Two Case Studies of Software Architecture for Multimodal Interactive Systems: Voice-Paint and a Voice-enabled Graphical Notebook

Arno Gourdol, Laurence Nigay, Daniel Salber and Joëlle Coutaz
Université Joseph Fourier
Laboratoire de Génie Informatique
B.P. 53 X — F-38041 Grenoble Cedex — France
gourdol@imag.fr

Abstract

This paper discusses software architectures of multimodal systems. The recent availability of new input technologies brought a whole new type of systems, able to support communication with the user through multiple interaction channels. Multimodal systems that allow modalities to be combined seem to be the most promising in the field of multimodal interaction. The aim is to overcome limitations of these modalities when used separately.

We call synergic systems, systems which allow both different modalities to be used in parallel and modalities to be combined to obtain a command. VoicePaint and Notebook are two multimodal synergic systems developed by our team. We focus on their software architectures in this article.

Introduction

Our goal is to identify software architecture components through the presentation of available multimodal systems. The design of software architectures for multimodal systems is still an area of exploration: parallel to the development of graphical user interfaces, natural language processing, speech recognition, gesture analysis and computer vision have made significant progress. Systems integrating these techniques and based on the usage of multiple modalities open a complete new world of experience.

We first propose a taxonomy of multimodal systems and then introduce two multimodal applications developed in our team, VoicePaint and NoteBook. Through the presentation of the applied software architecture model we identify the components involved in the two applications. We highlight the difference between implementation modules and model components by detailing each component of the software design of VoicePaint and NoteBook. We then focus on architecture aspects specific to multimodal systems such as abstract representation mechanism, media-dependent versus media-independent data, fusion of data, and levels of fusion.

1. Taxonomy

A *modality* is associated with a physical hardware device. Keyboards, mice, microphones, graphics displays, allow the user to communicate with a system using the corresponding modalities (typing, graphic input, speech, graphic output).
A *multimodal system* is a system that supports many modalities for input and output. For example, a system allowing interaction using a keyboard (typing modality) and a microphone (speech modality) is multimodal.
We propose a classification for multimodal systems, based on a continuous classification space defined by two axes:

- *Supported use of the modalities:* values along this axis indicate to what extent different modalities can be used at the same time. We distinguish two main levels: sequential (the modalities must be used one after another) and parallel (modalities may be used simultaneously).
- *Level of interpretation of input data or generation of output data:* values along this axis indicate how a single command or result is issued by combining modalities. Independent commands are created with only one modality. Combined commands are created with several modalities.

To classify a system according to these criteria, we identify a set of its features f_i (e.g. supported commands). Each feature f_i is weighted according to its estimated importance (w_i) and given an estimation of its location along the two previously defined axes, giving a point p_i.

$$f_i = (p_i, w_i)$$

The position γ of the considered system in the classification space is defined by the following equation.

$$\gamma = \frac{1}{\Sigma_w} \times \sum_i p_i \times w_i \qquad \Sigma_w = \sum_i w_i$$

Figure 1 shows the diagram and the approximate classification of VoicePaint and Notebook. Systems above the curved line are considered multimodal.

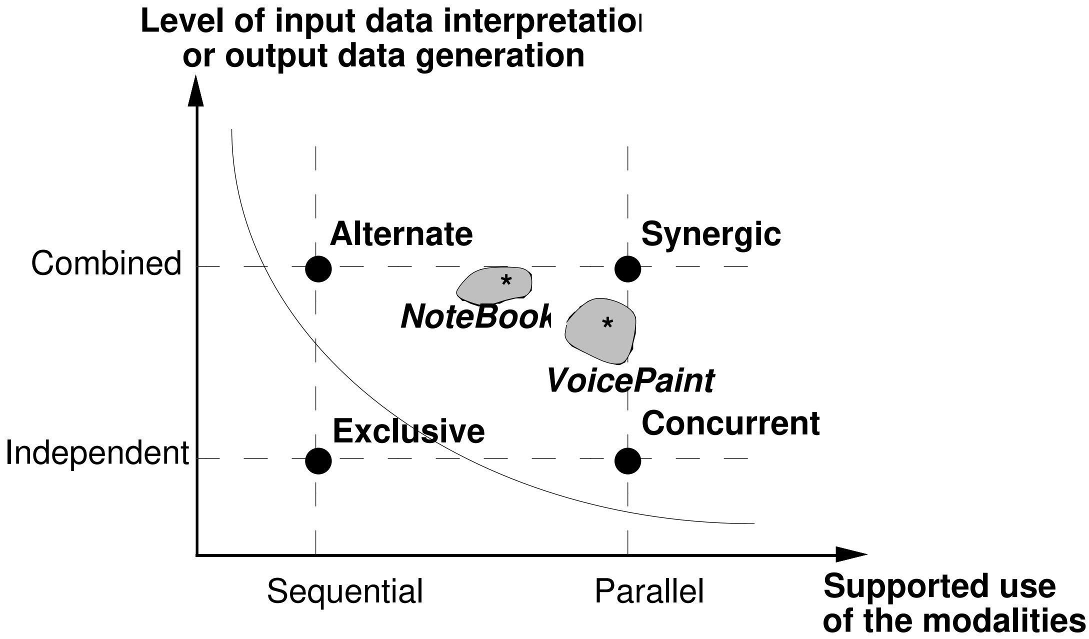

Figure 1. A Taxonomy of Multimodal Interactions
and Classification of NoteBook and VoicePaint

We consider four particular points in the classification space, corresponding to particular values of the axes: exclusive, alternate, concurrent, synergic.

- *Exclusive:* data expressed through different modalities are interpreted independently. For example, a graphical and vocal interface where the user has to switch explicitly between modes offers two different modalities but only one can be used to express a single command.
- *Alternate:* data expressed through different modalities can eventually be combined. For example, the Apple® Macintosh® Finder™ allows the user to select a document with the mouse and then open it with the keyboard (by typing the keyboard shortcut Command–O). The two modalities (pointing and typing) are used sequentially and combined into a single action.
- *Concurrent:* if input data are interpreted independently. For example, the VoiceFinder is a system that adds voice input to the Macintosh Finder [Articulate 90]; with this system, the

user can issue the voice command “Empty the Trash” while simultaneously opening a document by double-clicking it. The two modalities (speech and mouse) are used in parallel from the user perspective, but they express two independent commands.

- *Synergic:* if input data are combined. For example, in the VoicePaint or NoteBook applications, the user enters a single command using two modalities (speech and mouse) in parallel; the input data are combined to be interpreted by the system.

We classify VoicePaint and Notebook as synergic multimodal applications, because of their closeness to the synergic point. Systems allowing modalities to be combined and particularly synergic systems seem to be the most promising, but our knowledge for designing and building such systems is very primitive. We deal here only with systems called synergic in our taxonomy.

2. Presentation of VoicePaint and NoteBook

2.1. VoicePaint, a voice-enabled drawing application

VoicePaint is a colour, pixel-oriented drawing program (*à la* MacPaint). It handles speech, keyboard and mouse inputs. The application is developed on the Apple Macintosh platform using a custom application framework, Human Interface Kernel (HIK) [Gourdol 89].
The voice services interface and voice acquisition hardware is VoiceNavigator by Articulate Systems [Articulate 90]. VoiceNavigator provides discrete, speaker dependent voice recognition.
VoicePaint is a traditional Macintosh application with all the standard interface elements, as defined in the Apple User Interface Guidelines [Apple 86]. Because of this, mouse and keyboard can be used to manipulate windows, menus, dialogues. But because it is also a voice-enabled program, commands may also be issued by voice. For example it would be possible to create a new document or print the current one using voice commands.
VoicePaint also allows to use voice commands to modify brush attributes while drawing; this feature relies on synergic data fusion and would not be possible without multimodal input.
VoicePaint is more completely described in [Gourdol 91].

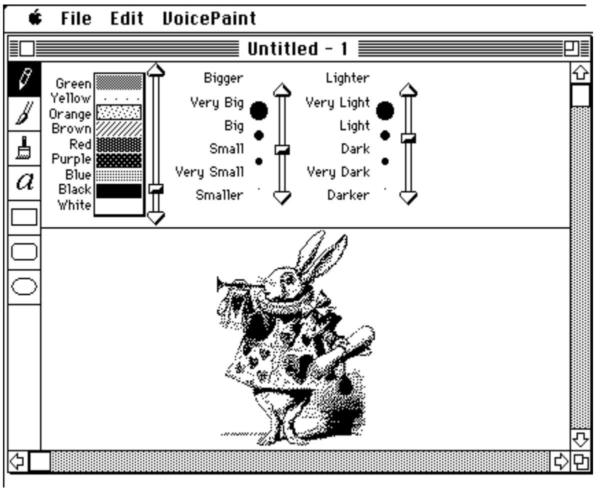

Figure 2. A snapshot from VoicePaint

2.2. Notebook, a voice-enabled electronic notebook

The Notebook application is a voice-enabled electronic notebook that handles both speech, mouse and keyboard inputs. The user can write a note, turn or remove pages from the notebook.

The voice interface services are provided by the Carnegie Mellon Spoken Language Shell (CM_SLS) [Lunatti 91].

Speech inputs apply to commands only. The content of the notes have to be typed in, they cannot be dictated. This limitation reduces the size of the application vocabulary, i.e. the set of words that the application must “know”. For example it is possible to create a new note, or search a given note using voice commands.

Synergic data fusion is used to perform the insertion of a new note by specifying the insertion position using the mouse and issuing the voice command "Insert a note".

The application is developed on the NeXT workstation with Interface Builder [Webster 89].

A complete definition of the Notebook external specifications is provided in [Nigay 91a].

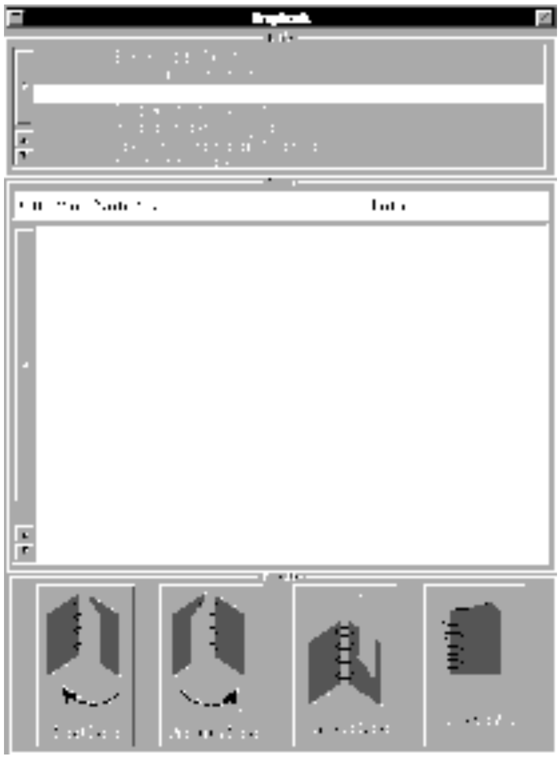

Figure 3. A snapshot from the Notebook application

To develop these two applications, we have applied the same software architecture model. We will now present this model and show how we used it to build our two applications.

3. Software architecture aspects for multimodal systems

First of all, we detail each component of our software architecture model and their relationships. This model, which integrates multiple modalities, derives from models intended for graphical user interfaces only, such as the Arch model [UIMS 91] and the PAC-Amodeus model [Nigay 91].
In a second part, we show how the model can be applied to design real applications, using VoicePaint and NoteBook as examples.

3.1. The applied software architecture model

This model aims at meeting the flexibility and adaptability software quality criteria of McCall [McCall 77], thus allowing efficient management of modifications in the application. Some of the components of the model are common with those of the Arch model and the PAC-Amodeus model. Figure 3 shows the components of the model and their relationships. Each of these components is described below.

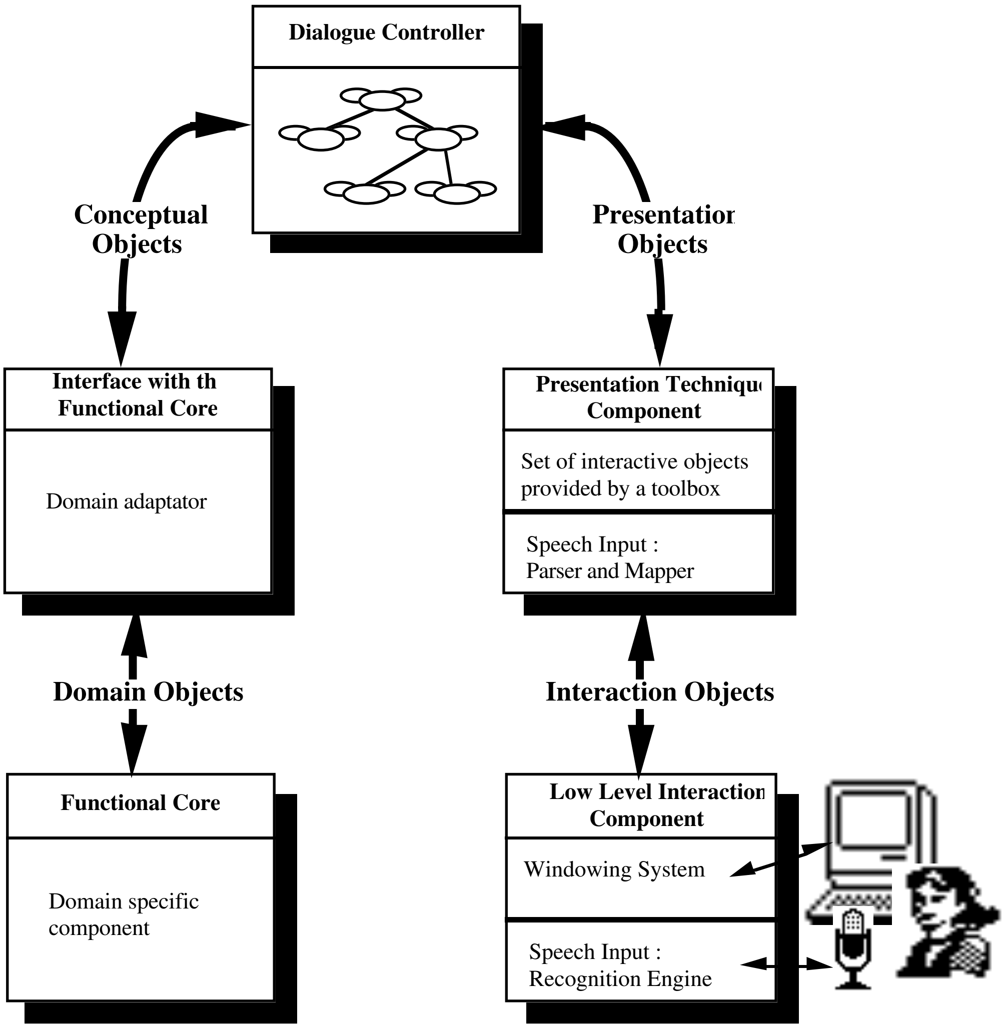

Figure 4. The software components of the applied model and the interfaces between them.

The Functional Core (FC) implements domain specific concepts in a presentation independent way.

The Interface with the Functional Core (IFC) allows communication between its two surrounding components, implementing a communication protocol. This protocol is fully described in [Coutaz 91].

The Dialogue Controller (DC) is the keystone of our model. It has the responsibility for task-level sequencing. Each task of the end-user corresponds to a thread of dialogue. This observation suggests a multiagent decomposition: a collection of agents can be associated with each thread of dialogue.
The DC receives events both from the FC via the IFC and from the user via the Low-Level Interface Component (LLIC) and the Presentation Techniques Component (PTC): the level of abstraction of events received from the user is the command

level (events may be the identifier or the parameters of a command). The lexical and syntactic analysis of user events have been performed in the PTC and LLIC components. At this level, the modality of the events is lost.

The Presentation Techniques Component (PTC) bridges the gap between the Dialogue Controller (DC) and the Low Level Interaction techniques.
The PTC describes the presentation (i.e., Image of the system as defined by D. Norman [Norman 86]). It implements the perceivable behaviour of the application for outputs as well as input commands. The interaction media is taken into account only at this level of abstraction.
Regarding outputs, the Presentation Objects, which convey data to be presented to the user, are translated in terms of one or several Interaction Objects. Interaction Objects are built from services provided by toolboxes such as Motif™ or Macintosh System 7 Toolbox
As an example of translation performed inside the PTC, let us consider an example confirmation message from NoteBook, asking the user if he really wants to delete a note. The confirmation message is a Presentation Object and can be presented to the user with a graphical representation, i.e. a window displaying a message, compound of basic Interaction Objects such as a Window, an OK Button and a Text Label. The Presentation Object may also be mapped onto a voice message such as: "Do you really want to delete this note?". There are at least three ways to define the mapping between the Presentation Object and the Interaction Objects:

- the mapping might be hard-coded, the same Interaction Objects are always used for the same Presentation Object;
- the user could also choose the Interaction Objects, for example the user can decide that all error messages should be spoken, i.e. presented using voice Interaction Objects;
- the mapping can also be defined by a set of rules, for instance based on an interaction history or a user model.

As for user's inputs, the PTC is in charge of the syntactic analysis of the user events. It dispatches the events towards the DC. This component depends on the syntax of the interaction language used by the application. It does not, however, depend on the semantic level of the application.
It is the right place to abstract or standardize different types (different interaction modalities) of events to obtain a command (identifier and/or parameters of a command).

The Low Level Interaction Component (LLIC) denotes the underlying platform, both software and hardware. It supports the physical interaction with the user. It manages (time-stamps and queues) end-user's events from different media and has the responsibility for their lexical analysis.
Some of the low-level events are not transmitted to the PTC. Indeed, lexical tasks such as resizing a window, are locally performed by the LLIC. In addition, in the case of spoken utterances, this component could include mechanisms for confirmation allowing the user to intercept a recognition.
The LLIC, or part of it, does not depend on the application. It includes basic services such as the windowing system, the spoken language shell etc. From the developer's perspective, it is considered as a constraint.

To sum up, let’s see how the system reacts to a user action. Input events from the user are fed into the Low-Level Interaction Component. They are then transformed into Interaction Objects, and handled to the Presentation Techniques Component, which may transform them into Presentation Objects. Those objects are then processed by the Dialogue Controller. The Controller eventually creates Conceptual Objects and passes them to the Interface with the Functional Core. This “domain adaptor” then transforms them into Domain Objects which are then processed by the Functional Core. Other Domain Objects are created, and follow the reverse path, up-to, if needed, the Low-Level Interaction Component. At each stage, objects are processed locally and discarded if possible. That is, the Functional Core will probably never deal with each and every key presses, but rather with string of characters, or still more abstract objects.

We need now to show how we applied this software architecture model to the software design of the VoicePaint and NoteBook applications.

3.2. From Model to Reality !

This section describes the software architecture design of VoicePaint and NoteBook. The components implementing these two applications are those exposed in the previous section. However, we show that some discrepancies between the design and the model may appear, mainly due to the software development tools.

3.2.1 The software design of VoicePaint

The VoicePaint architecture consists in the software components exposed in the software architecture model:

The functional core (CorePaint) contains primitives applying dyes onto a canvas, taking into account parameters of the brush as well as the dynamic of the application of the brush (i.e. more paint can be applied at the beginning of the stroke than at the end).

Because of the conceptual simplicity of VoicePaint, the Domain Adaptor does nothing in our case.

The Dialogue Controller (DC) is made up of a hierarchy of PAC agents and is supplemented by the Dialogue Manager of Human Interface Kernel (HIK). This Dialogue Manager provides high level standard services to the application program. These services are available without any agent hierarchy. This way, the PAC agents hierarchy of the Dialogue Controller is made simpler.

The Presentation Techniques Component (PTC) dispatchs commands to the appropriate agent in the PAC hierarchy of the DC. Voice commands are received by the PTC as strings from the Low-Level Interaction Techniques Component, then the PTC abstracts the received string (Interaction Object) into a command (Presentation Object).

The Low-Level Interaction Techniques Component (LLIC) includes the voice recognition services provided by the Voice Navigator recognition system, the Macintosh Toolbox and HIK for the graphical input and output.

3.2.2 The software design of the Notebook

Each component of the model exposed before, appears in the design of the Notebook. There is a slight difference between the implementation modules of the Notebook and model components, due to the software development tools. Indeed, the Notebook has been implemented in Objective-C on the NeXT workstation using NeXT Interface Builder [Webster 89] to define the graphical behaviour and CM-CLS to define the speech behaviour.

The FC manipulates a set of text files. Its main functions cover the standard file manipulation functions. No specific code has been written for it.
The IFC maintains a mapping between concepts of the FC and concepts belonging to the DC, for example, the link between the

number of a note (a DC concept) and the corresponding file name in the FC.
The DC is organized as a hierarchy of agents. For example there is one agent dedicated to the management of the Create Note and the Delete Note commands. It receives media independent messages. The agent is linked with the task and does not care how the user provided the message (i.e., which interaction modalities have been chosen to express a particular command).

The PTC is split into two main parts: the graphical definition (input and output) and the speech definition (input only).
The graphic definition is defined using Interface Builder. Interface Builder is a tool that allows the developer to define the graphical behaviour by direct manipulation. The link between the DC and the PTC is established using outlets (symbolic targets) mechanism (DC to PTC) and the action (call-back procedure) mechanism (PTC to DC).
In CM-SLS, the speech behaviour is defined with a grammar. This grammar is application-dependent and is used to parse the recognized utterances. To bridge the gap between the PTC and the DC, we defined a mapper between interaction objects (i.e. recognized utterances parsed through the parser) and Presentation objects (the Presentation part of an agent).

The LLIC is also split into two parts:

- Graphic output and mouse-key events are embedded inside Interface Builder.
- Interaction objects passed from the LLIC to PTC are recognized utterances (i.e. spoken utterances recognized by the CM-SLS), i.e. a string built up from a set of recognized words.

We showed that our architecture model is well-suited for the design of real applications. We would now like to point some aspects that are specific to multimodal interaction.

4. Our software architecture model and multimodal interaction handling

As shown through the presentation of our two applications, the implementation of synergic multi-modal user interfaces requires the fusion of several different objects of various modelling techniques. Each of these modelling techniques is dedicated to a specific modality. The fusion of these objects of different modelling techniques leads us to answer these two questions:

- How to perform the fusion? Does a uniform framework or common representation help?
- At which level of the software architecture do we have to perform the fusion?

We expose our views of these two problems in the following paragraphs and explain how we resolved it in the design of our two applications.

4.1. How to perform the fusion?

The integration of various modelling techniques leads to define a common representation at a particular level of abstraction. Moreover the use of a common representation for fusion is two-fold:

- The common representation might be used to represent data to be fusionned.
- The common representation might be used to represent the results of fusion.

In our applications, we use a common representation for multimodal fusion. Because the fusion is performed at the Dialogue Controller level, we use an abstract command representation. Whenever possible, media-independent commands should be used. They provide a greater flexibility to the user by allowing him to choose the media used to specify its command without making the dialogue controller more complex. Fusions are also easier to do this way.

The interaction modality discrimination appears in the Presentation Techniques Component. At this level of abstraction, interest in events is expressed using a formalism specific to each interaction media. For example, in the Notebook application, the location of mouse events determines which agent of the DC is concerned, a way to translate agent's interest. Agent’s interest to some speech events is specified using a grammar. The Presentation Techniques Component is in charge of abstracting the media-dependent data into a complete or incomplete command to be sent to the Dialogue Controller.

This mechanism implies an implicit multimodal fusion inside the Dialogue Controller which is not dependent of media. Although we aim at a media-independent Dialoguer Controller, a mention of which media are used to specify the abstract command should be available. Sometimes, the PTC would not be able to abstract: in this case it is important to specify the media. For example, to send a voice message in a voice mail system, we are interested in the digitized sound wave, and not in the abstract command it may (or more likely may not) represent. Moreover, in the general case, even if the PTC is able to abstract data, the Dialogue Controller should be able to backtrack and reverse to the media-dependent data. This is the reason why the mention of the media is necessary.

We now study into details how fusion is performed in our applications. As mentioned above, we do not do complex syntax analysis, but let our agent architecture do the work for us.
For this purpose, we use:

- in VoicePaint, a lightweight multitask kernel built into HIK to allow the input of several media at the same time,
- in Notebook, two different processes, one to handle speech input and one to manage the graphical behaviour.

As a result, for example, while a mouse command is processed, voice messages can be acquired, recognized, and sent to the appropriate agents. We chose to have the fusion of events done by a dedicated supervisor agent. No explicit fusion has to be specified either in the mouse or voice handler. Instead, we just allow the occurrence of several different media at the same time. Each communication mode can be represented with as many event messages as necessary; these messages are sent to the agents and their processing can be done in parallel or in synergy.
Although our approach does not allow complex, media dependent analysis (of the "Put That There" kind), it is an easy way to allow multimodal input. It is extensible, and it significantly enhances otherwise monomodal applications.

On the other hand, to allow complex media dependent analysis, the common representation could be used as a frame to perform the fusion such as in ICPDraw [Wretö 89]. In this case the different modelling techniques can be hooked onto this unique abstract representation.

In the short term, we think that a medium range approach, using a common representation both as a framework for fusion and as the result of fusion is a reasonable approach. Indeed, this compromise allows:

- an easy backtracking inside the mechanisms of abstractions,
- fusions at different levels of abstraction.

Although we only perform implicit fusion inside the Dialogue Controller level in our two applications, we have identified three levels of fusion taking place at different places inside our software architecture. We present these levels of fusion in the following paragraph.

4.2 Levels of fusion

As we explained above, synergic multimodal interaction implies fusion of data provided by the user using different media. Inside our software architecture model, we identify three levels of fusion:

- Semantic fusion: The DC is a good place for combining the results of two commands. To cement these results we use a dedicated agent dispatching the results to other agents. This is the case of Voice Paint.
- Syntactic fusion: The hierarchy of DC agents is also a good place to implement synergic multimodal interactions where multimodal events can be combined into a higher level to complete a command. The specification of a command often involves user actions distributed over multiple agents. These user's actions could be performed using different modalities. To cement the distributed multimodal actions into a higher abstraction (command level), we use a dedicated agent, parent of the agents which receives the elementary actions. This is the case of the Notebook.
- Lexical fusion: The lexical fusion is performed by the LLIC. A typical example of lexical fusion is found on the Macintosh: the Shift key combined with mouse click allow multiple selections.

5. Conclusion

We propose a software architecture model that provides a structure for the design of multimodal applications; through two examples, we detailed how this model suits the particular needs of the conception of real multimodal applications. However, our aim is not to provide a rigid organization, but we showed that functionalities can shift from component to component. We also identified different levels of abstraction, but the level of abstraction of data manipulated by each component may be adapted to the nature of data. Thus, our model can be tailored to take into account the goals of the developers, their weighting of quality criteria, and the constraints of the development tools.

Although our model, an extension from the Seeheim model, provides a way to design software architectures for multimodal application, we feel the urge to provide a media-independent way to specify commands. We are going to study a reliable mechanism, independent of the media used, allowing incomplete commands and continuous data capture.

6. References

[Apple 86] Apple Computer, Inc. Human Interface Guidelines: The Apple Desktop Interface, Adisson Wesley, New-York, 1986.

[Articulate 90] Articulate Systems Inc., The Voice Navigator Developer Toolkit, 1990.

[Coutaz 91] J. Coutaz, S. Balbo: Applications: a dimension space for User Interface Management Systems. In proceedings of CHI'91 Conference, New Orleans, April 27-May 2, 1991, pp. 27,32.

[Gourdol 89] Human Interface Kernel: Developer's Guide, 1989.

[Gourdol 91] Architecture des Interfaces Homme-Machine Multimodales, Master's Thesis, Université Joseph Fourier, 1991.

[Lunatti 91] J-M Lunatti, A. I. Rudnicky, Spoken Language interfaces: The OM system. In proceedings of CHI'91 Conference, New Orleans, April 27-May 2, 1991, pp. 453,454.

[McCall 77] J. McCall, Factors in software quality. (Ed.) General Electric, 1977.

[UIMS 91] The UIMS Workshop Tool Developers A Metamodel for the Runtime Architecture of an Interactive System, 1991.

[Nigay 91a] L. Nigay, A case Study of Software Architecture for Multimodal Interactive System: a voice-enabled graphic notebook, Technical Report, October 1991.

[Nigay 91b] L. Nigay, J. Coutaz, Building User Interfaces: A Cookbook for Organizing Software Agents. ESPRIT Basic Research Action 3066 AMODEUS (Assimilating Models of DEsigners, Users and Systems), 1991.

[Norman 86] D. A. Norman, S. W. Draper, User Centred System Design. Lawrence Erlbaum Associates Publ., 1986.

[Webster 89] B. F. Webster, The NeXT Book, Addison Wesley, New York, 1989.

[Wilson 91] M.D. Wilson, D. Sedlock, J.-L. Binot, P. Falzon: “An Architecture for Multimodal Dialogue”, in M.M. Taylor, F. Neel & D.G. Bouwhuis (Eds.), Proceedings of the second Venona Workshop on Multi-Modal Dialogue, 1991.

[Wretö 89] J. Wretö, J. Caelen,ICPDraw, Rapport final du projet ESPRIT MULTIWORKS n° 2105, 1989.