Advances in Software Architecture Design Applied to Human Computer Interaction Processing

Jérôme Lard[1,2], Célestin Sedogbo[1], Pascal Bisson[1]

(1)THALES Research & Technology,
Domaine de Corbeville, 91404 Orsay Cedex, France
(2)LRI, Laboratoire de Recherche en Informatique
Université Paris Sud XI, 91400 Orsay
firstname.lastname@thalesgroup.com

Abstract. As today's means of interaction become more and more various and sophisticated, interaction demands the integration of heterogeneous modalities, such as voice, gesture, graphics and animation, as well as appliances, such as traditional laptop and desktop workstations, mobile phones, Personal Digital Assistants (PDA), tablet PC, etc. One solution for providing users with wider access to existing systems and for enhancing user-friendliness of existing interfaces is to design intelligent interaction systems that dynamically adapt to the interaction environment and react appropriately in various contexts of use, without implying any modification to the core application. We will illustrate here the concept of Human Interaction Container (HIC) which introduces an important shift in the field of Human-Computer Interaction (HCI), moving from an application-centric to a user-centric perspective, through the adoption of a service-oriented view of application and user interface capabilities.

1 Introduction

The HIC aims at encapsulating all software components dedicated to user-system interaction management into a context-aware and context-sensitive container enacting as a mediator between the application services and the presentation services. This container is designed so to ease the logical separation between application, interaction and presentation, and handle all the interaction processes enabling an application and its various user interfaces to communicate with each other.

It offers application-independent and interface-independent interaction services which support intelligent adaptive interaction. These generic interaction services include dialogue processing, task and activity planning, user adaptation, multi-modality management and multimedia presentation generation.

Our approach aims at being able to dynamically generate or adjust a presentation which best fits the user's expectation according to its own context and its current task. Also, contextual information will be assessed to constrain the design or re-design of a presentation to take into account some user's data at runtime.

However, this work does not target man-machine interface design but rather interactive dialogue between the user and the computer itself. We define this dialogue as an inter-process communication management between applications of different natures. Hence, we consider that most of the current computing architecture is based on a three tier paradigm: *database, application* and *presentation* and that each of this tier can be considered as an application itself: *database application, data processing application* and *presentation application*. We believe that bringing more intelligence to the man-machine interaction resides in the introduction of a new tier between the application and the presentation layers. This is illustrated in **figure 1** thereafter. One could think that providing intelligent interaction is to implement another application dedicated to this task. We decided not to do so and to study a new kind of distributed architecture, proposing high level services oriented toward the interaction for all applications and clients connected to it. We can say that the interaction management provided by this platform is applied to *mutual application understanding*.

Therefore, our choice is to put the interaction at the center of the exchange between the system and the user. This is the solution we provide to answer the increasing system and software complexity. A system is easy to use when it gives a limited number of functions. Nonetheless, such a system is very powerful in accomplishing what it was designed for. On the other hand systems which try to be as generic as possible tend to offer too many functions. At first, they may seem easy to use but when using them for more intricate tasks they easily get too complicated to work with. There is no doubt that the increasing number of users will lead to more diversity in the software industry. People will want to access services which may have to be created on demand. The next major trend in the software industry is expected to be oriented towards services development [1].

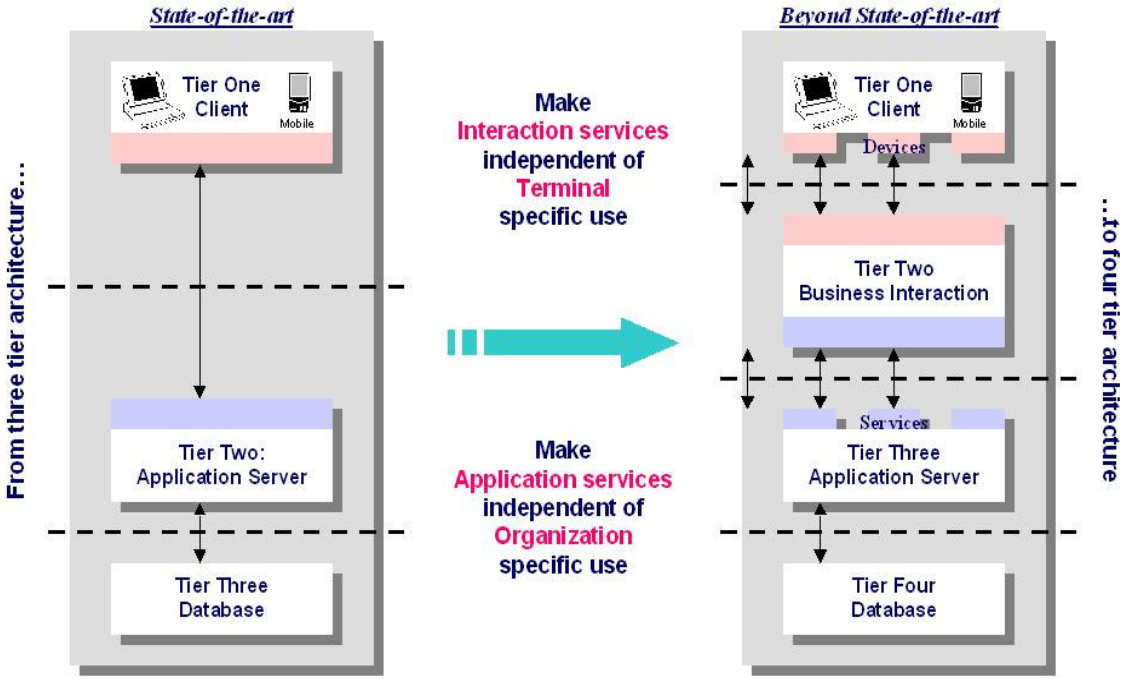

Figure 1. HIC Functional Architecture

2 Human-Computer Interaction Overview

In order to define generic interaction services we need to be able to sharply isolate what the specificities of the HCI domain are. Unfortunately the definition of the HCI field is very wide. We borrowed a definition from the ACM committee to illustrate this: Human-computer interaction is "*a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them*".

Starting from this definition we were able to go back to the core concepts of man-machine interaction. Human is used to interact with his environment by direct and physical manipulation. When the environment becomes more complex, we usually tend to find solutions to overcome natural and physical limitations to stay in control. The perception of the complexity of such an environment and the kind of control that we want to operate can be assisted by a computing system.

In more and more situations the computer becomes the intermediate between us and our personal environment [2]. Unfortunately, computers do not provide direct manipulation like other types of artifacts. (books for example allow us to interact with our inward whereas computers should allow us to augment our vision of the outside world. Books help us think about our inside space. This is a type of disappearing interface where the actual object which we manipulates disappears to leave more space for our thoughts). This is the trend in pervasive computing in which computerized appliances become our preferred interface for everyday tasks. Thus, three types of interaction can be characterized. The direct perception/action (PA) is called *human environment interaction*. When introducing a computer in the PA loop, two types of interaction paradigms may appear: *human-computer interaction* and *sensor-actuator interaction*. Sensor-actuator interaction gives an augmented view to the user and the capacity of intelligent PA on the environment through the computer system.

Bi-directional interaction between the man and the computer help us define what the investigation domains of the HCI are. Following the ACM committee categories, we focus on four major domains: *Use and Context, Development Process, Human* and *Computer*. The Computer domain is concerned with Input/Output Devices, Dialogue Techniques, Dialogue Genre, Computer Graphics and Dialogue Architecture. The last category is the one on which we will mainly focus through the rest of this article.

Originally, the goal of computer science was to accompany us in our tasks considering our own limits either physical or intellectual. Therefore, one important issue of the HCI is to help the user to better understand his environment providing augmented situation awareness. This is what we called augmenting human intellect. This doesn't quite fit into the augmented reality paradigm since the goal seems so "simple": providing the right information, at the right time and to the right person.

3 Intelligent Architecture Design

3.1 Human-Interaction Container

In order to allow intelligent and adaptive information presentation we decided to encapsulate all user interactions in a container which at runtime represents an electronic avatar of the real user. As a consequence the HIC carries the interaction logic representing the dialogue between the user and the system. The container role is to manage at runtime the existing links between the developed interaction components. For instance, interaction components are developed on the basis of the generic architecture shown in **figure 2**. We will implement interaction components such as *dialogue* and *multimodal processing*. These components are later linked together with the help of business rules. Nevertheless their mapping can evolve at runtime to provide a more consistent and more transparent interaction through the task and with different devices. The implementation of this approach is based on current state of the art architecture. We have studied and adopted a distributed component/container based architecture. Nevertheless the idea of reconfiguring the internal logic deployed by a container at runtime has not yet been provided in state of the art implementations of component/container architecture. Consequently we had to design our own execution environment based on both state of the art specifications and our own requirements allowing dynamic service calls, asynchronous communication between components and intelligent management of human-system communication through high level services.

3.2 HIC Architecture

A way to implement an intelligent and adaptive interaction model like the one described above is to make the application model independent from the input modalities which we use and also independent from the devices. We have to design our approach in two steps.

First we need to identify what services should be provided by the characterized interaction. Then we need to invent models which represents business rules and user's tasks in order to help the interaction process to provide intelligent adaptation to the various context of use.

We made the choice to represent and implement this approach through the adoption of a new layer introduced in classic middleware architecture. The representation chosen clearly states a strong independence on each side of our business interaction layer and decouples presentation logic from application logic. The first advantage of this separation is to allow application developers to focus on functions provided by their applications and not on the rendering. It allows the HMI (Human Machine Interface) developers to focus on the development of easy to use presentation artifacts close to mental representation of a business model. Now it also demands to be able to find interaction specialists to develop the interaction layer which gathers a broad spectrum of scientific expertise which is not related to HMI or to business application functions.

The architecture presented in **figure 1** is illustrating the introduction of a fourth tier into standard three tier architecture. This new layer called *Business Interaction Tier* contains some *Business Interaction Logic*. This approach is model based [3][4]. The models we are considering relevant for the business layer are the following ones: user model, task model and a new concept called *Business Interaction Pattern* (BIP). Other models are used for the purpose of Input/Output presentation such as terminal models but are not used directly by the dialogue.

User Model is referencing data about the user ranging from his role into the work organization, to his access rights to knowledge database, to his device preferences. There is no information here about the user's task. The *User Profile* is gathering user identity for a computing context.

The *Task Model* was defined by [5] as the *user computing intention*. A task should be platform-independent and application-independent. If we take a text editing application, a user needs to be able to accomplish his task, editing the text, without impediment from the computing platform or the text editing tool he is using. Then, his task can be described as a multiple step operation in which the user will successively { start the text editing application, browse a folder, choose a file to edit, open the file, edit the file, save the file, close the file, terminate the text editing application }. In the context of our laboratory, the end users have very specific needs for computer assisted tasks. It means that their activity is not random in any way. They have to follow extremely rigid business rules.

For this we designed a way to specify business rules into BIP. BIP can be seen as a key that we need to have in order to use the task model. For instance, if we go back to the text editing example given above the task described is very generic and can be applied for many business rules definitions. If I use this task model I will not have to create very complicated business rules to use it. But if we address a police officer's every day work, he might have to be reminded to print the police report he just edited in order to have it signed by witnesses.

The models briefly presented here are used by the *dialogue processor* to provide adapted business interaction for users. The **figure 2** below presents a generic multi modal architecture in which the dialogue processing is represented.

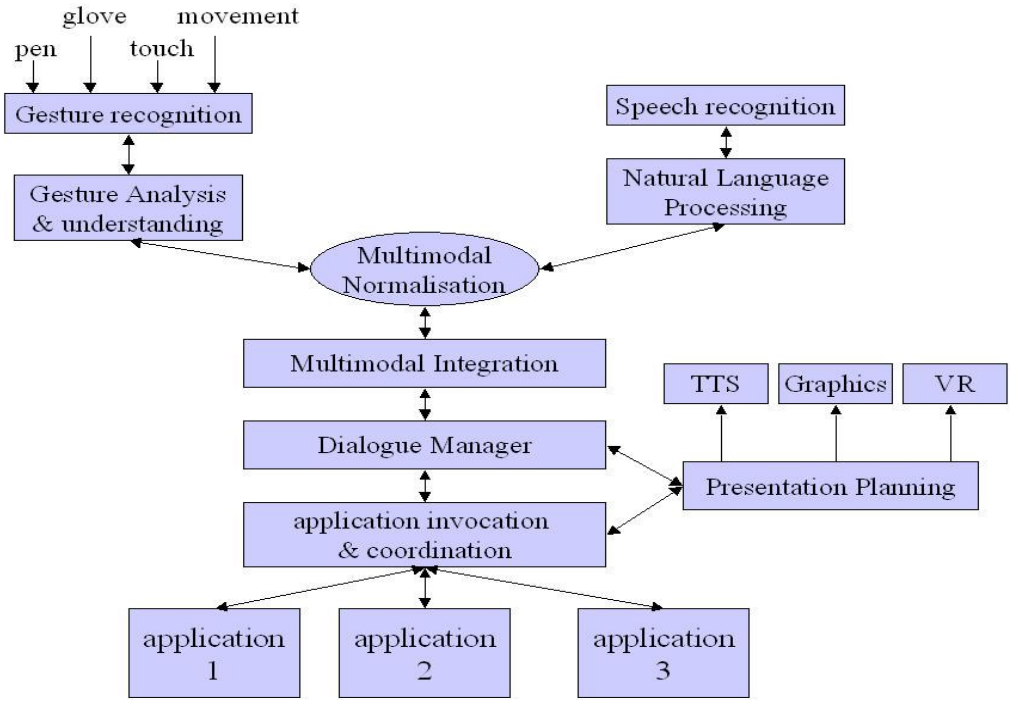

Figure 2. Generic Multimodal Architecture

3.3 Interaction Middleware

3.3.1 Introduction

To begin this part we would like to give an agreed upon definition of a middleware: "*a middleware is a connectivity software which consists of a set of enabling services that allow multiple processes running on one or more machines to interact over the network*". If we stop our study here then we might consider a lot of software fitting into a middleware definition. We might very well say that UNIX is a middleware.

Another definition says that a middleware is a piece of software usually described as a conversion or a translation layer between two processes. This definition is closer to what we intend to do with a middleware. To understand this better we have to go a little bit further into the three identified middleware types: System Middleware, Explicit Middleware, Implicit Middleware.

System Middleware corresponds to the conversion layer which exists between raw data produced on a specific computing platform and the operating system using it.

Explicit Middleware encompasses middleware systems that we most often see today. We can identify this layer as the intermediary layer between an operating system and any application above. This kind of middleware provides an automation of non functional processes previously embedded in the application itself. CORBA (Common Request Broker Architecture) and RMI (Remote Method Invocation) are two examples of explicit middleware in use today. When we study hierarchical organization of explicit Middleware we can note that they are often subdivided into three distinctive tiers: *Distribution Services layer, Common Services layer* and *Domain-Specific Services layer* [6].

Implicit Middleware is identified by the process of intermediation or interpretation between a business application and the application of presentation associated with it. Java offers such a middleware since objects created with the Java Language can be analyzed and displayed using common Java API (application programming interface) for graphic representation (AWT or SWING libraries). Consequently we developed our own implicit middleware between presentation and business logic. This implicit middleware that we called *interaction middleware* is more than just a layer providing a conversion between two processes. This layer

analyzes and tries to interpret what the intention of a user is and what kind of information an application or a sensor wants to present to the user connected to the system.

3.3.2 Interaction Middleware Architecture Implementation

We implemented our middleware infrastructure through a collaboration with CSLI at Stanford University. They developed what we called a meta operating system. iROS is a middleware based on the tuple space of IBM which is a Message Oriented Middleware (MOM). This middleware has been used to raise hardware events to a higher level of abstraction so that, for instance, a mouse event can be associated with a voice command. We used this middleware as a basis for interaction services development and integration. We also began our design looking at web-services middleware architecture. The **figure 3** is a layered representation of our architecture on the right in perspective of an architecture recommended for web-services interaction. We may try to find a link between this kind of representation and the ISO seven layers which does not always correspond.

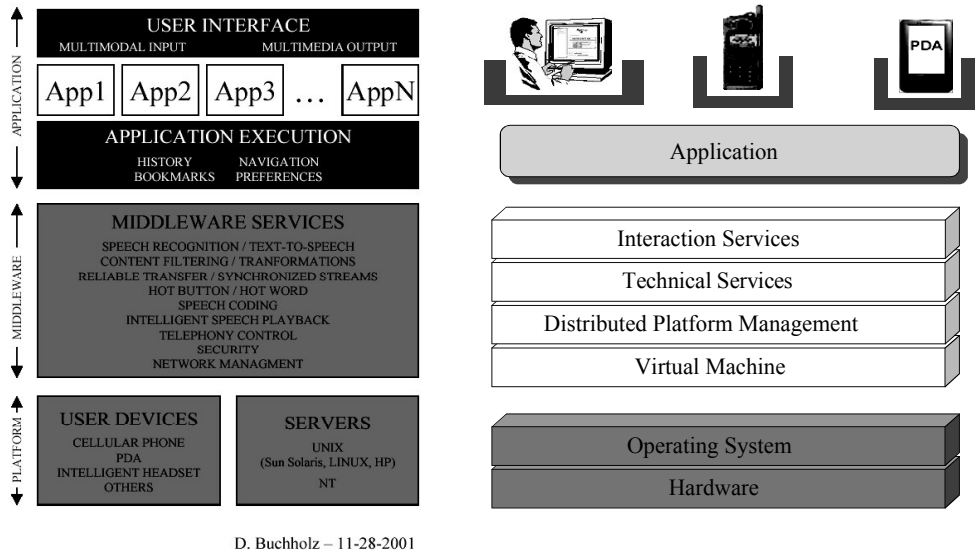

Figure 3. Middleware Architecture Inspired from web-services: on the left D. Buchholz, on the right our architecture

We completed an architecture in which three layers of services are appearing:

- Standard Services; Technical Services ;Interaction Services

We will describe briefly this architecture now. *Standard Services* are basic distribution middleware services. For that we relied mainly upon iROS native capacities for distribution and service declaration. The *Technical Services* layer provides service support for higher level services, *Interaction Services*. This last layer is developed below in **figure 4**.

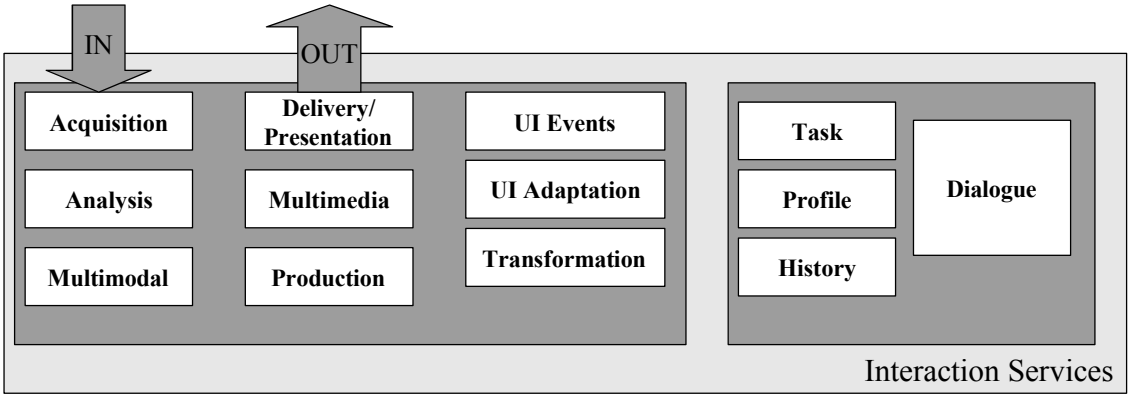

Figure 4. Interaction Services Layer

In this upper layer, we separated our work into two groups. One group of services is taking care of the multimodality aspects of the processing while the second one is dedicated to interactive dialogue processing. The services described are understood as if they were an interface to given functionalities which are implemented in different ways inside the HIC. The container will use the interface provided by this layer shared by all the users, interpret it as a component specification and implement its own version of it for a specific context and user interaction.

3.3.3 Interaction Middleware at Runtime

Based on the deployed architecture represented below we were able to offer an interaction platform which is composed of the following elements: an interaction middleware with generic interaction services, interaction components as those described above and at last an interaction container.

It is now possible to imagine deploying a business model based on one or more applications and several devices for which it is possible to choose diverse interaction modes.

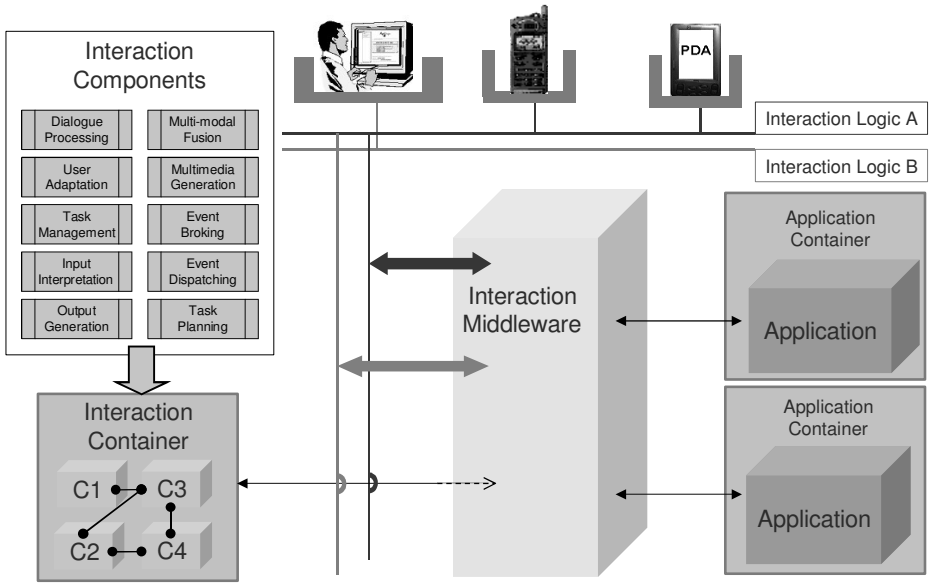

Figure 5. Interaction Middleware deployed

Conclusion

This platform has been implemented. We tested our interaction processing paradigm and architecture on an operational application dedicated to air traffic management. To test our middleware, we integrated a fully functional application of this field for which only one interface was developed. We were able to enrich the interaction that was provided at first as we added voice recognition and text to speech functions. We also tested automatic task and activity planning. Not only could the interaction become richer if the application was not modified but the fact that we had to model the task and the business rules of the users allowed us to manage high level interaction similar to what we worked on a few years ago: natural language processing and understanding. It means that not only is a user command acknowledged and executed but it is also understood by the dialogue processor which may transform this order for the target application. The same process is used the other way to interpret a message coming from an application. This message can be augmented with other information that the dialogue processor would take the initiative to complete for the best user situation awareness.

References

[1]David Chappell & Associates., *The Service Oriented World*. September 2003

[2]Thomas Kirste, Stefan Rapp, *Architecture for Multimodal Assistance Systems*. International Status Conference, HCI, October2001

[3]Montserrat Sendin, Jesús Lorés, Fransisco Montero and Victor López, *Towards a Framework to Develop Plastic User Interfaces*. Mobile HCI, September 2003

[4]David Thévenin, Thèse de Doctorat de l'Université Joseph Fournier de Grenoble, "Adaptation en Interaction Homme-Machine : le cas de la plasticité", Genoble 2001

[5] Zhenyu Wang, David Garlan, *Task Driven Computing*. May 2000

[6]Douglas C. Schmidt, Frank Buschmann,*Patterns, Frameworks, and Middleware: Their Synergistic Relationships*. 2003

Jan Gulliksen, Bengt Goransson, Inger Boivie, Stefan Blomkvist, Jenny Persson and Asa Cajander, *Key Principles for User-Centred Systems Design*. Behavior & Information Technology, nov-dec 2003, pages 397-409

[7]Kurt Geihs, *Middleware Challenges Ahead*. IEEE Computer Society, 2001

[8]Mauro Marinilli, *The Theory Behind User Interface Design*. November 2002

[9]John Carroll, *HCI Models, Theories and Frameworks: Towards a multidisciplinary Science*.

[10]Mike Kuniavsky, *Observing the user experience: a practitioner's guide to user research*. Morgan Kaufmann Series.

[11]Jan Brochers, *A pattern approach to interaction design*.

[12]Donald A. Norman, *The design of everyday things*.