In the absence of signals from the other party, or in single-user mode, the LEDs serve as an additional form of feedback to enhance user immersion by flashing along according to the rhythm of the music.

7.1.3 Signal Processing Algorithm Implementation

Programmers need to implement the algorithm components. Within the multimodal application, two types of sensory level data have to be sent to the kernel. Therefore, programmers are required to develop algorithm components to translate *Point3D* from accelerometer to swing modality and *SlidebarToken* to pressing modality as well as analyzing the pressing to retrieve sliding modality. So, programmers are only required to define the XML-based configuration files according to the standard provided and directly integrate the components in the Modal Layer, which is discussed in Figure 5-5.

7.1.4 Application Development

In *Mobile DJ*, the pressing motion enriches the musical track by interspersing the melody with chords and different instruments, or changing the rhythm and the harmony. The interface supports both a short "tap" as well as a long "press". The "buttons" are the upper track segments. A tap on a "button" adds a chord to the music; the system will automatically generate an appropriate chord based on the key and the instrument of the currently playing song. Since we have seven "buttons", this allows the user to generate up to seven different chords. Long presses trigger a change in rhythm or harmony of the music in a similar fashion. Also, sliding is mapped to the scratching while swinging is mapped to the LED display.

Table 7-2 Modalities Mapping Table for *Mobile DJ*

Input Modality	**Token**	**Output Modality**
Sliding	1 – 7	Scratching
LongPress	1 – 7	Changing Chords (A-G)
ShortPress	1 – 2	Changing Background Music
	3 - 4	Changing Chords Style
	7	Changing Instruments
SwingUp	-	LED Display Pattern A
SwingDown	-	LED Display Pattern B

7.2 Interactive Robot Control (Publish/Subscribe)

In the second case, we apply the same development cycle with a different implementation approach through the publish/subscribe paradigm. This application is situation in an exhibition, where users are allowed to manipulate robot cars in a constrained area using different controllers. With the advances in the technology of distributed system and interactive devices, it is possible to relax the constraints imposed on the users by the limited space and the modalities provided by requiring input and output modalities to handover to each another according to the context environment.

Figure 7-5 shows the floor plan of the robotic area, which is designed to provide a multimodal controlling experience to the user. In the robotic zone, the view of the robot cars are occluded by a number of obstacles. Each robot car is equipped with a webcam, and users "see" where the cars are by viewing the camera feed. Users standing in the control zone either manipulate the car through a customized controller or with their mobile devices. Once the users activate their mobile devices, the camera feed from the robot will stream to their devices. Otherwise, the feed will be displayed on the large LED screen.

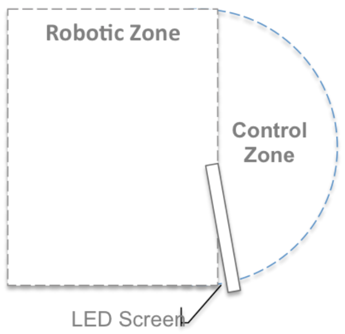


Figure 7-5 Floor Plan of the robotic zone in the exhibition hall

Table 7-3 summarizes the required effort from different users according to the proposed development processes. In our scenario, three input devices and three output devices are involved. Device engineers are required to model the abstracted device interface and its corresponding device-dependent components while modality designers are required to model the modalities. Also, an analyzer for translating incoming coordinates from tilting devices to orientation is developed by programmers and finally end-users create a mapping between modalities using the GUI editor.

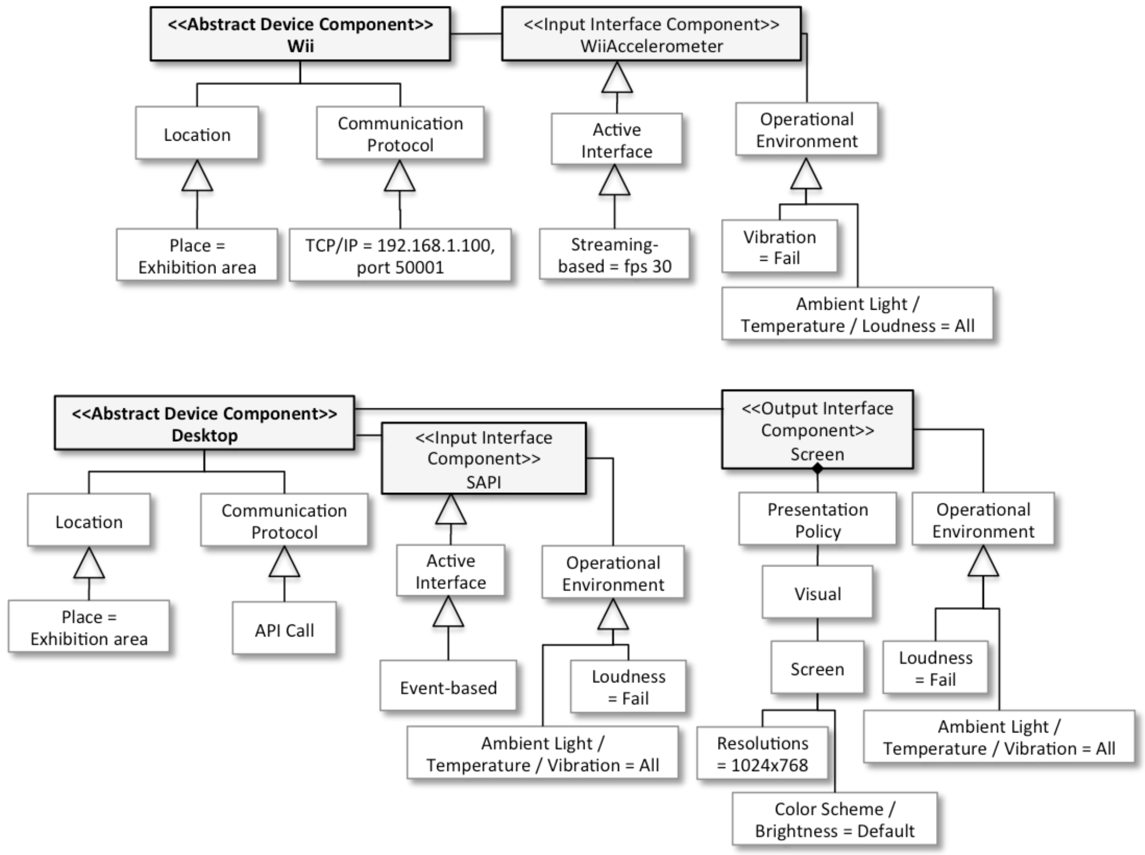
Table 7-3 Roles involved in different development processes and its corresponding MVC design pattern and description in i*Chameleon Platform

Process	Roles	MVC	Description
Device Interface Modeling	Device Engineer	View & Controller	1. Abstracted Device Component - Wii Device - iPhone - i*CATch - Desktop - xBee location detector
		View	1. Desktop Computer (Screen) 2. iPhone (Screen) 3. i*CATch Robot 4. xBee location detector
		Controller	1. Wii Controller 2. iPhone (Accelerometer) 3. Desktop with SAPI (Voice recognition engine)
Modality Modeling	Modality Designer	Modal	1. Input Modality Modeling - Tilting (front / back / left / right) - Voice input - Video input 2. Output Modality Modeling - Robot move forward / backward, turn left / right and stop - Video output
Signal Processing Algorithm Implementation	Programmer	Controller	1. Translate continues point set into orientation (for detecting tilt events) 2. Analyze xBee signal strength to determine the client location
Application Development	Interaction Designer	View	1. Mapping between modalities

7.2.1 Device Interface Modeling

In our scenario, five devices are involved and Figure 7-6 shows an example of modeling of two of them, a Wii device and a desktop computer. Both of them are located in the exhibition area and Wii controller is connected to the i*Chameleon kernel by TCP/IP communication protocol while desktop’s signal is directly transfer by API call from the kernel. Therefore, if the wireless signal strength is low, the kernel will switch to using voice control if both devices are activated. Besides, referring to the operational environment, the vibration affects the Wii accelerometer while SAPI would not work properly in a noisy environment. So, the i*Chameleon platform can make of this criteria to select the accuracy of input devices according to the context.

After translating the model into XML-based modeling language and compiling to the i*Chameleon kernel, device driver source code will be generated and the device will also be registered to the kernel. Abstract device components will be generated as an XML-based properties file while input interface components are publishers and output interface components are subscribers. The source code already includes the device description file, event listeners and communication protocol. Finally, the device engineer only needs to implement the layer between the generated blueprint and the device dependent library.



<<Abstract Device Component>> Wii
<<Input Interface Component>> WiiAccelerometer
Location
Communication Protocol
Active Interface
Operational Environment
Place = Exhibition area
TCP/IP = 192.168.1.100, port 50001
Streaming-based = fps 30
Vibration = Fail
Ambient Light / Temperature / Loudness = All
<<Abstract Device Component>> Desktop
<<Input Interface Component>> SAPI
<<Output Interface Component>> Screen
Location
Communication Protocol
Active Interface
Operational Environment
Presentation Policy
Operational Environment
Place = Exhibition area
API Call
Event-based
Loudness = Fail
Ambient Light / Temperature / Vibration = All
Visual
Screen
Resolutions = 1024x768
Color Scheme / Brightness = Default
Loudness = Fail
Ambient Light / Temperature / Vibration = All

Figure 7-6 An example to illustrate the modeling of will controller and desktop.

7.2.2 Modality Modeling

When the hardware abstraction layers are modeled, modality designers are required to model the involved modalities at different abstraction levels and the modeling is independent of the implementation methodology of the kernel. Figure 7-7 shows four models of input modalities. When the user executes the “robot forward” command, it consists of two simultaneous and coordinated modalities: a button needs to be pressed and the orientation of the handheld device needs to be tilted towards the front. Therefore, it models the input modality *ButtonPressWithTiltFront* as the bottom right model in Figure 7-7. Besides, in order to detect the orientation of the handheld device, a continuous point set with (x,y,z) coordinates, *Point3D,* are required as input. With a suitable algorithm, tilt actions (front, back left and right) can be distinguished from *Point3D* modality components. So, points and translated tilting tokens are modeled as uni-modalities. The button press event is described as a sensory level, event-based input with associated with a Boolean variable. After that, when the programmers defines the algorithm component which subscribes *Point3D* and publish *TiltFront*, those defined modalities would be related and connected.

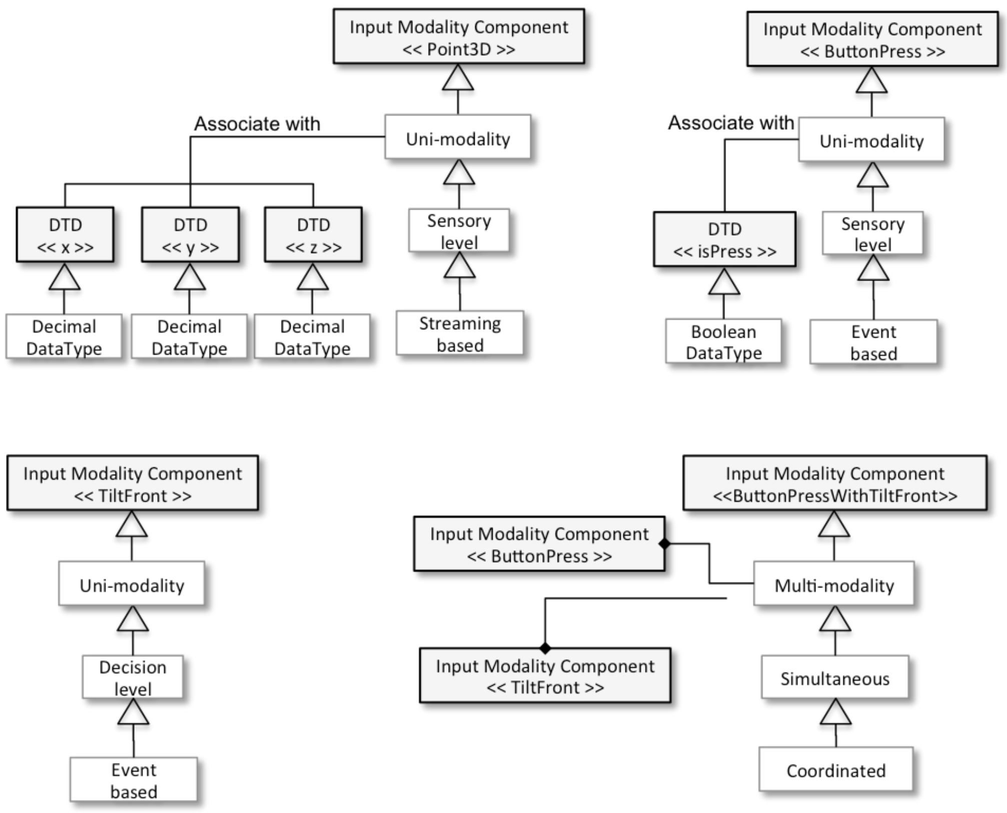


Figure 7-7 Modeling of four input modalities of interactive robots control

Similar to abstract device components, a meta-model will be represented in XML and compiled to the i*Chameleon kernel. *Point3D, TiltFront* and other related modalities will be translated into source code and necessary event notifiers and listeners will also be generated. Therefore, modality engineers are not required to contribute any coding effort within this development process.

7.2.3 Signal Processing Algorithm Implementation

While device engineers generate the structure of the hardware drivers and modality designers define the abstracted modality definition, programmers are responsible for implementing the algorithm for translating from one modality to another modality. In our scenario, we need to analyze the *Point3D* data and retrieve the orientation and the signal strength of xBee to detect the location of the client. Therefore, regardless of the hardware used, once the kernel receives the continuous *Point3D* object, it will publish them to this algorithm so as to generate tilting events.

In i*Chameleon, we developed a simple command line tool for generating the structure of the components. The following parameters are required:

- Kernel IP address: The IP address of the i*Chameleon kernel.
- Kernel Port number: The port number that match the i*Chameleon service.
- Communication protocol: TCP/IP, xBee or Bluetooth to indicate the way of communication. Similar to the abstract device component, associated packages are generated and programmers do not required inputting any coding effort.
- Subscribe modality: The signature of the required input modality.
- Publish modality: The signature of the output modality.

Syntax:

```
call KernalIP KernelPort {tcpip|xbee|bluetooth} SubscribeModality PublishModality
```

After the abstract classes have been generated for the algorithm component, programmers can continue the implementation of the needed algorithms.

7.2.4 Application Development

The final step of developing a multimodal application is mapping between the defined input modality components and output modality components. This process involves an interaction designer, who does not need to be a computing expert, using a drag-

and-drop editor to do the mapping. After the modality designer deploys the components to the i*Chameleon kernel, the components will automatically be found in the graphical editors (shown in Figure 7-8).

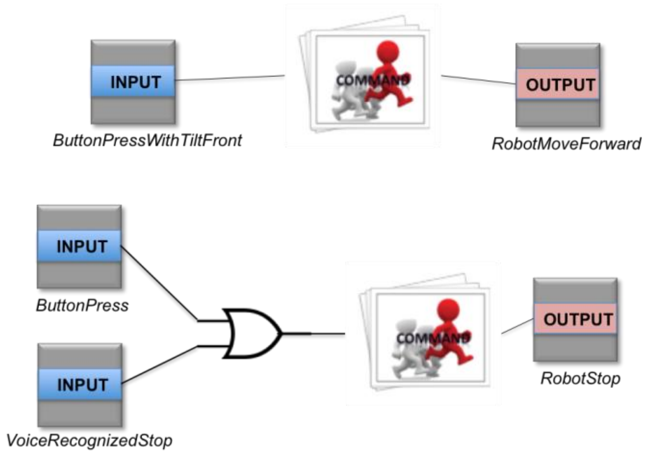


Figure 7-8 Commands created by interaction designers with using i*Chameleon graphic editor

In our case, we have four sensory level input components (*xBee Signal strength*, *Point3D*, *VideoStreamingIn* and *ButtonPress*), four feature level input components (*TiltFront*, *TiltBack*, *TiltRight* and *TiltLeft*) and four decision level input components (*ButtonPressWithTiltFront*, *ButtonPressWithTiltBack*, *ButtonPressWithTiltLeft*, *ButtonPressWithTiltRight* and *VoiceRecognizedStop*). Besides, six uni-output modality components are defined (*Location*, *VideoStreamingOut*, *LeftMotorOn*, *LeftMotorOff*, *RightMotorOn* and *RightMotorOff*) and five multi-output modalities are deployed (*RobotMoveForward*, *RobotMoveBackward*, *RobotMoveLeft*, *RobotMoveRight* and *RobotStop*). For example, instructing the robot to move forward only requires the matching of *ButtonPressWithTiltFront* as the input component to *RobotMoveForward* as the output component, as shown in Figure 7-8 (upper). Besides, Table 7-4 also shows the use of OR gate to define a command. In order to achieve our task, the following commands are created:

Table 7-4 Modalities Mapping Table for Robot Control System

Input Modality	Output Modality	Logic Gate	Description
ButtonPressWithTiltFront	*RobotMoveForward*	/	Instructs the robot to move forward
ButtonPressWithTiltBack	*RobotMoveBack*	/	Instructs the robot to move backward
ButtonPressWithTiltLeft	*RobotMoveLeft*	/	Instructs the robot to turn left
ButtonPressWithTiltRight	*RobotMoveRight*	/	Instructs the robot to turn right
ButtonPress	*RobotStop*	OR	Instructs the robot to stop

VoiceRecognizedStop			
VideoStreamIn	*VideoStreamOut*	/	Streaming the video signal from the robot to the screen

7.2.5 Runtime

The developed application can be executed from the graphic editor. When the multimodal application is executed, the kernel invokes a handshake protocol to make contact with all available devices. The devices respond by sending an XML-based configuration document to the kernel. The kernel sends its certification back to the devices to finish the handshake protocol. At the same time, if the components are distributed among the network, a handshake protocol will be issued in order to check the availability of the modalities. The initiation process is then completed.

Handover of device components – In our case, a client can either control the robot using an iPhone (remote distance) or Wii (within the exhibition area). If the client uses the iPhone while no signal is being generated from the Wii controller, the iPhone will be activated as a controller; otherwise, Wii controller will be in active mode. Therefore, same modality components, algorithm and commands can be applied to different interaction widgets. Using the signal strength from xBee, a client's location can be detected. With the device modeling, the kernel is able to determine which device should be trusted. The detected client's location also helps the kernel to decide whether to stream the video signal to the screen of the mobile device or desktop computer.

Dynamic adaption of components – With the use of publish and subscribe architectural pattern, components can be deployed to the kernel without restarting the application. Therefore, if the programmers develop another algorithm component to analyze the tilting events from raw data points, once deployment is finished, the kernel will compare the creditability of all available components and devices and choose the best subscriber.

Chapter 8. Conclusion

This chapter concludes the thesis in two sections. The first section summarizes the research work and major contributions of the thesis, followed by pointing out the limitations of the proposed system. Finally, suggestions on possible further work on the proposed development cycle and multimodal platform are made.

8.1 Summary of Research

This thesis studies a generalized methodology and challenges of developing a multimodal application. We present a comprehensive development cycle with i*Chameleon middleware to deal with the static binding between different interactive components and the tight coupling between the application programming interfaces and the application programming sequences to the user's interactions or dedicated to specific modalities. Besides, in order to decrease the complexity caused by the the multidisciplinary nature of multimodal interaction application development and to increase the efficiency of the integration of components, the model-view-controller design pattern is applied. Components are classified into different categories based on their functionality, interaction style, division of labor and separation of concerns. The clear *separation-of-concerns* realized in the design of i*Chameleon enables cross-disciplinary collaboration among device engineers, modality designers, programmers and interaction designers in order to simplify and accelerate the application development process. It enables remote controlling and collaborative work among a group of people and fast integration of components.

From the viewpoint of software engineering, meta-modeling is defined for each component and the complex development cycle is divided into four independent processes: (1) Device interface modeling – model device dependent output interface (View) and input interface (Controller), (2) Modality modeling – model the behaviors of input and output modalities (Model), (3) Signal processing algorithm implementation – develop algorithm to translate modality from sensory level to decision level and (4) Application development with GUI editor – enable non-computing users to create their own interactions.

Finally, we have successfully developed two prototype platforms of i*Chameleon to validate our contributions in two different types of technologies: web services and publish/subscribe middleware. While using the same modeling technique, different interactive components can be generated. The web service architecture provides a standard SOAP for communication among different components over the Internet while the publish/subscribe paradigm allows components to be plugged-in and detached from the kernel dynamically. Also, the publish/subscribe paradigm supports dynamic binding. During the runtime environment, the i*Chameleon kernel can bind to different components according to the task or the content environment in order to provide the best experience to the end-users.

We simulate two multimodal applications, a collaborative tangible musical interfaces and a robot car control application, to demonstrate the development process according to our suggested procedure. The result shows that the use of MVC design pattern can clearly separate the application into different components and each component can be deployed independently. It not only increases the flexibility and reusability of the interaction components, but also shows itself to be successful and effective in facilitating user-customizable multimodal interaction for a variety of environments and requirements.

8.2 Future Work

One direction for future work will be the use of modalities among different users. Once the hardware and software platform is developed, the interpretation of the input modalities according to the culture, context environment or other human-oriented factors needs to be studied. One way to approach the problem is to build a general model for “mobile interaction for kids”. This can provide an ontology for multimodal human computer interaction. This could be done by user experiments over the i*Chameleon platform, especially studying the self-customized commands. For example, novice users can be invited to design their own multimodal commands for controlling a photo viewer. Since most of the existing platforms can only provide limited commands for selection and those commands cannot be customized, i*Chameleon can overcome this limitation and offers a better user experience and rich sets of user experience data for both customers and programmers.

In term of software engineering, although the discussed methodology can facilitate the development procedures of multi-modal applications, more research is still needed to define a generalized architectural pattern or meta-model which can be applied to other domains. The challenges would be the interaction between components and users.

Last but not the least, the ultimate goal for multimodal human computer interaction is to provide human-centered methodology for controlling applications. Tasks can be transferred from one device to another while commands can be adaptively changed according to the context and the processes need to be invisible for the customers. This facilitates the development of multimodal interaction and human computer interaction.

References

[1] A. Jaimes, N. Sebe, D. Gatica-Perez, T.S.H. (Eds. . 2007. Special Issue on Human-centered Computing. *IEEE Computer*. 40, 5 (2007).

[2] Altherr M, Erzberger M, M.S. 1999. iBus—a software bus middleware for the Java platform. *Workshop on Reliable Middleware Systems* (1999), 43–53.

[3] Application Develop- ment Software: 2007. *http://www.idc.com/getdoc.jsp?containerId=IDC_P644*.

[4] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. 2010. A view of cloud computing. *Commun. ACM*. 53, 4 (Apr. 2010), 50–58.

[5] Atrey, P.K., Hossain, M.A., El Saddik, A. and Kankanhalli, M.S. 2010. Multimodal fusion for multimedia analysis: a survey. *Multimedia Systems*. 16, 6 (Apr. 2010), 345–379.

[6] B'Far, R. 2004. Multichannel and Multimodal User Interfaces. *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. New York : Cambridge University Press.

[7] Bolt, R.A. 1980. "Put-that-there": Voice and gesture at the graphics interface. *ACM SIGGRAPH Computer Graphics*. 14, 3 (Jul. 1980), 262–270.

[8] Bouchet, J. and Nigay, L. 2004. ICARE: a component-based approach for the design and development of multimodal interfaces. *CHI '04 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2004), 1325–1328.

[9] Bouchet, J., Nigay, L. and Ganille, T. 2004. ICARE software components for rapidly developing multimodal interfaces. *Proceedings of the 6th international conference on Multimodal interfaces* (New York, NY, USA, 2004), 251–258.

[10] Burbeck, S. 1987. *Applications Programming in Smalltalk-80: How to use Model-View- Controller (MVC).* Softsmarts, Inc.

[11] Clarkson, J. D., and Yi, J.L. 1996. A synthetic forces tactical training system for the USMC commander. *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation* (Orlando, Florida, 1996), 275–281.

[12] 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. *Proceedings of the fifth ACM international conference on Multimedia* (New York, NY, USA, 1997), 31–40.

[13] Cohen, P.R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L. and Clow, J. 1997. QuickSet: multimodal interaction for simulation set-up and control. *Proceedings of the fifth conference on Applied natural language processing* (Stroudsburg, PA, USA, 1997), 20–24.

[14] Cohen, P.R. and McGee, D.R. 2004. Tangible multimodal interfaces for safety-critical applications. *Commun. ACM*. 47, 1 (Jan. 2004), 41–46.

[15] Concepts | OpenInterface Platform: *http://www.openinterface.org/platform/concepts*.

[16] Cooper, S., Dann, W. and Pausch, R. 2003. Teaching objects-first in introductory computer science. *ACM SIGCSE Bulletin*. 35, 1 (Jan. 2003), 191.

[17] Cugola, G., Margara, A. and Migliavacca, M. 2009. Context-aware publish-subscribe: Model, implementation, and evaluation. *2009 IEEE Symposium on Computers and Communications*. (Jul. 2009), 875–881.

[18] Dragicevic, P. and Fekete, J.-D. 2004. Support for input adaptability in the ICON toolkit. *Proceedings of the 6th international conference on Multimodal interfaces* (New York, NY, USA, 2004), 212–219.

[19] Dumas, B., Lalanne, D. and Oviatt, S. 2009. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. *Human Machine Interaction*. D. Lalanne and J. Kohlas, eds. Springer Berlin / Heidelberg. 3–26.

[20] Dupuy-chessa, S., Bousquet, L. Du, Bouchet, J. and Ledru, Y. 2005. Test of the icare platform fusion mechanism. *In DSVIS05, LNCS* (2005).

[21] Elouali, N., Rouillard, J., Le Pallec, X. and Tarby, J.-C. 2013. Multimodal interaction: a survey from model driven engineering and mobile perspectives. *Journal on Multimodal User Interfaces*. 7, 4 (Jun. 2013), 351–370.

[22] EMMA: Extensible MultiModal Annotation markup language: *http://www.w3.org/TR/emma/#s4.1.2*.

[23] Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.-M. 2003. The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 2 (Jun. 2003), 114–131.

[24] Fernandes, V., Guerreiro, T., Araújo, B., Jorge, J. and Pereira, J. 2007. Extensible middleware framework for multimodal interfaces in distributed environments. *Proceedings of the ninth international conference on Multimodal interfaces - ICMI '07*. (2007), 216.

[25] V. Fernandes, T. Guerreiro, B. Araújo, J.A. Jorge, and J.P. 2007. Architectural patterns revisited – a pattern language. *International Conference on Multimodal Interfaces* (2007), 216–219.

[26] Fritz, G., Seifert, C., Luley, P., Paletta, L., Almer, A., Attewell, J. and Savill-smith, C. 2004. Mobile vision for ambient learning in urban environments. Learning and Skills Development Agency.

[27] Glass, J., Weinstein, E., Cyphers, S., Polifroni, J., Chung, G. and Nakano, M. 2005. A Framework for Developing Conversational User Interfaces. *Computer-Aided Design of User Interfaces IV*. R. Jacob, Q. Limbourg, and J. Vanderdonckt, eds. Springer Netherlands. 349–360.

[28] Greenberg, S. and Fitchett, C. 2001. Phidgets: easy development of physical interfaces through physical widgets. *Proceedings of the 14th annual ACM symposium on User interface software and technology - UIST '01* (New York, New York, USA, 2001), 209.

[29] Harbor Research, I. 2009. *Pervasive Internet & Smart Services Market Forecast*.

[30] Horst.Roessler Juergen.Sienel, W.W.J.H.M.K., Rssler, H., Sienel, J., Wajda, W., Hoffmann, J. and Kostrzewa, M. 2001. Multimodal Interaction for Mobile Environments. *International Workshop on Information Presentation and Natural Multimodal Dialogue* (Private Network Department; Alcatel SEL AG Research and Innovation, 2001).

[31] Jaimes, A. and Sebe, N. 2007. Multimodal human-computer interaction: A survey. *Comput. Vis. Image Underst. Computer Vision and Image Understanding*. 108, 1-2 (Oct. 2007), 116–134.

[32] Johnston, M. 2009. Building multimodal applications with EMMA. *Proceedings of the 2009 international conference on Multimodal interfaces - ICMI-MLMI '09* (New York, New York, USA, 2009), 47.

[33] Kenneth W.K. Lo, Chi Kin Lau, Michael Xuelin Huang, Wai Wa Tang, Grace Ngai, S.C.F.C. 2013. Mobile DJ: a Tangible, Mobile Platform for Active and Collaborative Music Listening. *Proceedings of the 13th International Conference on New Interfaces for Musical Expression* (Daejeon and Seoul, Korea Republic, 2013), 217–222.

[34] Kim, W., Kim, S.D., Lee, E. and Lee, S. 2009. Adoption issues for cloud computing. *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia* (New York, NY, USA, 2009), 2–5.

[35] Krahnstoever, N., Kettebekov, S., Yeasin, M. and Sharma, R. 2002. A Real-Time Framework for Natural Multimodal Interaction with Large Screen Displays. *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces* (Washington, DC, USA, 2002), 349–.

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

[37] Li, K. and Hudak, P. 1989. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*. 7, 4 (Nov. 1989), 321–359.

[38] Lo, K.W.K., Tang, W.W., Leong, H.V., Chan, A., Chan, S. and Ngai, G. 2012. i∗Chameleon: A unified web service framework for integrating multimodal interaction devices. *2012 IEEE International Conference on Pervasive Computing and Communications Workshops* (Lugano, Mar. 2012), 106–111.

[39] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M. 2004. Scratch: a sneak preview. *Proceedings. Second International Conference on Creating, Connecting and Collaborating through Computing, 2004.* (2004), 104–109.

[40] McCowan, L., Gatica-Perez, D., Bengio, S., Lathoud, G., Barnard, M. and Zhang, D. 2005. Automatic analysis of multimodal group actions in meetings. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. 27, 3 (Mar. 2005), 305–317.

[41] Ngai, G., Chan, S.C.F., Ng, V.T.Y., Cheung, J.C.Y., Choy, S.S.S., Lau, W.W.Y. and Tse, J.T.P. 2010. i*CATch: a scalable plug-n-play wearable computing framework for novices and children. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10* (New York, New York, USA, 2010), 443.

[42] Nigay, L. and Coutaz, J. 1993. A design space for multimodal systems. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93* (New York, New York, USA, 1993), 172–178.

[43] Noceti, N., Caputo, B., Castellini, C., Baldassarre, L., Barla, A., Rosasco, L., Odone, F. and Sandini, G. 2009. Towards a Theoretical Framework for Learning Multi-modal Patterns for Embodied Agents. *Image Analysis and Processing – ICIAP 2009*. P. Foggia, C. Sansone, and M. Vento, eds. Springer Berlin / Heidelberg. 239–248.

[44] Nunes, N.J. and Falcão, J. 2001. Wisdom – A UML Based Architecture for Interactive Systems 2 Software Architecture Models for Interactive Systems. *Interactive Systems Design, Specification, and Verification*. Springer Berlin / Heidelberg. 191–205.

[45] Obrenovic, Z. and Starcevic, D. 2004. Modeling multimodal human-computer interaction. *Computer*. 37, 9 (2004), 65–72.

[46] Oulasvirta, A., Rattenbury, T., Ma, L. and Raita, E. 2011. Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing*. 16, 1 (Jun. 2011), 105–114.

[47] Oviatt, S. 1997. Multimodal interactive maps: designing for human performance. *Hum.-Comput. Interact.* 12, 1 (Mar. 1997), 93–129.

[48] Oviatt, S. 2003. The human-computer interaction handbook. J.A. Jacko and A. Sears, eds. L. Erlbaum Associates Inc. 286–304.

[49] Saha, D. 2003. Pervasive computing: a paradigm for the 21st century. *Computer*.

[50] Salber, D., Blandford, A., May, J. and Young, R.M. 1995. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. *INTERACT* (1995), 1–7.

[51] Satyanarayanan, M. 2001. Pervasive computing: vision and challenges. *IEEE Personal Communications*. 8, 4 (2001), 10–17.

[52] Serrano, M., Nigay, L., Lawson, J.-Y.L., Ramsay, A., Murray-Smith, R. and Denef, S. 2008. The openinterface framework: a tool for multimodal interaction. *CHI '08 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2008), 3501–3506.

[53] Shen, J. and Pantic, M. 2009. A software framework for multimodal humancomputer interaction systems. *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on* (2009), 2038–2045.

[54] Shen, J., Shi, W. and Pantic, M. 2011. HCI 2 Workbench: A development tool for multimodal human-computer interaction systems. *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on* (Mar. 2011), 766–773.

[55] Simpson, R., Lopresti, E., Hayashi, S., Nourbakhsh, I. and Miller, D. 2004. The smart wheelchair component system. *Journal of rehabilitation research and development*. 41, 3B (May 2004), 429–42.

[56] Sinha, A.K. and Landay, J.A. 2003. Capturing user tests in a multimodal, multidevice informal prototyping tool. *Proceedings of the 5th international conference on Multimodal interfaces* (New York, NY, USA, 2003), 117–124.

[57] Tang, W.W., Lo, K.W.K., Chan, A.T.S., Chan, S., Leong, H.V. and Ngai, G. 2011. i*Chameleon: a scalable and extensible framework for multimodal interaction. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems* (New York, NY, USA, 2011), 305–310.

[58] Tang, W.W.W. i * Chameleon : A Scalable and Extensible Framework for Multimodal Interaction. 1–6.

[59] Turk, M. 2014. Multimodal interaction: A review. *Pattern Recognition Letters*. 36, (Jan. 2014), 189–195.

[60] Westeyn, T., Brashear, H., Atrash, A. and Starner, T. 2003. Georgia tech gesture toolkit: supporting experiments in gesture recognition. *Proceedings of the 5th international conference on Multimodal interfaces - ICMI '03* (New York, New York, USA, 2003), 85.

[61] XHTML+Voice Profile 1.0: *http://www.w3.org/TR/xhtml+voice/*.

[62] Yim, J., Qiu, E. and Graham, T.C.N. 2008. Experience in the design and development of a game based on head-tracking input. *Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play '08* (New York, New York, USA, 2008), 236.