The AMODEUS Project
ESPRIT Basic ResearchAction 7040

CERD Preliminary Modelling Report from the PAC perspective

Nigay, L., Salber, D. & Coutaz, J.

LGI-IMAG

15th May 1994

Amodeus Project Document: SystemModelling/IR9

CERD Preliminary Modelling Report from the PAC perspective

Summary: This document is an early analysis of the CERD exemplar performed from the PAC-Amodeus perspective. It is not self-contained: knowledge of the PAC-Amodeus modelling approach is necessary for a good understanding of the CERD description in terms of the PAC concepts.

Design issues addressed in this report:

- **design issue 1: Awareness of pending messages**
- **design issue 2: Software design of proactive feedback for flights selection**
- **design issue 3: Syntax of commands**
- **design issue 4: Assign a code to a flight**
- **design issue 5: Scribble line: multimodal output**
- **design issue 6: Undo a request sent after clicking confirm**
- **design issue 7: Two dedicated screens: one for the display, one for the operations**
- **design issue 8: Direct manipulation and one single screen**
- **design issue 9: Two-handed interaction**

Awareness of pending messages

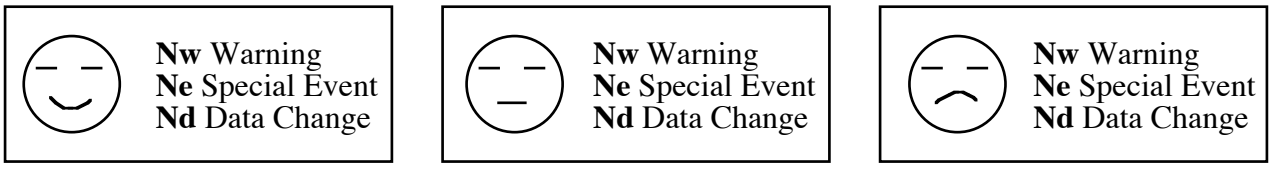
There are three categories of messages, from the highest priority to lowest one: Warnings, Special Event and Data Change. All the messages are displayed one after an other in the message area. To display the next message in the queue, the user has to touch the message area. The number of messages key next to the message area displays the number of all pending messages (including the current one). There is no indication of the categories of the pending messages. Thus, the user is not aware of possible waiting messages which have a higher priority than the current one.

We therefore propose three solutions. These solutions present three levels of informative display:

a) The simplest solution shows the user if there is a pending message with a higher priority than the current one. To make this information perceivable by the

user, different possibilities are available: blinking the number of messages key, adding a small icon next to the number of messages, highlighting the key. This design decision is linked to the importance of the role of the messages for the ATCO.

b) At any time, the system displays the number of pending messages for each category. Three display zones are therefore necessary instead of one. To gather this information in a synthetic way, a status man icon can be used.



(Nw=0) et (Ne=0) et (Nd>=0) (Nw=0) et (Ne>0) et (Nd>=0) (Nw>0) et (Ne>=0) et (Nd>=0)

Figure 1. The status man showing the number and category of pending messages.

c) The system allows the user to scroll the list of queued messages. The scrollable list contains all the messages which have not been acknowledged. Two arrows next to the message area allow up and down scrolling. The messages in the list are ordered according to their priority. Once a message is acknowledged (by clicking on it in the message area), it is then discarded from the list.

Figure 2. The scrollable message area.

PAC D1	Awareness of pending messages

PAC Architecture modelling

For each of the above solutions, the Functional Core Adapter (FCA) receives the messages from the network which is part of the Functional Core (FC). It maintains three queues of messages according to the categories. Thus, the FCA adapts the data from the FC to make them perceivable by the user through the Dialogue Controller (DC). Within the DC, a dedicated PAC agent, “Message”, manages the display of the presentation of the messages. At the arrival of a message inside the FCA, the FCA notifies the Message agent through its abstraction part.

For solution a), the abstraction part of the Message agent maintains the number of messages of each category. Moreover, the abstraction part maintains the category

of the current displayed message. This agent is then able to notify the user when there is a pending message with a higher priority than the current one.

For solution b), the abstraction part of the Message agent maintains the same information as in solution a) and its control part performs the mapping function between the current state (number and categories of pending messages) and the status man icon to be displayed.

For solution c), the abstraction part of the Message also contains the contents of the messages. This design choice is guided by efficiency reasons: since the list of messages is now scrollable, the messages contents should be maintained by the agent. The presentation part of the agent now contains two supplemental buttons to allow up and down scrolling.

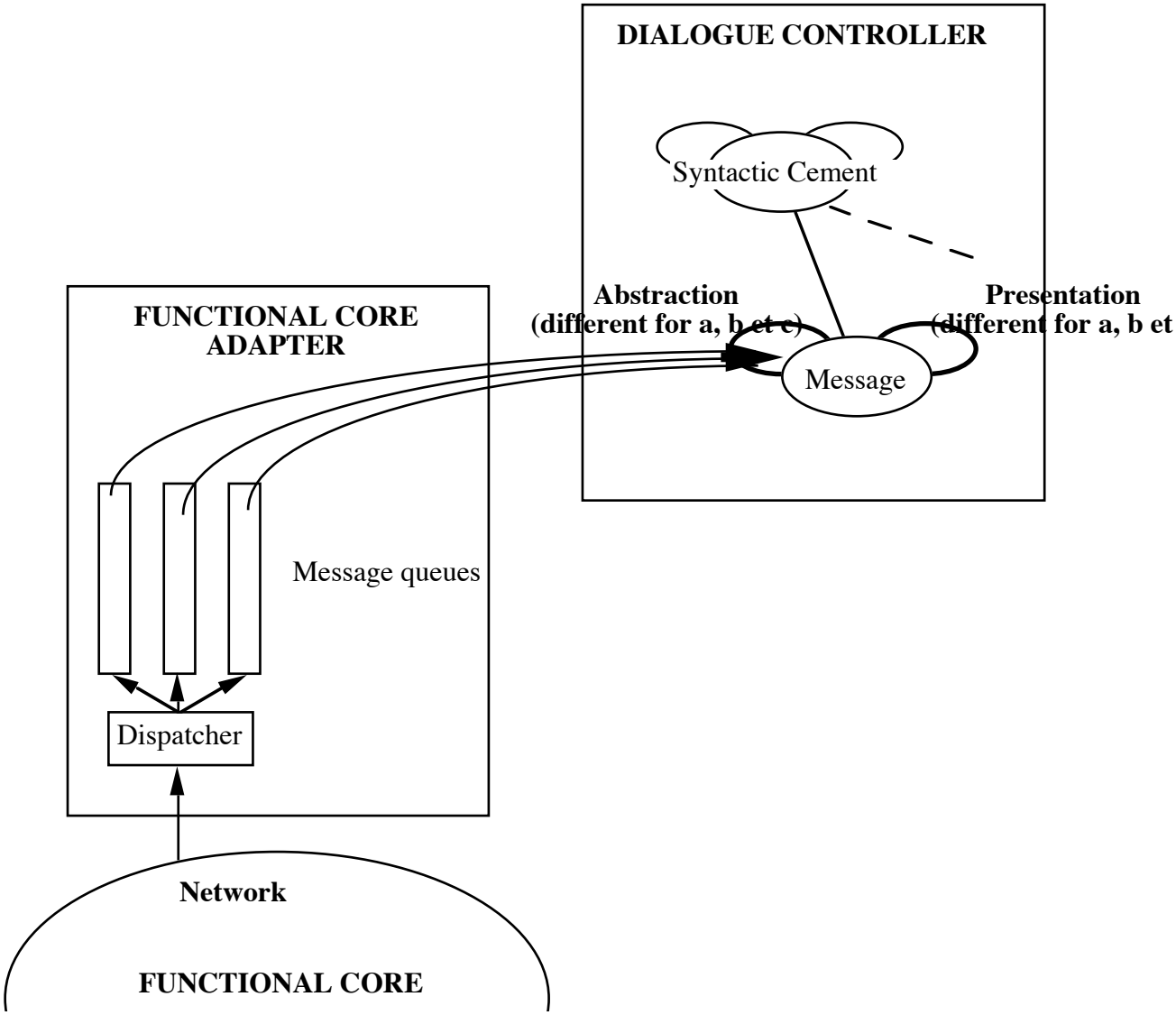


Figure 3. Architecture for solutions a, b and c

Software design of proactive feedback for flights selection

When an operation is chosen by the user, the keys representing available flights for this operation are highlighted. This is an example of proactive feedback and has some influences over the software architecture. Figure 6 shows the message passing inside the hierarchy of PAC agents to implement proactive feedback. The FCA maintains the list of available flights. When an operation is initiated by the Commands Palette agent, the corresponding operation is sent to the syntactic cement agent. The abstraction part of this agent then asks the FCA for the available flights by sending a rule to compute the valid flights for this operation. After receiving the list of flights, the agent asks the matrix agent to highlight the corresponding flights. The control part of the matrix agent maintains a table to map flights identifiers to the flight PAC agents. This table is used to dispatch the highlight message to the target flight PAC agents.

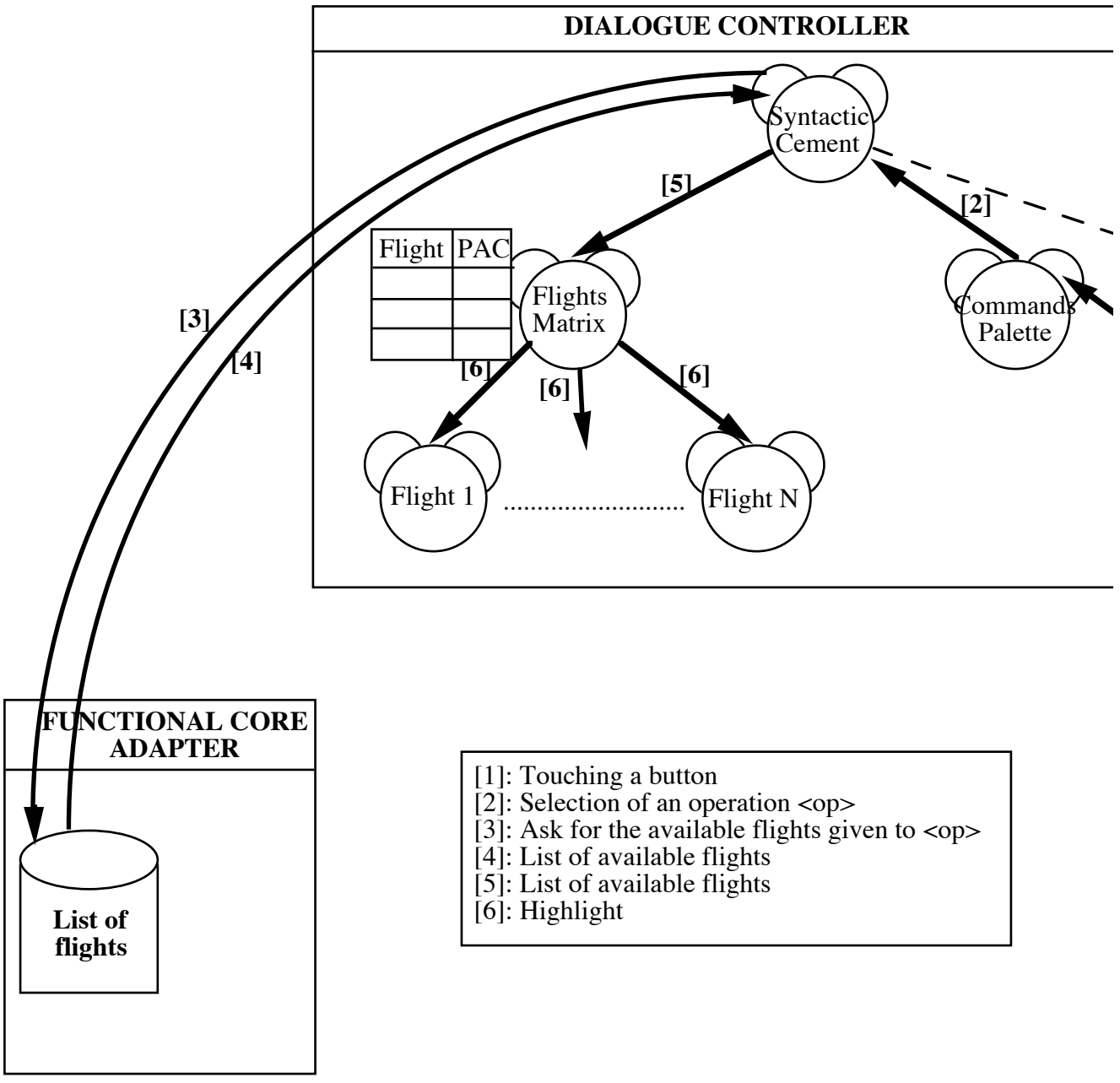


Figure 4. Message passing to implement proactive feedback

PAC I1	Proactive feedback for flights selection

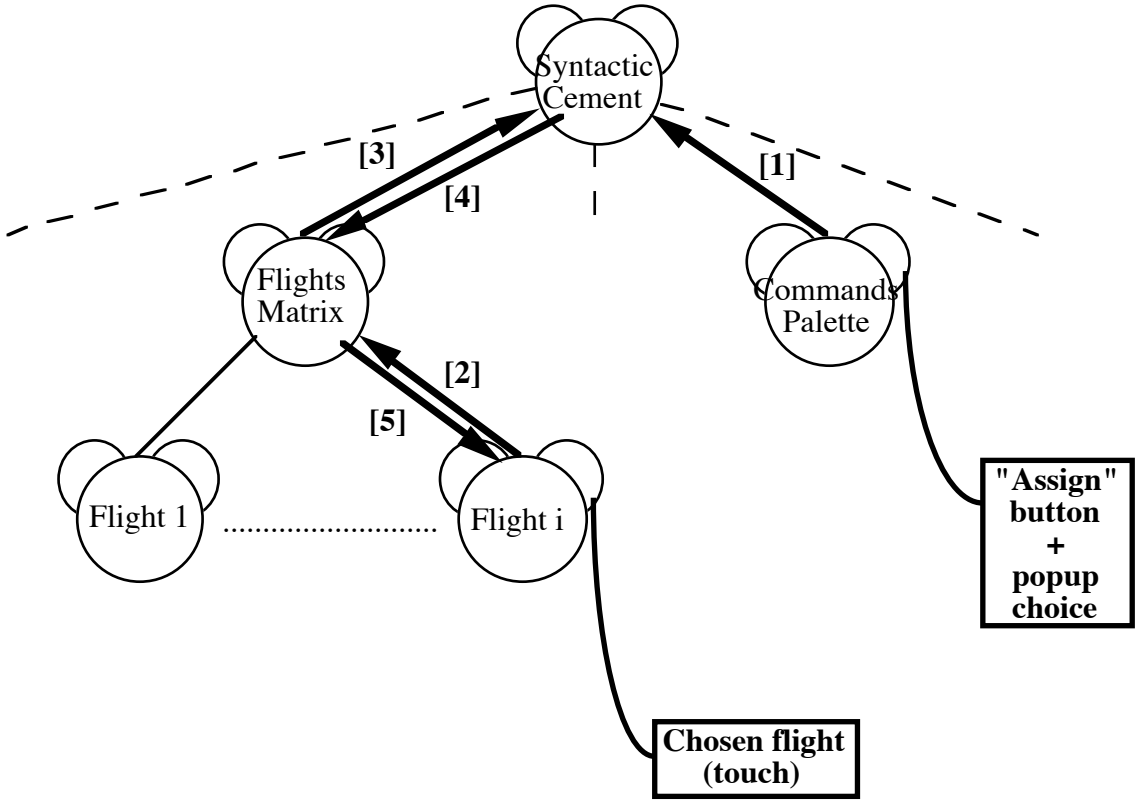
Syntax of commands

In all four tasks (Swap, Reposition, Resequence, Assign), the syntax of commands is strictly defined: command followed by parameters. We propose to extend this syntax to allow the parameters to be specified before the command. To do so, we don't need to modify the above hierarchy of agents (see Figure 6). If a flight is first selected, the syntactic cement agent will maintain the selection in its abstraction part. When the operation is then selected, the cement agent will complete the command.

PAC D2	The syntax of commands should allow more flexibility

Assign a code to a flight

The assignment of a code to a particular flight (Assign command) is an example of a passive command: it is therefore managed within the DC and doesn't involve the FC. The assignment command is a case of semantic repair. The concept of code assigned to a flight is not a task-domain concept and is not present in the FC. Since there are only three codes in use, the three choices can be presented as a pop-menu available by clicking the Assign button. Figure 7 shows the message passing within the architecture when the user assigns a code to a flight.



[1]: selection of the assign operation + value of the popup choice
[2]: selection of this flight (i)
[3]: selection of flight i in the matrix
[4]: assign value of flight i in the matrix
[5]: assign value of this flight (i)

Figure 5. Message passing for the assignment command: only the DC is involved.

PAC I2	Assignment of codes to flights

PAC D3	A popup menu of codes would avoid a dedicated screen to assign a code to a flight

Scribble line: multimodal output

The scribble line shows commands as they are specified. For example the reposition operation requires two flight parameters. They are displayed both in the scribble line and highlighted in the matrix. Two output interaction languages are used simultaneously using the same output device (i.e., screen). The same piece of information (i.e., the selected flights) is rendered at the same time using two different languages: it is therefore a case of redundancy. The rendering redundancy suggests that the piece of information is located in a single location in the architecture (i.e., abstraction part of an agent). A good place to store this

information is the abstraction part of the syntactic cement agent. This agent handles the scribble line and is also a common father agent of both the Commands Palette and Matrix agents.

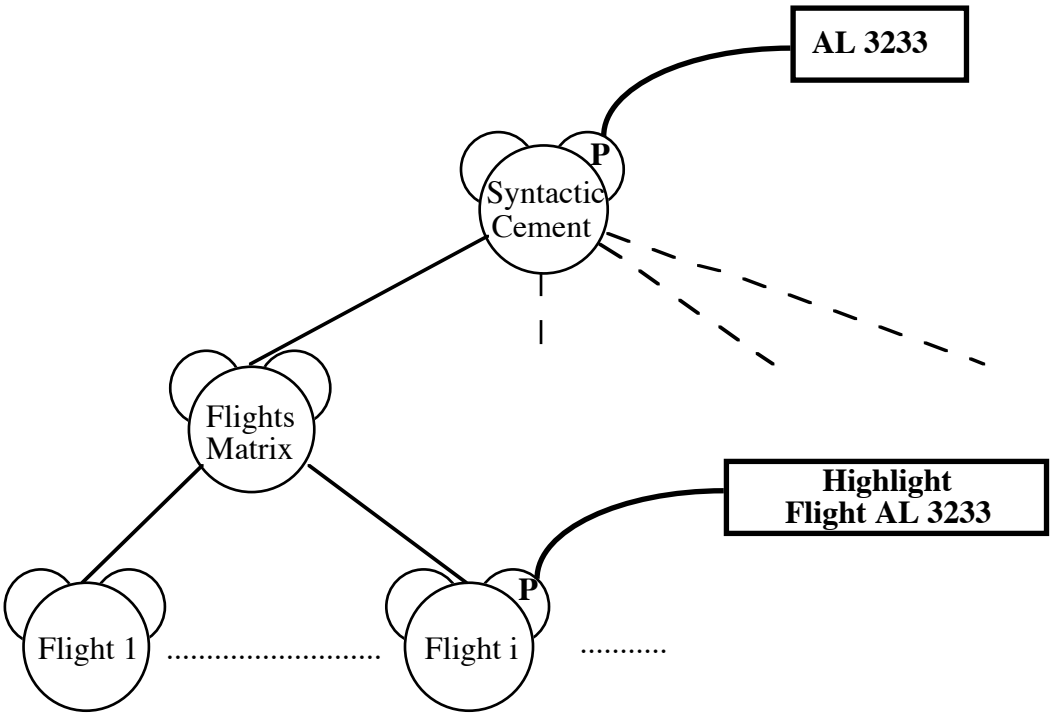


Figure 6. Multimodal output: rendering redundancy while specifying the Reposition command.

PAC I3	Multimodal output for rendering the construction of a command

Undo a request sent after clicking confirm

When a command is confirmed by clicking the Confirm button, it is then sent over the network through the FCA and cannot be cancelled anymore. To alleviate this problem and allow the cancellation of a confirmed command, the FCA maintains the last request sent over the network. If the request needs to be undone, the FCA computes the corresponding cancellation request according to the stored request. This mechanism provides a one-level undo. A multi-level undo could be realised by maintaining an history of requests inside the FCA.

PAC D4	Add undo facility

Two dedicated screens: one for the display, one for the operations

When specifying the parameters of a command, a new screen is displayed, thus replacing the flights matrix. To avoid this screen swapping and to enhance screen stability, a second touch panel display device can be used. The main display always shows the matrix of flights. The second one is dedicated to the specification of commands. This disposition allows the user to be always aware of the most relevant information (i.e., the matrix of flights). The agent hierarchy of the DC is the same as shown on figure 7. The choice of the display device to render the presentation of an agent is performed inside the Low Level Interaction Component (LLIC) of the PAC-Amodeus model. Since there are two output devices, the system is then multimodal for output. This two-display configuration is similar to the CUBRICON system described in [Neal 91].

PAC D5	A second display device to make the flight matrix perceivable by the user at any time

Direct manipulation and one single screen

Another way to insure screen stability is to allow direct manipulation on the flights matrix. For reposition and resequencing operations, the user can drag directly on the screen the flights to be moved within the matrix. To prevent manipulation errors, the user has to press a button to enter the “drag mode”. The current mode (i.e., drag or not) must be perceivable by the user. For example, the mode change can be performed by clicking a toggle button with two visible states (e.g., the mode is shown by the label of the button). Once the user has performed the modification on the matrix by dragging, he can either confirm or cancel by touching the appropriate button. This will return the system to normal mode (as opposed to drag mode). This solution leads to a simplified agent hierarchy. The agent sub-hierarchy dedicated to the management of the screens which allow the user to specify the parameters of commands is no more relevant. Instead, the specification of the parameters is handled by the matrix agent and its son agents. Each son agent handles a given matrix element. This agent decomposition supports a simple implementation of direct manipulation.

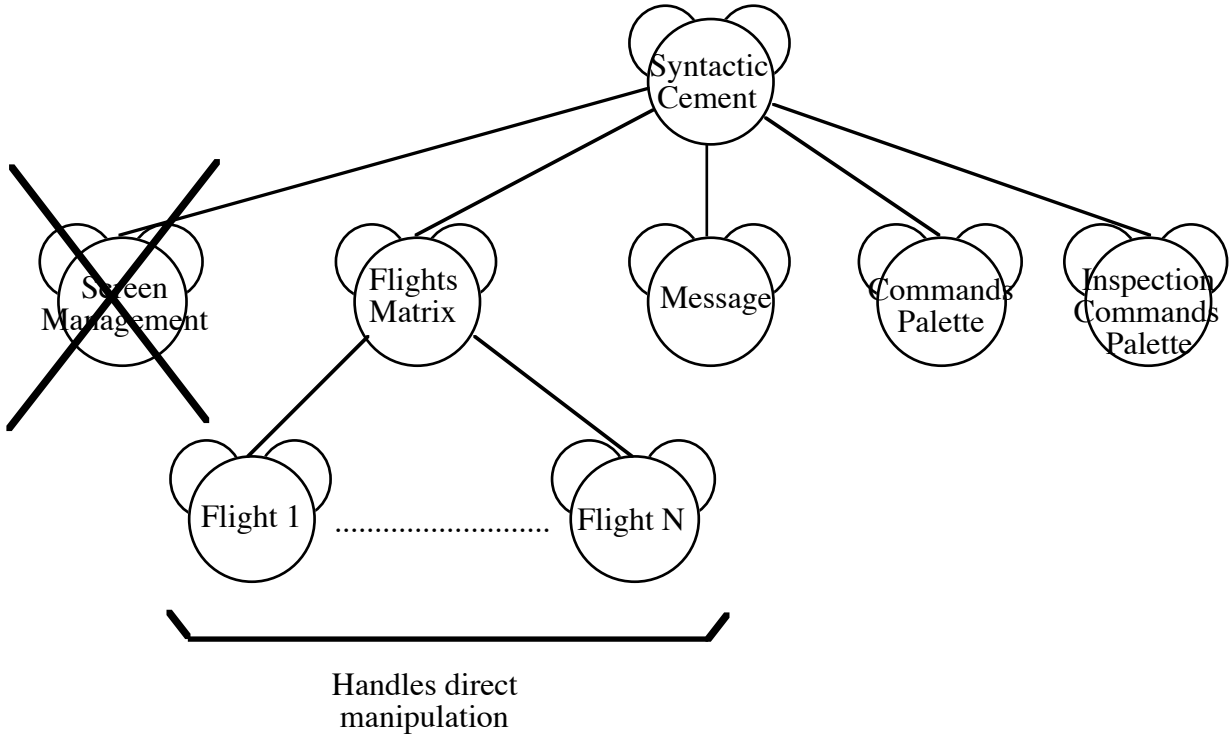


Figure 7. With direct manipulation,
the screen management sub-hierarchy are no longer required.

PAC D6	Allow direct manipulation of the flights matrix

Two-handed interaction

"Direct manipulation interfaces using a pointing device could be more efficient with the addition of a second pointing device." [Chatty 94] Starting from this statement, we can apply a two-handed interaction in the design of CERD. It is particularly suitable for the Swap command. Real-world manipulation and swapping of objects involves two handed actions. To swap to flights, the user simultaneously selects the two flights using both hands and moves them. Nevertheless, the current touch screen technology doesn't support such two-handed interaction: for two simultaneous inputs, the touch screen will provide two x values and two y values instead of two (x, y) couples. Thus, two simultaneous inputs define four (x, y) combinations and therefore four points on the screen. From a software architecture point of view, the handling of two parallel inputs is easy to handle because independent agents in the hierarchy are able to function simultaneously.

PAC D7	Two-handed interaction for Swap operation

References

[Chatty 94]

S. Chatty, Issues and Experience in Designing Two-Handed Interaction , CHI'94 Conference Companion, Boston, USA, pp. 253-254.

[Neal 91]

J.G. Neal and S. Shapiro, Intelligent Multimedia Interface Technology, Intelligent User Interfaces, J.Sullivan and W.Tyler (eds.), ACM Press, Frontier Series, Addison-Wesley publisher, 1991, pp. 11-43.