The AMODEUS Project

ESPRIT Basic Research Action 7040

GIO Interactors and PAC-Amodeus: Integration through the Dynamic Gesture System Exemplars

M.Bordegoni, J.Coutaz, G.P.Faconti, L.Nigay, D.Salber

23rd February 1994

Amodeus Project Document: System Modelling/WP27

AMODEUS Partners:
MRC Applied Psychology Unit, Cambridge, UK (APU)
Depts of Computer Science & Psychology, University of York, UK. (YORK)
Laboratoire de Genie Informatique, University of Grenoble, France. (LGI)
Department of Psychology, University of Copenhagen, Denmark. (CUP)
Dept. of Computer & Information Science Linköping University, S. (IDA)
Dept. of Mathematics, University of the Aegean Greece (UoA)
Centre for Cognitive Informatics, Roskilde, Denmark (CCI)
Rank Xerox EuroPARC, Cambridge, UK. (RXEP)
CNR CNUCE, Pisa Italy (CNR,CNUCE)

Abstract

In this working paper, we use a 3D Drawing Editor (3DDE) as an integrating material for the GIO (i.e., CNUCE) and PAC-Amodeus modelling approaches. 3DDE consists of a scene where the templates of some simple 3D objects are shown. By picking any one of the templates a new instance of a corresponding object is created and located in a space. Object instances can be subsequently manipulated by means of either the hand input device or the space-ball or the combination of both. We provide an overview of the capabilities of 3DDE in terms of the M2LD classification space. The concepts and principles of the M2LD framework is briefly introduced. We then present the PAC-Amodeus architecture of 3DDE followed by the GIO interactors description. The last section compares the two approaches and make explicit mapping links between GIO and PAC-Amodeus.

1 Introduction

The Dynamic Gesture System (DGS) is a software platform to develop systems that support gesture as input. Gesture is specified as a sequence of postures (hand poses) performed by using a dataglove. 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.

In this working paper, we use 3DDE as an integrating material for the GIO and PAC-Amodeus modelling approaches. 3DDE consists of a scene where the templates of some simple 3D objects are shown. By picking any one of the templates a new instance of a corresponding object is created and located in a space. Object instances can be subsequently manipulated by means of either the hand input device or the space-ball or the combination of both. In the following section, we provide an overview of the capabilities of 3DDE in terms of the M2LD classification space. In section 3, we present the PAC-Amodeus architecture of 3DDE followed by the GIO interactors description. The last section compares the two approaches and make explicit mapping links between GIO and PAC-Amodeus.

2 Classifying 3DDE

In the following, we will 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, 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: M2LD

M2LD 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. Let:

di, be the number of distinct input devices provided by the system,
do, be the number of output devices,
li, be the number of distinct input interaction languages supported by the system,
lo, be the number of distinct output interaction languages.

In M2LD, the system is said to be:

MonoDevice for input (or output) if *di* =1 (or if *do* =1),
MultiDevice for input (or output) if *di* > 1 (or if *do* >1),
MonoLanguage for input (or output) if *li* = 1 (or if *lo*=1),
MultiLanguage for input (or output) if *li* > 1 (or if *lo* >1).

As shown in figure 1, 3DDE is:

MonoLanguage and MultiDevice for input. 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. 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.

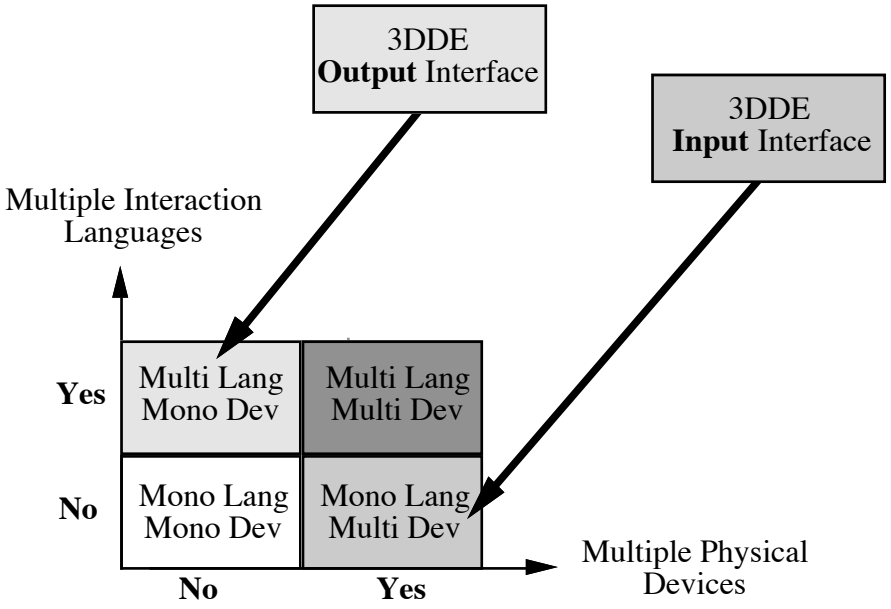


Figure 1: Location of the 3DDE in M^2LD.

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

We now analyse the system from a dynamic perspective considering the system in use at a given time.

2.2.1 O^2LD classification space

O^2LD , 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 2, point "3DDE-input" denotes the location of 3DDE within the O^2LD 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 2, close to the point "Oblig. L Oblig. D".

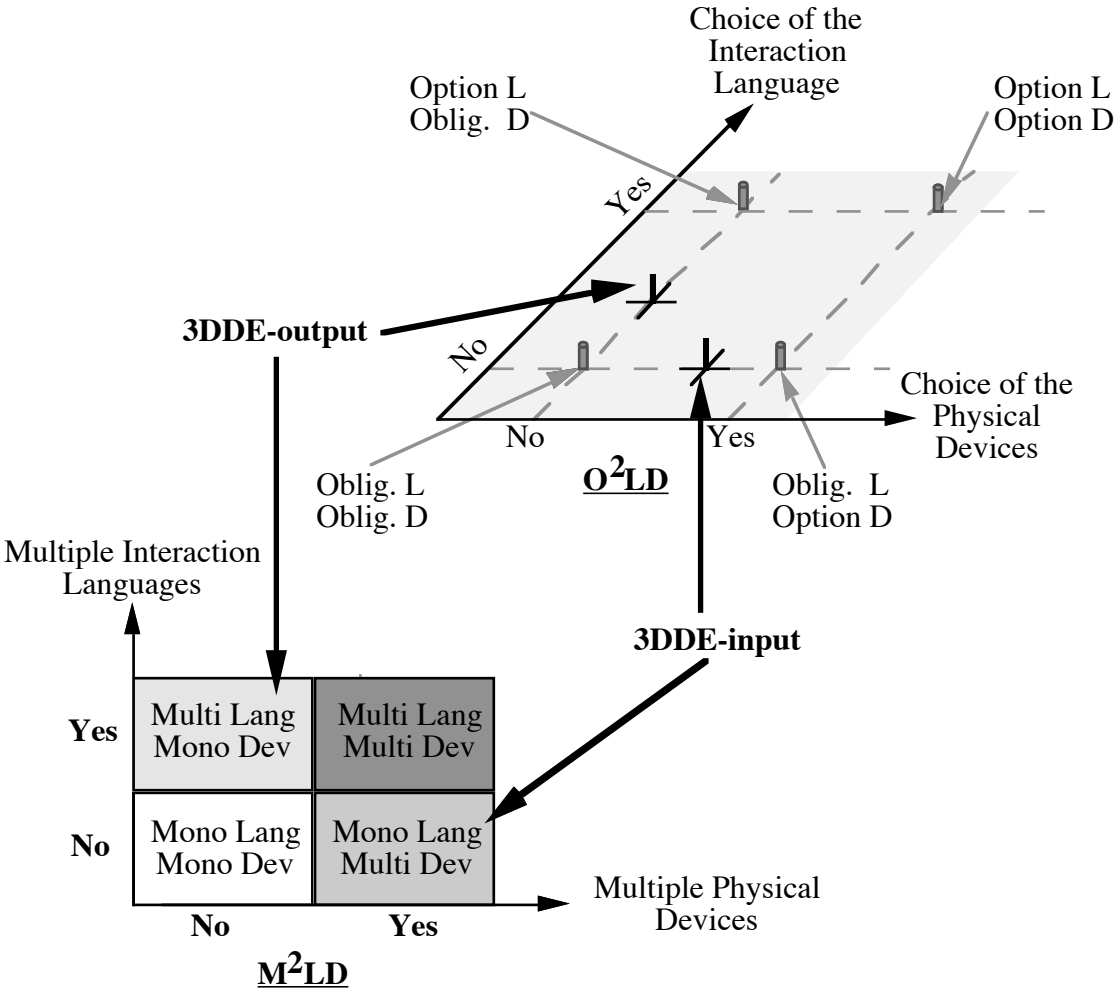


Figure 2: Location of the 3DDE in the O^2LD classification space. "3DDE-input" point is the location of the 3DDE input interface and "3DDE-output" point the 3DDE output interface.

XX LAurence, je te laisse modiifer le fichier"

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 3):

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 3). 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 3). 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 3). 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 3).

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

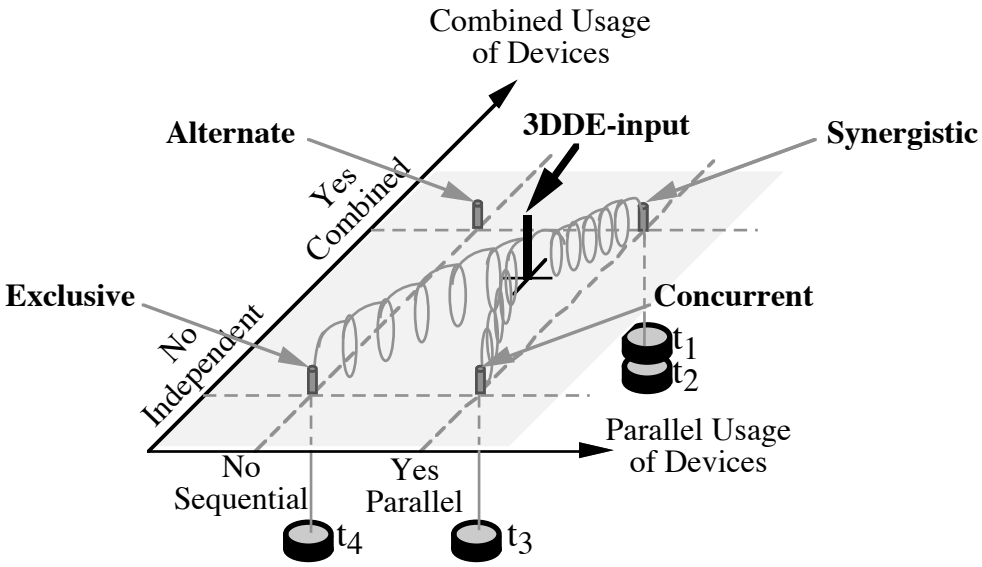


Figure 3: Location of the 3DDE input interface in the ULD classification space.

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)
Concurrent Usage of Languages (ULD)

3 PAC-Amodeus architecture

Figure 4 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 4 highlights message passing for the painting synergistic command using the dataglove while changing the color by clicking a button of the space-ball. Figure 5 makes explicit the actions of the fusion engine on the melting-pot objects within the dialogue controller.

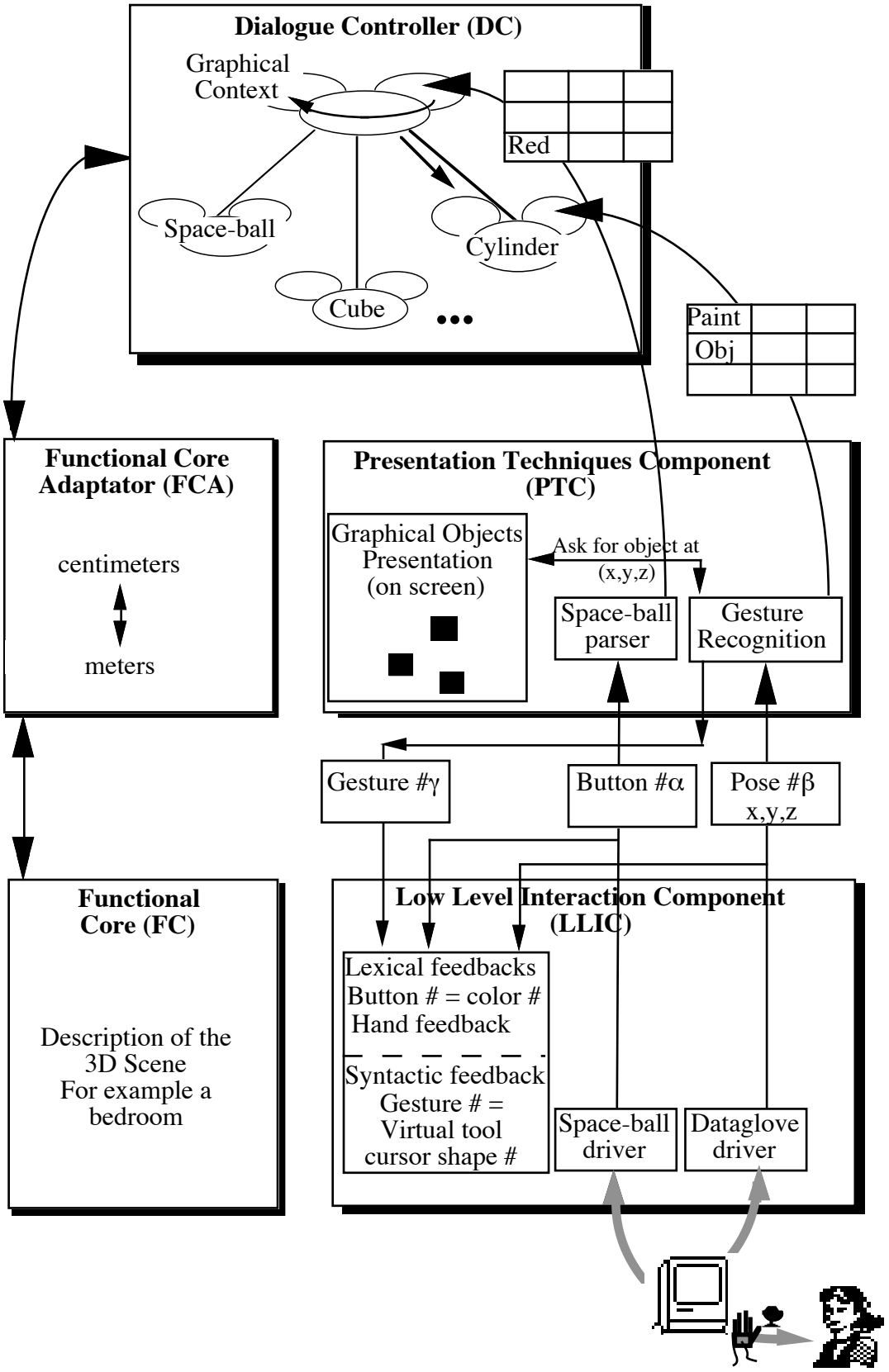


Figure 4: PAC-Amodeus architecture of the 3DDE.

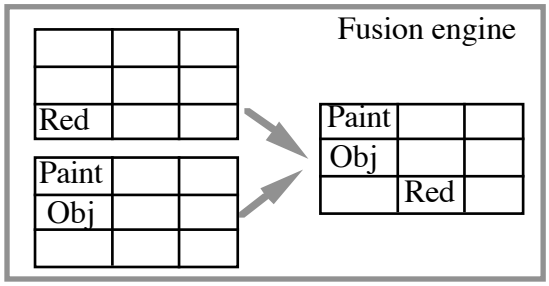


Figure 5: Dialogue Controller: Fusion engine.

4 GIO Interactors

In this section a description of the 3DDE system is presented in terms of GIO interactors. The specification is quite general and needs further refinement; however it gives an idea of the underlying methodology and also presents concepts that are used to compare the GIO and PAC approaches in the next section.

We start by considering the major components that make up the system:

Functional Core that exchanges with the user interface four different types of information through an equal numer of communication channels (or gates):

scene to direct the presentation of the objects in the scene,

g_out to direct the presentation of the dataglove feedback,

s_out to direct the presentation of the spaceball feedback,

g_in to receive input from the gesture system,

s_in to receive input from the spaceball,

gs_in to receive the combined input from the spaceball and the gesture system.

I_Scene the interactor responsible for presenting the scene through the *p_scene* gate,

I_Spaceball the interactor responsible for handling the input from the spaceball and its feedback. It receives input from the user through the *i_spaceball* gate and from the functional core through the *s_out* gate, presents output through the *p_spaceball* gate, and gives input either to the functional core or the *I_Fusion* interactor through the *s_in* gate,

I_Gesture the interactor responsible for recognizing the gesture and presenting the appropriate feedback. It receives input from the *I-Pose* through the *i_posture* gate and from the functional core through the *g_out* gate, presents output through the *p_glove* gate, and gives input either to the functional core or the *I_Fusion* interactor through the *g_in* gate,

I_Pose the interactor responsible for recognizing the postures. It receives input from the user through the *i_glove* gate and delivers postures to the *I_Gesture* interactor through the *i_posture* gate,

I_Fusion the interactor responsible for combining the input from the spaceball and the gesture system received respectively through the *s_in* and the *g_in* gates

I_Scene is a concretizing interactor missing of the abstraction components, *I_Pose* and *I_Fusion* are abstracting interactors missing the concretization components, and *I_Spaceball* and *I_Gesture* are full interactors including both abstraction and concretization components.

Figure 6 presents the composition of interactors modelling 3DDE. Such a composition highlights the flow of the data within the system.

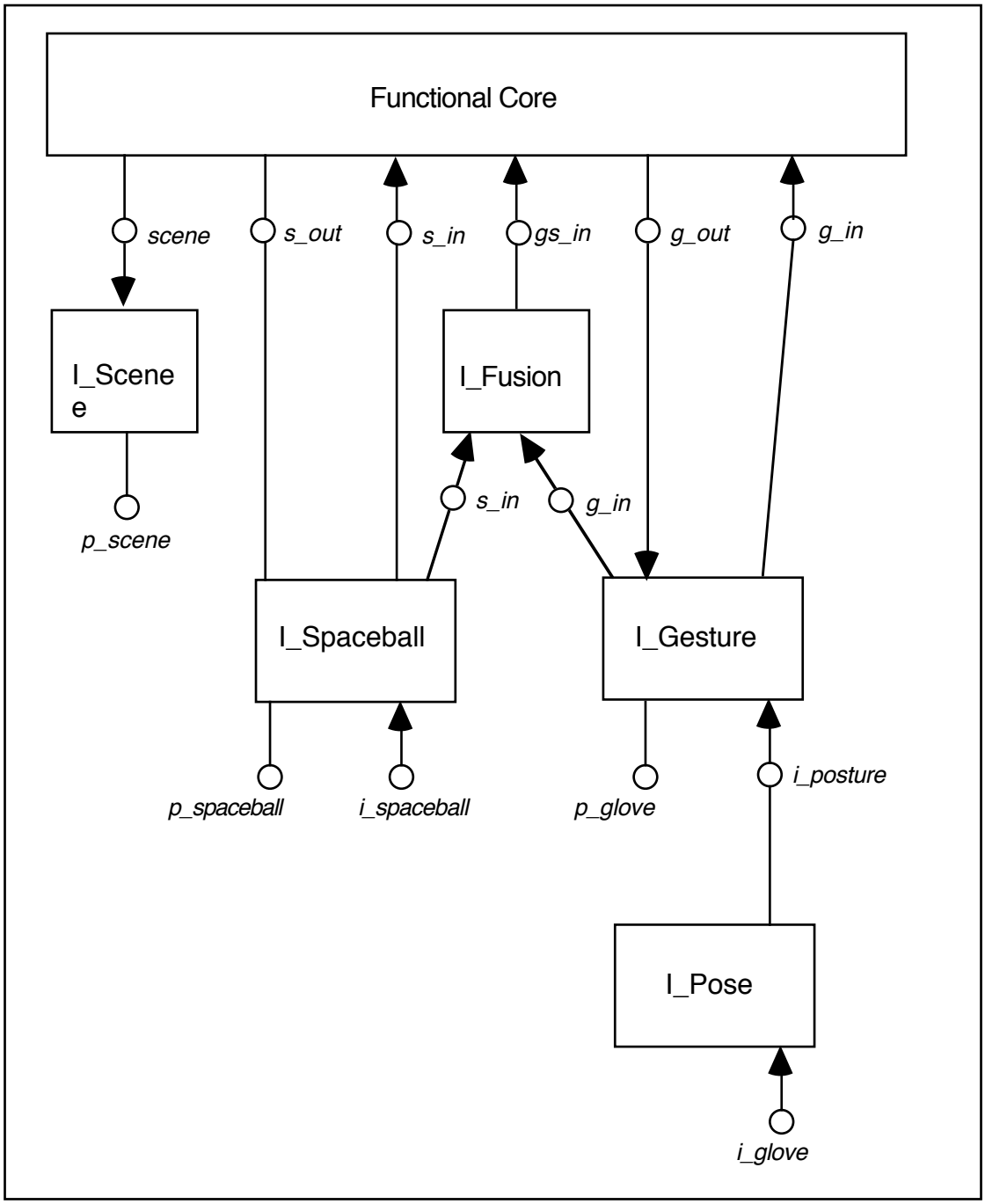


Figure 6: Interactors modelling the 3DDE.

The corresponding LOTOS behaviour expression derived from figure 6 is:

```
(
    (
    Functional Core [s_in, g_in, gs_in, scene, s_out, g_out]
    |[gs_in]|
    I_Fusion [s_in, g_in, gs_in]
    )
    |[s_in, g_in, s_out, g_out]|
    (
        I_Spaceball [i_spaceball, s_out, p_spaceball, s_in]
        |||
        (
            I_Gesture [i_posture, g_out, p_glove, g_in]
            |[i_posture]|
            I_Pose [i_glove, i_posture]
        )
    )
)
|[scene]|
I_Scene [p_scene, scene]
```

Behavioural constraints are further applied to the above specification in order to meet the interactional requirements of 3DDE. Following the GIO methodology, this is achieved by localizing the specification and considering only the subset of interactors whose behaviour needs to be constrained.

Lets consider the *synergistic usage of input devices* to paint an object selected by pointing at it with the dataglove while modifying the color with the space-ball. In this case the I_Fusion interactor is required to combine together the color selection made by pressing one of the spaceball buttons with the object selection made by means of the dataglove. The fusion is allowed within a temporal window within which both the s_in and the g_in actions must be performed involving the I_Spaceball, the I_Gesture, and the I_Fusion interactors. A controlling agent is required expressing exactly that behaviour. This is achieved by the following fragment of specification:

```
    (
        I_Fusion [s_in, g_in, gs_in]
        |[s_in, g_in]|
```

```
        (
            I_Spaceball [i_spaceball, s_out, p_spaceball, s_in]
            |||
            I_Gesture [i_posture, g_out, p_glove, g_in]
        )
    |[s_in, g_in]|
    ControllingAgent [s_in, g_in]
```

where the behaviour of the ControllingAgent is:

```
ControllingAgent [s_in, g_in] ::=    urge s_in, g_in in
                                     time timeout (waittime) in
                                     s_in;  ( g_in [] timeout )
                                     []
                                     g_in; ( s_in [] timeout )
```

The above fragment of specification can be graphically represented as in the following figure:

The above fragment of specification can be graphically represented as in the following figure:

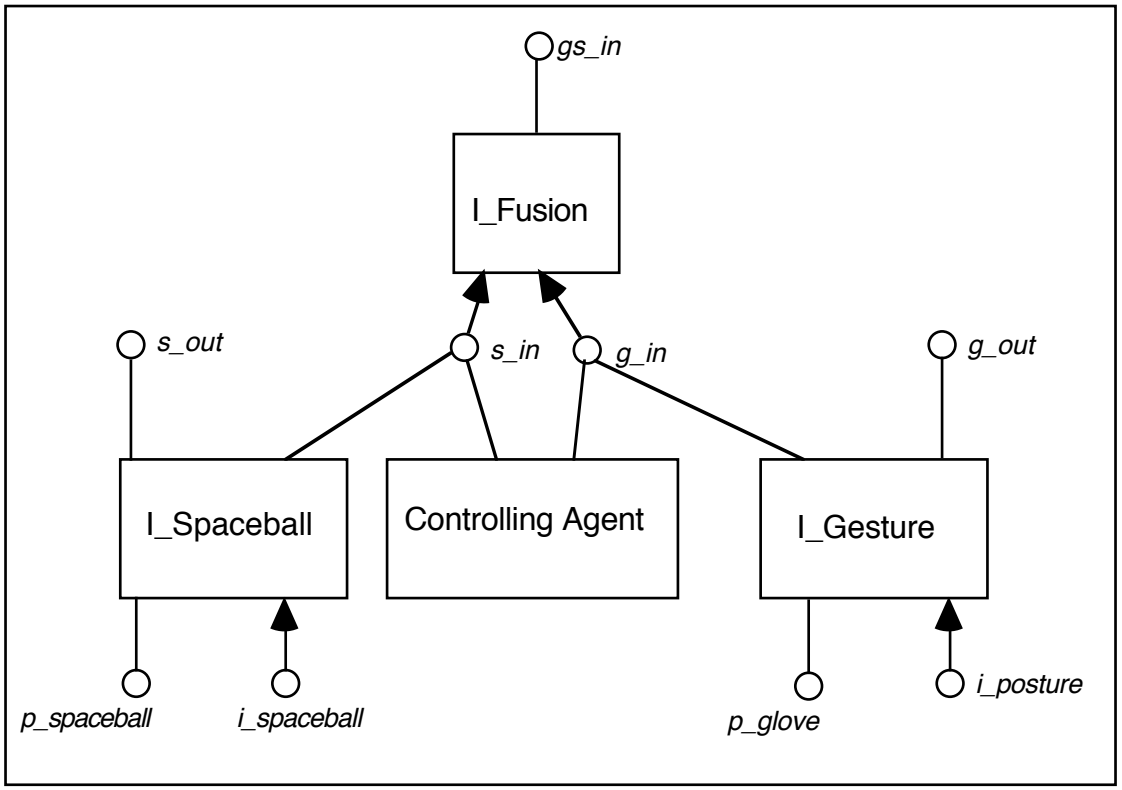


Figure 7: Applying constraints to the interactors modelling the 3DDE.

5 Integrating the two approaches

The GIO model of interactors is based on the process algebra approach underlying the LOTOS notation. It is used to formally describe the behaviour of a User Interface System from a high level of abstraction to low level details by means of successive refinement steps. Similarly, the PAC model can be used to recursively describe the organization of a User Interface System at any level of detail by taking an approach leading to a real implementation of such a system. PAC-Amodeus integrates technical constraints due to the existence of toolkits within the conceptual PAC approach. As a result, in PAC-Amodeus, agents are used to model the keystone component of the computer system.

The goal pursued by GIO is to prove behavioral properties of User Interface Systems resulting from the composition of several interactors. For example, GIO can show how the use of different physical devices (such as a dataglove versus a mouse or a spaceball) in a User Interface might lead to different system behaviours. The goal pursued by PAC and PAC-Amodeus is not to prove properties about a particular architecture but to help the software designer to specify a global conceptual software architecture that matches the external specifications of a computer system.

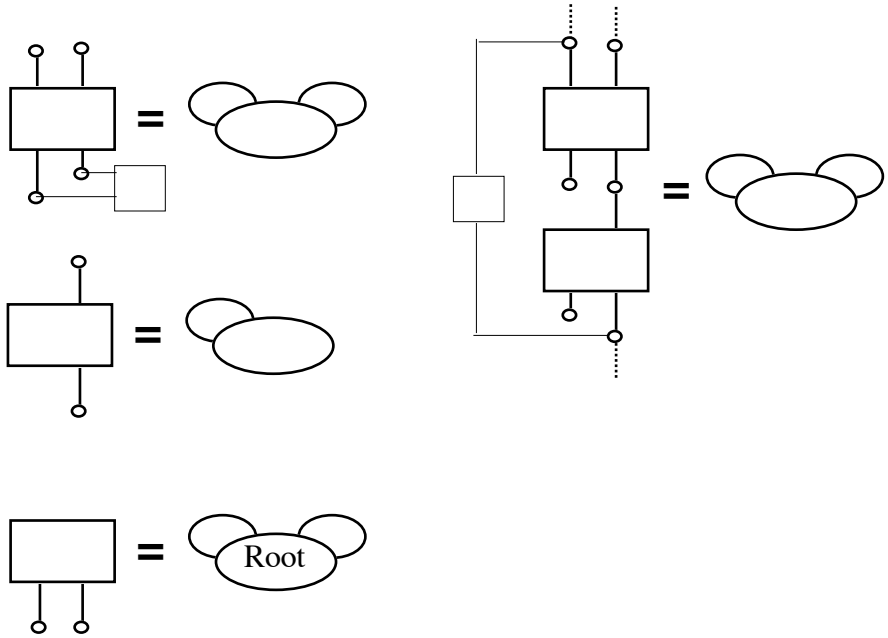
Apart from the overall motivation driving the two different approaches, a major difference between GIO interactors and PAC agents is concerning the structuring of their components. It is interesting to remark that such a

difference is induced by common driving principles in defining a GIO architecture or a PAC/PAC-Amodeus architecture. Reusability is one of such principles. A GIO interactor is essentially a composition of processes able to abstract and concretize respectively user and system generated tokens by following a hard-wired predefined behaviour: that is the controller of an interactor is simply defined by the syncronization occurring amongst its components. One heuristics adopted by the GIO designer to augment reusability is to use external controllers. A controller is an agent whose role is to define constraints on the sequences of observable input and output events an interactor can be engaged with. In other words, interactors are lego-like components externally customized by controllers. A similar reasoning applies to PAC. A PAC agent has exactly the same role as a GIO interactor. However, GIO controllers correspond to PAC agents whose presentation and abstraction parts are empty. Whereas PAC blurs the distinction between the notions of levels of abstraction and control, GIO makes it explicit: in PAC, agents which are not leaves in the hierarchy, are abstracting/concretizing processes or controllers of the information flow.

In the GIO model, the low level interators are the only agents that can engage interaction directly with the user. In PAC-Amodeus, this is not necessarily true. In addition to the vertical abstraction/concretization process, one can exploit an horizontal abstraction-concretization mapping facility: the abstract and presentation facets of the agent are then partially/totally implemented in the Functional Core Adaptor or the Presentation toolkit component. This is an example of how a conceptual model can be engineered to consider the existence of implementation tools in a realistic way.
We have identified two rules that may clarify the correspondence between the GIO and PAC approaches. Rule 1 shows how a GIO interactor or a set of GIO interactors match a PAC agent depending on their input an output facilities. In the top left column of Rule1, a full interactor with an optional controller is equivalent to a complete PAC agent. In the midle left, an interactor with no concretization and an optional controller is a PAC agent with no P facet. In the bottom left, an interactor with no abstraction output and no concretization input corresponds to the root agent of he hierarchy (we have reached the highest level of abstraction). In the right column of rule 1, a hierarchical composition of interactors may be bundled into one PAC agent (because the implementation tools that are available for the system provides the low level functionality covered by the low level interactors).

Rule 1:

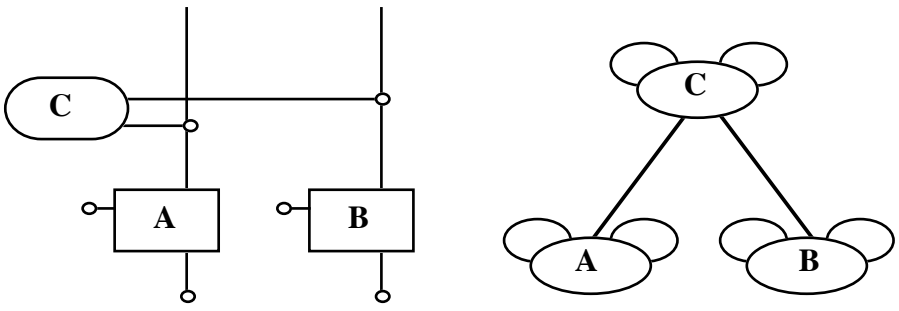
One interactor or a linear set of interactors with optional controller = one PAC agent



As shown by Rule 2, a GIO controller linking two otherwise independent interactors is modelled in PAC as an agent whose competence is to control the two agents.

Rule 2:

Two interactors linked by a controller = Two PAC agents, sons of one PAC agent



1. Tables used in the Dynamic Gesture System

The Input-Action Handler contains the following table, for mapping the gesture id used by the Gesture System and the function name associated with. The function name is then sent to the application.

gesture id	function name
#1	create
#2	paint
#3	connect SB
#4	delete
#5	scene zooming
#6	scene rotating

The Input-Action Handler also includes another table for mapping a recognized gesture (or pose) into a cursor (virtual tool).

gesture id	cursor (parameters)
no gesture	hand
#1	"application"
#2	brush (color)
#3	"application"
#4	rubber
#5	hand
#6	hand

The Feedback System is then in charge of visualizing the cursor associated with the gesture being recognized. If no gesture is recognized, a rendered hand is visualized.
Sometimes, the visualization of the cursor is left to the application. For example, that happens for the "create" function. In this case, the cursor takes the shape of the 3D object being created, and selected either picking it up or selecting the Spaceball buttons. It is up to the application to show the proper cursor according to the selected object.
In some other cases, some parameters used for displaying the cursor are sent by the application to the Gesture System. That happens, for example, for the "paint" function. The brush color is decided by the application and communicated to the Gesture System.

The application uses the following table for associating a function with other parameters and for sending some information to the Gesture System.

function name	event	information to GS
paint	SB button 1	color :: red
	SB button 2	color :: blue
	SB button 3	color :: green
	SB button 4	color :: yellow