The AMODEUS Project
ESPRIT Basic Research Action 7040

PAC-Amodeus Overview and worked examples

Joëlle Coutaz, Laurence Nigay and Daniel Salber

14th December 1993

SM/IR1

System Modelling: PAC-Amodeus approach to modelling user interface architectures

What does the PAC-Amodeus approach offer?

Given the external specification of an interactive system (i.e., the user interface as perceived by the user), our approach provides the software designer with rules to devise the general software architecture of the system.

PAC-Amodeus, based on early experience on PAC, is a blend of the software components advocated by the Arch model and the refining process in terms of agents advocated by multi-agent models such as PAC. Arch is a conceptual model which makes explicit the hooks with reusable software such as user interface toolkits. PAC, like MVC, generalizes the distinction between concepts and presentation techniques by applying the separation of concerns at multiple levels of abstraction, i.e., among cooperating agents. They differ however in the way they perform the distribution and the cooperation.

To whom?

Software designers.

Thumbnail sketch of the PAC-Amodeus approach:

From the system perspective, the approach works in a bottom up way: given the perceivable behaviour of the system, the approach helps the software designer to identify the set of internal components (e.g., agents or interactors) that will correspond to the specification of the user interface. For example, if the user interface definition specifies that two tasks can be executed in parallel or in an interleaved way, then two clusters of agents should be introduced in the architecture to support the specified multithreaded interaction.

How, when or where might it fit into existing practice?

If you consider the software development process models used in software engineering such as the waterfall model or the V cycle, then our approach fits exactly between the user interface specification step and the implementation phase. Before coding, implementers must specify the components of the code and their relationships. This specification defines the architecture of the code and PAC-Amodeus is useful to devise this architecture.

How might the PAC-Amodeus approach be used?

Modelling by software designers. Devising the architecture from the specification of the user interface can be done by applying the set of heuristic rules that have been developed for PAC-Amodeus (and which have been implemented within PAC-Expert).

What information does its use require?

PAC-Amodeus requires the detailed specification of the user interface. Unfortunately, in current practices, this specification is usually expressed in natural language augmented by screen dumps. As a result, the description is often incomplete, ambiguous, or underspecified, leading to misinterpretations and wrong implementations. We are currently working on the definition of a task-oriented specification language based on UAN, that could be used by non computer scientists to inform software designers in a precise and complete way of the tasks which the user interface should support.

What skills are required to use it?

Software engineering skills are required.

PAC-Amodeus Worked Example n° 1: The Portholes system

1 Introduction

In general our preliminary approach addresses two issues:
• The graphical output user interface.
• The software architecture of the system.

2 Graphical user interface issues

Relating to the Issue 1, we address the problem of making connections with RAVE.

2.1 The matrix of icons

We propose different kinds of buttons according to the types:

- a user,
- a room,
- a device.

This may help to recognize the available connections. For example it is not possible to establish a V-phone connection with a vcr.

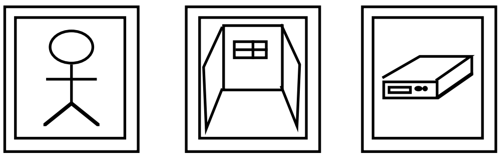


Figure 1: Different kinds of icon.

2.2. The current connections

It might be useful to know the current connections to decide whether to make AV connections. Present graphically all the current connections ("*Continuously accessible feedback mechanism*").

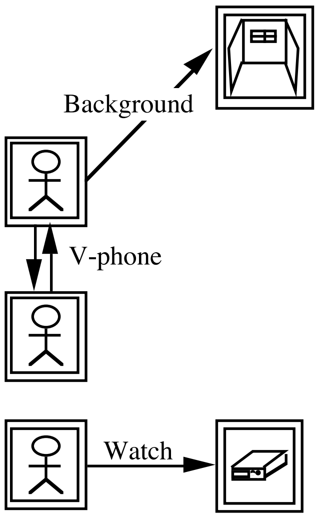
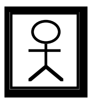


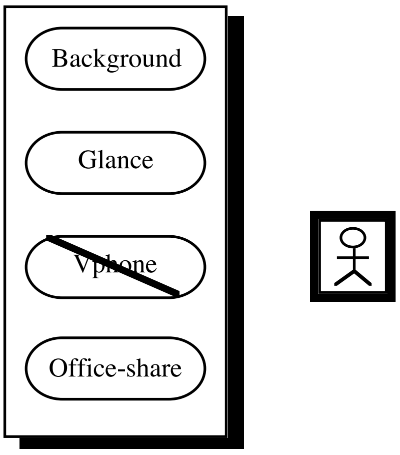
Figure 2: A graphical presentation of the connections.

2.3. Feedback mechanism

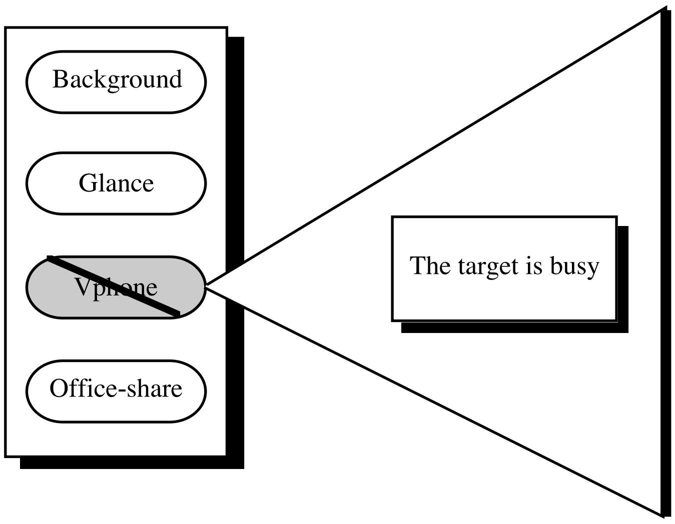
We propose a way to present "*Continuously accessible feedback mechanism*".
First the user selects the target:



According to the target, the icons of the non-accessible services are modified:



If the user wants to know why a particular service is not available, the user selects the corresponding icon.



3 PAC-Amodeus architecture: A first draft

As shown on figure 3, there is one agent per user. The user can then be connected from everywhere. This *User* agent will be the root of the hierarchy of agents corresponding to the graphical interface. This agent will be a process.

We then dynamically create an agent, which is also a unix-like process, per established connection (one agent per connection). All these *Connection* agents are able to communicate with the server Godard. They are special PAC agents without Presentation facet. The Abstraction facet maintains the state of the connection.

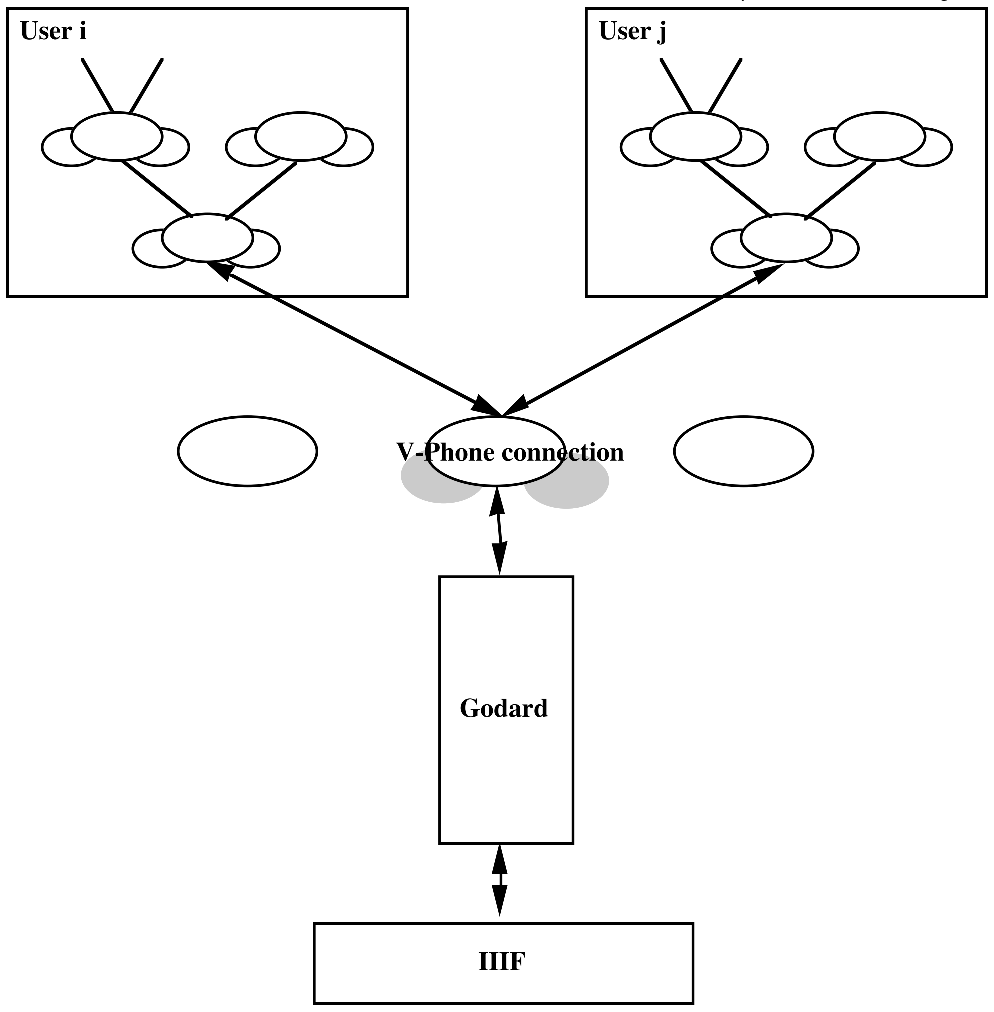


Figure 3: Global PAC_Amodeus architecture of the Portholes system.

We now consider the PAC-Amodeus architecture which manages the graphical output interface on each workstation. Our study is based on the Figure 3, titled "The Raven AV Connection Service Selection Interface" (RP3-IR1). The root of the hierarchy of PAC agents is a User Agent (see above).

One agent called Palette is in charge of the services buttons displayed on top of the window.

The two general services Answer and Quit are not managed by this Palette agent but by the root agent which has access to the network.

One Matrix agent corresponds to the matrix of name buttons. This agent receives the mouse input from the user on each button through its Presentation facet.

The last leave of the hierarchy is the System State agent. Its Presentation is a non editing window. Its abstraction is empty, and its main fonction is to display the state message received by the root agent. The role of this agent could be enhanced by adding a real error managing mechanism: its Abstraction would then maintain a symbolic representation of the errors. Its Control would map the symbolic representation to a perceivable rendering, such as a voice message, a graphical status man or a synergistic output presentation. An example of synergistic presentation could be: a voice message "A Glance connection is not allowed by this user", while blinking the name button of Lisette.

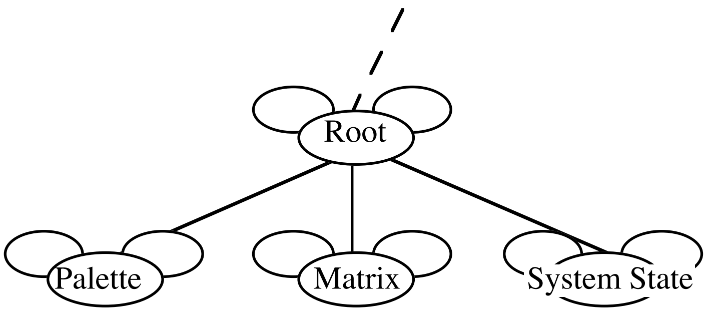


Figure 4: PAC_Amodeus architecture to manage graphical interface.

Figure 5 shows messages passing inside the hierarchy to provide a feedback when the user selects a name button

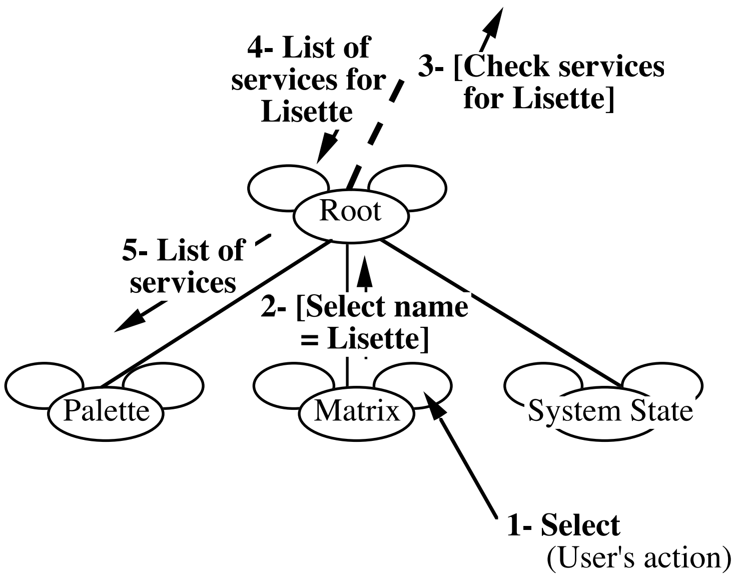

4- List of services for Lisette
3- [Check services for Lisette]
Root
5- List of services
2- [Select name = Lisette]
Palette
Matrix
System State
1- Select (User's action)

Figure 5: Messages passing for feedback.

PAC-Amodeus Worked Example n° 2: Dynamic Gesture System

Our second example is a RP1 common exemplar, fully described in [Bordegoni 93].

1 Introduction

The Dynamic Gesture System (DGS) is a software platform to develop systems that support gesture as input. Gesture is specified using a dataglove and a space-ball. DGS is composed of two parts: A Specification System used for defining poses and a Recognition System in charge of recognizing poses.
Based on DGS, two exemplars have been developed to show its potential applicability within 3D interfaces: a 3D Drawing Editor (3DDE) and a System for navigating through a 3D building [Bordegoni 93].
We provide an overview of the capabilities of 3DDE in terms of the M^2LD classification space [Nigay 93]. In section 3, we present the PAC-Amodeus architecture of 3DDE.

2 Classifying 3DDE

To present an overview of the system, we classify 3DDE according to three complementary perspectives: M^2LD, O^2LD and ULD. All of these classication schemes use a common material: the notions of interaction language and physical device. As specified in the glossary [Salber 93], an interaction language defines the set of well-formed expressions, i.e., a conventional assembly of symbols, that convey meaning. The generation of a symbol (or a set of symbols) results from a physical action, i.e., an action on a physical device, whose manifestation is an event. The message associated to this event is, precisely, a symbol (or a set of symbols).

M^2LD provides an overall rough static classification in terms of the number of interaction languages and physical devices that the system supports for input and output. O^2LD and ULD adopt a dynamic perspective. O^2LD provides a classification based on the options among interaction languages and devices that the system and the user have available at a given time. ULD analyzes how these options may be combined.

2.1 A static point of view: M^2LD

M^{2}LD stands for Mono/Multi Language and Device. It allows the characterization of a computer system in terms of the number of interaction languages and physical devices that the system supports for input and output.

• *MonoLanguage and MultiDevice for input.* (Figure 1) The input language defines the command structure for manipulating 3D objects:

[Command name, Parameter Values]

The input devices are the space-ball and the dataglove.

• *MultiLanguage and Monodevice for output.* (Figure 1)There is one output language to express the 3D graphical representation of the scene (composed of objects such as cubes, cylinders, etc.). Another output language defines the cursor shape which is modified according to the current active command. For example, the hand cursor (used as the default cursor shape) becomes a virtual tool such as a brush, when the current command is painting. The cursor language is a very simple (no composition between symbols), but is still a language since it defines a mapping between the notion of virtual tool and an analogic shape. 3DDE uses one output device only: the screen.

2.2 A dynamic point of view: O^{2}LD and ULD classification spaces

We now present the results of our analysis of the system from a dynamic perspective considering the system in use at a given time.

2.2.1 O^{2}LD classification space

O^{2}LD , which stands for Obligatory/Optional Language and Device, addresses the following two questions:

• The user has a well-formed intention: what are the choices with regard to input devices and input interaction languages to express the intention to the system?

• The system needs to render a concept or express a state change: does it perform any choice with regard to output languages and output devices?

Applying an analytical approach, we adopt a global view of the system to locate3DDE. 3DDE is:

• *Obligatory Language and Optional Device for input.* Since 3DDE is MonoLangage for input, it cannot offer the user with any choice: it is an Obligatory Language system for input. On the other hand, the user may have choice between multiple input devices. For example, a selected object may be moved, resized, rotated or zoomed using either the dataglove or the space-ball. In this context (see glossary), the dataglove and the space-ball are said to be *equivalent* (see glossary). In other contexts, the choice of the input device is constrained. For example, an object selection must be performed with the dataglove. In this case, we say that there is an *assignment* relation between the device and the language (see glossary). More generally, 3DDE has been designed so that the dataglove is used to specify the command name and the space-ball to specify parameter values. Thus, although 3DDE is Multidevice for input, it imposes assignment constraints on the usage of input devices depending on the current context. In Figure 1, point "3DDE-input" denotes the location of 3DDE within the O^{2}LD space for input.

• *Obligatory Language and Obligatory Device for output*. Rendering in 3DDE is based on two languages: one for the presentation of the 3D scene and one for expressing the current virtual tool . Although the two languages are used simultaneously (3DDE is MultiLanguage for output), the system does not perform any choice between these languages. There is an *assignment* relation between the output interaction language and the conceptual units of the system. Since the system is MonoDevice for output, the system has no choice about the physical support for rendering information. These observations justify the location, "3DDE-output" in figure 1, close to the point "Oblig. L Oblig. D".

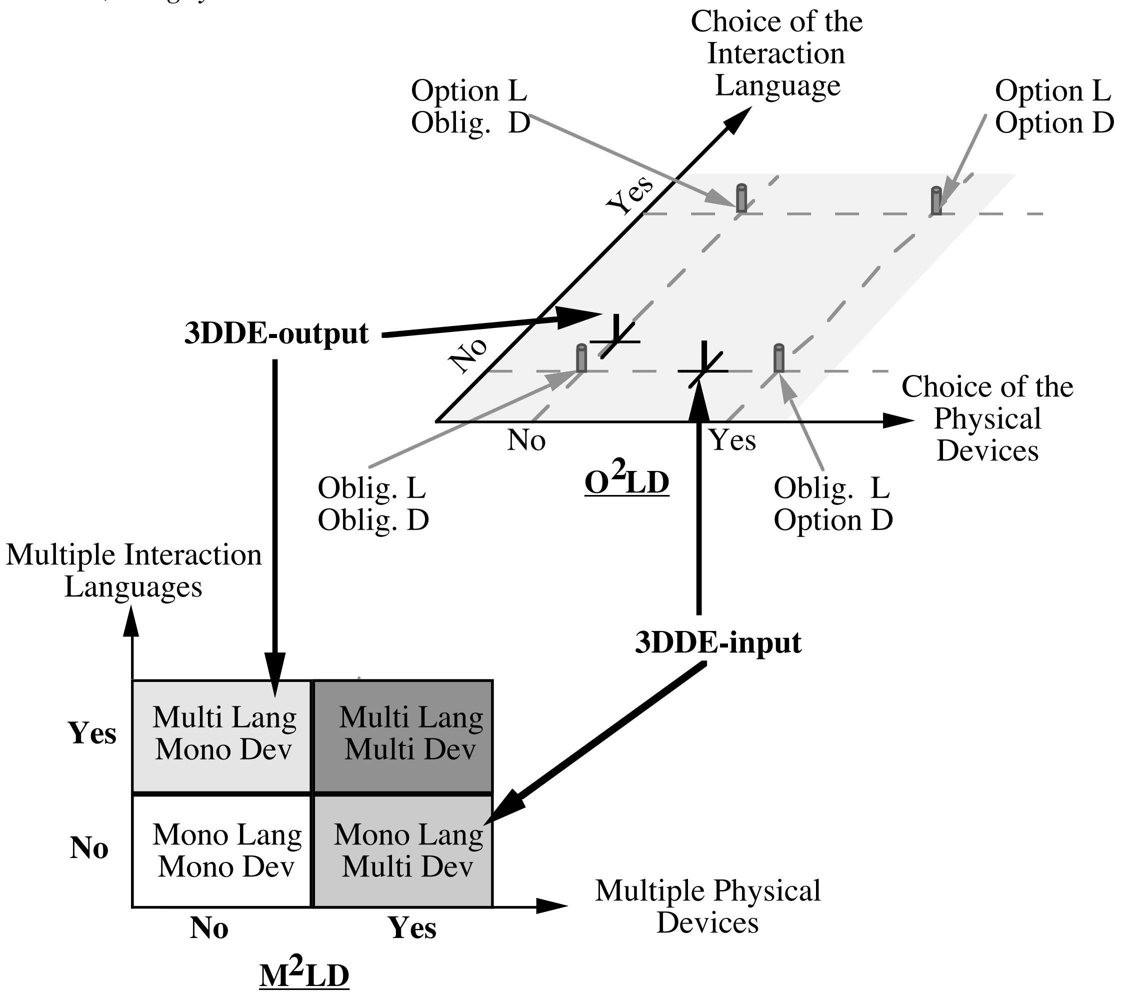


Figure 1: Location of the 3DDE in the O^2LD classification space. "3DDE-input" point is the location of the 3DDE input interface and "3DDE-output" point the 3DDE output interface.

2.2.2 ULD classification space

We now consider the usage dimension of languages and devices at some point in time by the system and by the user. Usage covers the absence or presence of combination of languages or devices over time. We consider four types of usage (see Figure 2):

• *exclusive usage* covers the sequential and independent use of languages (or devices); languages (or devices) are not used simultaneously (sequential use) and the expressions (or events) they convey are not combined (independent use).

• *concurrent usage* denotes the parallel but independent use of languages (or devices); languages (or devices) are used simultaneously (paralell use) but the expressions (or events) they convey are not combined (independent use).

• *alternate usage* means sequential and combined use; languages (or devices) are not used simultaneously (sequential use) but the expressions (or events) they convey are combined. Typically coreferences between expressions supported by different languages (e.g., 'put that there') requires combination.

• *synergistic usage* corresponds to the parallel and combined use; languages (or devices) are used simultaneously (parallel use) and the expressions (or events) they convey are combined.

Usage of languages (or devices) implies that at least two languages (or devices) are available at some point in time. We observe that 3DDE is MonoLanguage for input. Thus, the concept of "usage of input languages" is irrelevant for 3DDE. As far as input devices are concerned, the user can use them in a synergistic way, concurrently, or in an exclusive way.

• *synergistic usage of input devices*: The user can paint an object by pointing at the object with the dataglove while modifying the color with the space-ball (one button per color) (instant t1 on Figure 2). Also, an object can be created by selecting an object model while modifying the shape of the object using the space-ball (instant t2 on Figure 2). Here, two devices are used in parallel to specify the command create (objectId, size, location) but note that each device is assigned is a specific use (they are not equivalent).

• *concurrent usage of input devices*: This occurs when the user performs zoom operations on the scene with the dataglove while moving a selected object with the space-ball (instant t3 on figure 2). Two devices are used in parallel to specify two independent commands.

• *exclusive usage of input devices*: This situation is illustrated by the following sequence: the user selects an object using the dataglove. The space-ball can then be used to rotate the selected object (instant t4 on figure 2).

• *no alternate usage of input devices* is supported by 3DDE. This is true if the rotate, move, etc. commands are modelled as "rotate (angleValue)", "move (dxdyValue)" and are applied to the global variable "current selected object". If, on the other hand, these commands are defined as "rotate (objectId, angleValue)", that is, if they require an explicit object id as a parameter, then selection is not a full fledged command but an action whose effect must be combined with the specification of other parameters. In this case, we would assist to an *alternate usage of input devices* not an exclusive usage as presented above. Here, the distinction relies on implementation criteria. It may be the case however that an alternate usage from the system point of view is mentally modelled by the user as an exclusive usage. An interesting point to check for conformance mapping!

As shown in figure 2, these observations (instant t1, t2, t3 and t4) suggest to locate 3DDE input interface close to the synergistic point.

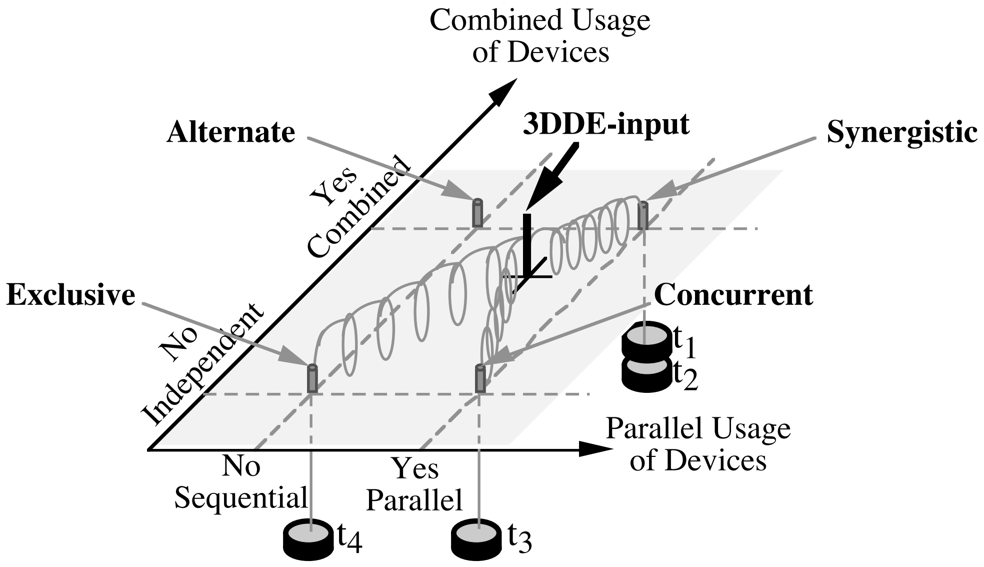


Figure 2: Location of the 3DDE input interface in the ULD classification space.

• For output, two languages are simultaneously used by the system to render two independent presentations: the 3D scene and the cursor shape. It then implies a *concurrent use of two output languages.*

2.3 Summary

Input interface of 3DDE:

MonoLanguage, MultiDevice (M^2LD)
Obligatory for Language, Optional for Device (O^2LD)
Synergistic, Concurrent and Exclusive Usages of Devices (ULD)

Output interface of 3DDE:

MultiLanguage, MonoDevice (M^2LD)
Obligatory for Language, Obligatory for Device (O^2LD)

3 PAC-Amodeus architecture

Figure 3 shows one possible PAC-Amodeus software architecture for 3DDE. We are aware that this proposal is rather sketchy but will be analyzed further later on.

The LLIC component receives hand events from the user and abstracts them in terms of poses. Similarly, space-ball events are abstracted in terms of location and button selections.

The Gesture recognition engine of the PTC component receives the poses and defines the dynamic gesture. To each dynamic gesture is associated a command. The vehicle of the command is a melting-pot object. Symmetrically, space-ball events are abstracted in terms of command, stored in a melting-pot object.

The DC is composed of a two-level hierarchy of PAC agents. The leaves of the hierarchy correspond to the 3D objects of the scene as well as to the space-ball which is special case of a graphics object.

The FCA maintains the mapping function between objects of the Functional Core and the objects of the DC. For example, the FC contains the description of a bedroom expressed in meters. The DC manipulates objects in centimeters. In this particular case, the AFC would perform scale mapping.

Figure 3 highlights message passing for the painting synergistic command using the dataglove while changing the color by clicking a button of the space-ball. Figure 4 makes explicit the actions of the fusion engine on the melting-pot objects within the dialogue controller.

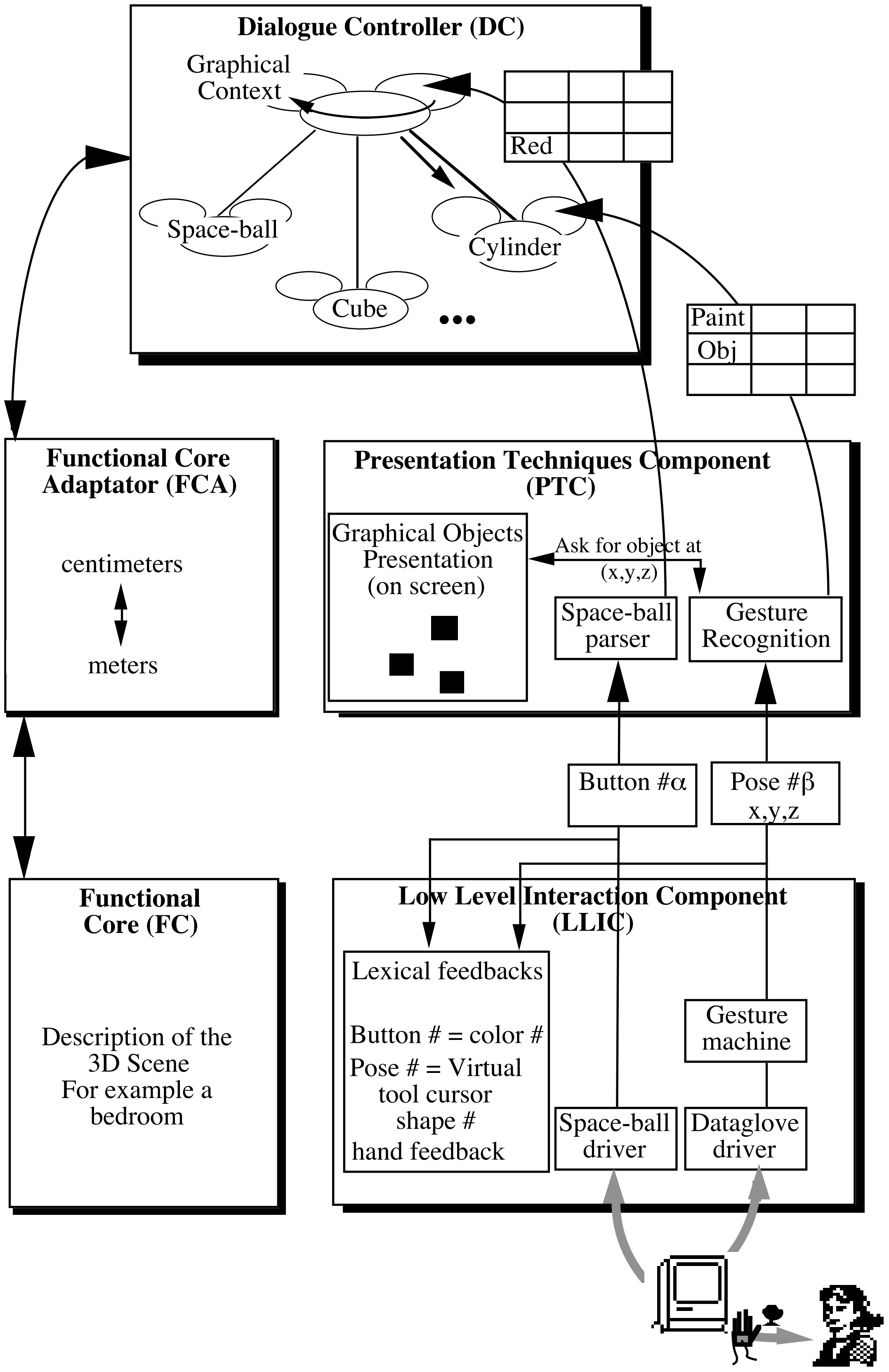


Figure 3: PAC-Amodeus architecture of the 3DDE.

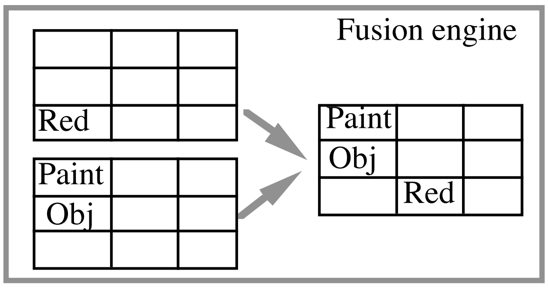


Figure 4: Dialogue Controller: Fusion engine.

References

[Bordegoni 93]

M. Bordegoni & M. Hemmje: An Interaction Model based on Hand Gestures for 3D User Interfaces, ERCIM Workshop on Multimodal Human-Computer Interaction, Working Material, Nancy, 2-4 nov. 1993.

[Nigay 93]

L. Nigay, J. Coutaz, D. Salber: GIO Interactors and PAC-Amodeus: Integration through the Dynamic Gesture System exemplars, Amodeus SM/WP27, 24 nov. 1993.

[Salber 93]

D. Salber, J. Coutaz, L. Nigay (editors): The System Modelling Glossary, Amodeus SM/WP26, 1st dec. 1993.