Project Nr 3066
AMODEUS
(Assimilating Models of DEsigners, Users and Systems)

BUILDING USER INTERFACES: ORGANIZING SOFTWARE AGENTS

L. NIGAY, J. COUTAZ

Laboratoire de Génie Informatique (IMAG)
BP 53 X, 38041 Grenoble Cedex, France
fax: (33) 76 44 66 75, tel. (33) 76 51 48 54
email: nigay@imag.fr, joelle@imag.fr

SUMMARY

This article presents the results of the system architecture modelling activity performed in the context of the Esprit BRA 3066 AMODEUS project. The goal of this research package is to provide insights and effective guidelines for designing and implementing interactive software. We propose to represent an interactive system at different levels of granularity ranging from a revisited version of the Seeheim model up to a highly parallel multi-agent PAC organization. Although multi-agent models are considered to be a promising way of supporting human factors as well as software engineering requirements, they are hard to apply. This article provides a set of heuristic rules that support the software design task.

KEYWORDS

interactive system, software architecture modelling, multi-agent model, PAC.

1. INTRODUCTION

The definition of an appropriate software architecture is an important issue in user interface design. Without an adequate architectural framework, the construction of interactive systems is hard to achieve, the resulting software is difficult to maintain and iterative refinement is made impossible. Software tools, such as toolkits, application skeletons, and UIMS's, designed to support the construction of user interfaces, do not alleviate the problem.

Toolkits such as Motif (1), application skeletons such as MacApp (2), and presentation-driven UIMS such as Interface Builder (3), do not embed the architectural principles that a software designer needs. For example, the "callback procedure" paradigm made popular by X Window (4), does not make any distinction between "semantic-domain dependent" concepts and "user interface-presentation dependent" issues. Thus, without an adequate software framework, the resulting interactive system may be an incredible mixture of concerns. Software tools for the construction of user interfaces will not solve architectural problems as long as the construction of these interfaces requires programming. Clearly, a reference model for identifying and organizing the components of interactive software is still a necessity.

Since Seeheim (5), a number of software architecture models have been devised and discussed in multiple workshops and working groups (6, 7). We observed at one special interest group on UIMS's at the CHI'91 conference, that all of these models are too general to be useful; they are too ambiguous, thus leading to multiple possibly wrong interpretations. One such misinterpretation is the sequential linguistic approach mapped onto the Seeheim model (8). Other approaches such as PAC (9) and MVC (10), put the emphasis on highly parallel architectures organized in terms of cooperative agents.

Multi-agent models offer some promising foundations for software engineering (e.g. modularity and distribution). As important, they support the opportunistic behavior of the end-user as well as multiple parallel inputs and outputs (9). However, our own experience with PAC revealed that, given the external specifications of a particular system, software designers have trouble identifying the agents necessary to the implementation of that system. This article aims to provide a preliminary cookbook for defining such agents.

This work has been performed in the context of AMODEUS, Esprit BRA project no 3066. Section 2 summarizes the goals of AMODEUS and indicates how our action fits in this general framework. In section 3, we provide a general description of the architecture of an interactive system and indicate the key components of that architecture. The central component is then refined in terms of agents. The structure of these agents as well as guidelines for defining their inter-relationships are described in sections 4 and 5 respectively.

2. THE CONTEXT: THE AMODEUS PROJECT

The AMODEUS project investigates how the various kinds of modelling techniques in HCI for representing the user, the system, and the design space, can be integrated into a unified framework (11). As a means of unification, the scenario methodology has been adopted (12, 13): a set of scenarios drawn from real experiments are analyzed by the different modelling approaches. The outcome of the exercise provides a basis for understanding the strengths and weaknesses of each modelling approach, their interrelationships and patterns of complementarity.

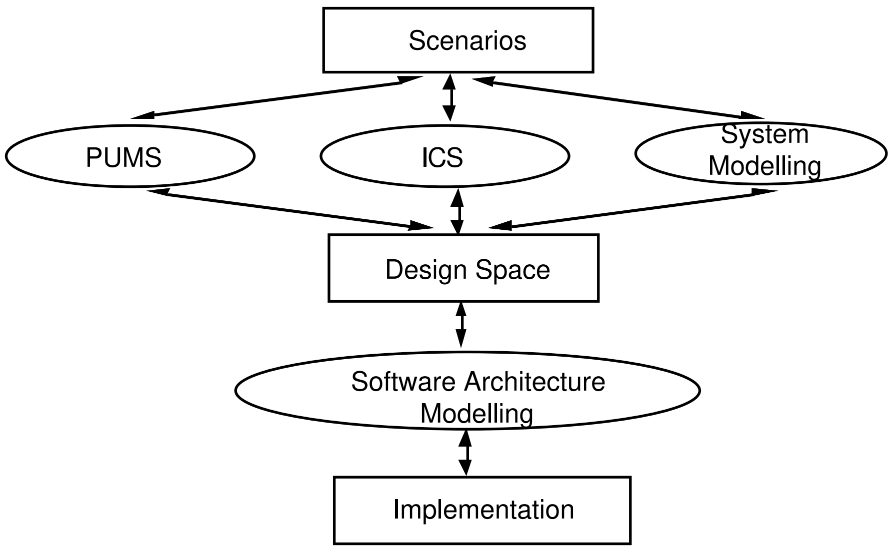

Fig. 1: The modelling techniques of the AMODEUS project.

As shown in Figure 1, a scenario is used as "tool for thought". It is an instance of a pertinent class of problems expressed in natural language. For example, a scenario may propose two design alternatives such as fish-eye views or the scrolling mechanism for displaying information on the screen. The system modelling technique expresses the problem in a formal way (14). Based on an unambiguous expression of the problem, user modelling techniques such as ICS (15) and PUMS (16) make predictions about user's behaviour and warn the designer of usability problems. The formal description of the problem as well as predictions and warnings inform the design space. Software architecture modelling (our approach) starts with the design solutions selected in the design space and leads to the implementation of these solutions. Like the other modelling approaches, it informs the design space in terms of specific criteria. Salient criteria at this level mightinclude efficiency and reusability, for example.

As shown in Figure 2, software architecture modelling activity in AMODEUS is a two step process:

- the first step is based on heuristics reasoning: a software architecture is derived for a particular design solution which, in turn, is drawn from psychological knowledge in HCI. This architecture makes explicit the agents involved in the software solution;

- formal reasoning is then used to describe the behaviour of the agents as well as their relationships in a precise and unambiguous way. Such description leads the way to the automatic generation of user interfaces.

In this article, we are concerned with the heuristic approach only. The formal notation, which combines CSP and Z descriptions, can be found in (17).

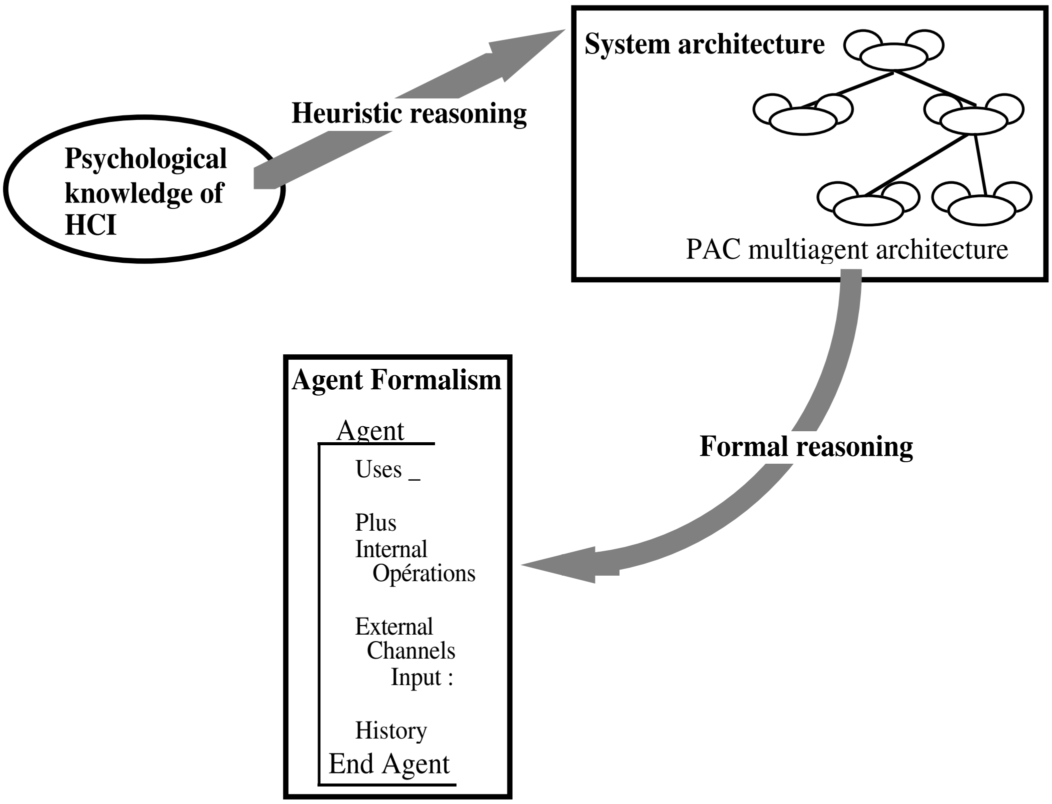

Fig. 2: Heuristic and formal reasoning used in AMODEUS for designing user interface architectures.

3. OVERALL ORGANIZATION OF AN INTERACTIVE SYSTEM

There is a consensus in user interface software design that there is an important distinction between the functional core and the user interface. The functional core of an interactive system is the software portion which implements domain dependent concepts. The user interface denotes the presentation (i.e. the *image* as defined by D. Norman in (18)). This distinction has been admitted as a sound basis for iterative refinement and helped in the emergence of the UIMS technology.

The Seeheim model refined the user interface portion of an interactive system into three components: the interface with the functional core, the dialogue controller and the presentation component. The role of each component was roughly described as the semantic, syntactic and lexical functionalities of the user interface. As mentioned above, the imprecision of this definition opened the way to multiple interpretations.

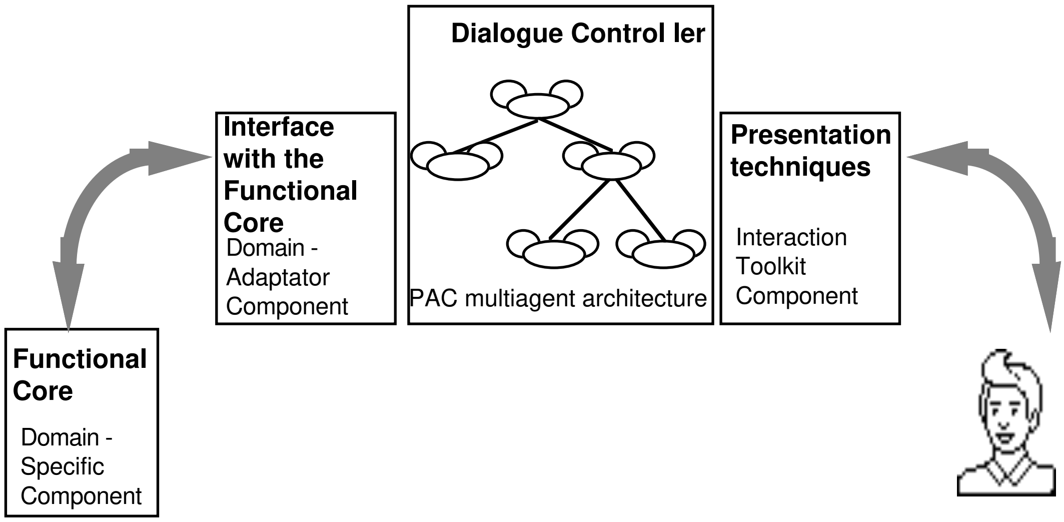

Fig. 3: The components of an interactive system.

Figure 3 shows our view of a general model of an interactive system. One salient property of this model is a nice balance organized around a key component: the dialogue controller. At the two extremes, the functional core and the user play a symmetric role at a high level of abstraction (the domain concepts). Next, come the two interface components: one for the functional core, a second one for the interaction toolkit. These components are discussed further in the following paragraphs.

The User and the Functional Core

The user and the functional core both produce and consume information (e.g. events) through the dialogue controller. This symmetric view of the general functioning of an interactive system where the key component is equally driven by the functional core and the user does not impose any model on the control of the interaction. The question of external VS internal VS mixed control (19) is irrelevant at this level of description. Clearly, the choice between the alternatives is guided by the case at hand.

This balance is in constrast with earlier interpretations of the Seeheim model which imposed an *a priori* scheme on the control of the interaction. First generation UIMS's, inspired by the compiler technology, found it very convenient to view the functional core as a semantic server. This external control of the interaction was adequate for domains such as graphics and text editing; it was inappropriate for control process systems where the functional core may produce information in an asynchronous manner from the point of view of the user (e.g., alert messages in a nuclear station security system).

Interface with the Functional Core

The interface with the functional core (IFC) serves as a buffer between the dialogue component and the domain specific concepts implemented in the functional core. It is designed to absorb the effects of change in its direct neighbours. As does any boundary, it implements a protocol. A protocol is characterized by temporal strategies, by the nature of data exchanged and by some linking mechanisms. These three dimensions are discussed in (20). Data exchanged need further explanation.

The IFC is a good place for semantic enhancement and semantic delegation. Both of these design operations can be expressed in terms of conceptual objects. (In our discussion, the term "object" covers the encapsulation of a set of data and operators.) A conceptual object is supposed to match the mental representation that the user elaborates about a particular domain concept. It may be the case that the functional core, driven by software or hardware constraints, implements a domain concept in a way that is not adequate for the user. For example, the concept may be split into multiple data structures.

Semantic enhancement may be performed in the IFC by defining a conceptual object which reorganizes the data of the functional core. Reorganizing may take the form of glueing together data structures of the functional core into a single conceptual object or, conversely, splitting a concept into multiple conceptual objects. Semantic enhancement may also take the form of a conceptual object that extends the properties of a domain concept. For example, the "status-man" described in (21), which counts the occurrences of semantic errors according to their level of gravity, extends the notion of error. It provides the user with a synthetic view of the semantic validity of the entity being built with the system. Counting errors is certainly not relevant to the functional core although it provides the user with useful task dependent information. A conceptual object is a good way to solve the problem.

Semantic delegation may be performed in the IFC to enhance performance. The semantic quality of feedback may require frequent round trips to the functional core. This long chain of data transfer may be costly with respect to time. Therefore response time may be inconsistent with the expectation of the user. Semantic delegation, which consists of down-loading functional core knowledge into the user interface is a way to reduce transmission load. In particular, if the functional core is implemented as a distinct process running on a distinct processor, it may be judicious to use the IFC as a cache of the data maintained in the functional core. As described in the next section, semantic delegation may be performed in the dialogue contoller as well.

The Presentation Techniques

The presentation techniques component (PTC) implements the physical interaction with the user via hardware and software. It includes a set of interaction techniques that defines the *image* of the system (output). It also deals with the events produced by the user (input). If we consider current practices, presentation techniques are constructed from concepts made available in user interface toolkits.

The PTC is the location for handling low level multi-media events (for input as well as for output). For example, low level input events produced by voice and gesture input devices may be time-stamped and queued in the same way mouse events and keyboard events are currently processed by windowing systems. Multi-media events should be combined into higher level events in the same way as keyboard and mouse events are currently combined into higher abstractions by toolkit dialogue boxes.

The Dialogue Controller

The dialogue controller is the key component of our model. Contrary to a number of user interface models, the dialogue controller is not a monolithic "obscure potato"! Instead, it is organized as a set of cooperating agents which smoothly bridges the gap between the IFC and the PTC. The nature of these agents and their relationships will be further described in the next sections. For now, we need to justify the overall functionality of the dialogue controller.

The overall functionality

The dialogue controller has the responsibility for task-level sequencing. Each task or goal of the user corresponds to a thread of dialogue. This observation suggests the choice of a multi-agent model. Indeed, a multi-agent model distributes the state of the interaction among a collection of cooperating units. Modularity, parallelism, and distribution are convenient mechanisms for supporting multi-thread dialogues. One agent or a collection of cooperating agents can be associated to each thread of the user's activity. Since each agent is able to maintain its own state, it is possible for the user (or the functional core) to suspend and resume any thread at will.

The dialogue controller receives events both from the functional core via the IFC, and from the user via the PTC. Bridging the gap between an IFC and a PTC has a number of consequences. In addition to task sequencing, the dialogue controller must perform data transformation and data mapping:

1) An IFC and a PTC use different formalims. One is driven by the computational considerations of the functional core, the other is toolkit/media dependent. In order to match the two formalisms, data must be transformed inside the dialogue controller.

2) State changes in the IFC must be reflected in the PTC (and vice versa). Therefore links must be maintained between conceptual objects of the IFC and presentation techniques in the PTC. As discussed in (20), a conceptual object may be rendered with multiple presentation techniques. Therefore, consistency must be maintained between the multiple views of the conceptual object. Such mapping is yet another task of the dialogue controller.

The Dialogue Controller at Multiple Resolution

Bridging the gap between the IFC and the PTC covers task sequencing, formalism translation, and data mapping. Experience shows that these operations must be performed at multiple levels of abstraction and distributed among multiple agents.

Levels of abstraction reflect the successive operations of abstracting and concretizing. Abstracting combines and transforms events coming from the presentation techniques into higher level events for higher abstractions. Conversely, concretizing decomposes and transforms high level information into low level information. The lowest level of the dialogue controller is in contact with the presentation techniques provided by the toolkit. This boundary is a rather fuzzy frontier.

In general, user interface toolkits such as the X Intrinsics, provide an abstraction mechanism for defining new interaction techniques. However, it is not always possible to build new interaction techniques from the predefined building blocks of the toolkit. For example, in an earlier version of the X intrinsics, interaction techniques would occupy rectangular areas only. In such conditions, the notion of a wall in a floor plan drawing editor, could not be implemented as a diagonal line widget. Instead, a presentation object "wall" would be defined as a new abstraction in the dialogue controller portion.

This example shows that the frontier between the dialogue controller and the presentation techniques component depends on the facilities provided by the user interface toolkit and the presentation requirements of the interactive system: specific presentation requirements that can be built from the building blocks of the toolkit belong to the interaction techniques component. Those which cannot be built with the toolkit are part of the dialogue controller.

As discussed above, the multi-agent approach is a promising way to support parallelism, distribution, multi-thread dialogues, and iterative design. Since agents should carry task sequencing, formalism transformation, and data mapping at multiple levels of abstraction, it is tempting to describe the dialogue controller at multiple grains of resolution combined with multiple facets.

At one level of resolution, the dialogue controller appears as a "fuzzy potato". At the next level of description, the main agents of the interaction can be identified. In turn, these agents are recursively refined into simpler agents. This description is nothing more that the usual abstraction/refinement paradigm applied in software engineering. Orthogonal to the refinement/abstraction axis, we introduce the "facet" axis. As described in the next section, an agent is also described along three facets: Abstraction, Presentation, Control. These facets are used to express different but complementary and strongly coupled computational perspectives.

Figure 3 shows the recursive description of the dialogue controller at multiple grains of resolution combined with the 3 facets dimension.

4. A PAC AGENT

In general, a PAC agent in the dialogue controller is composed of three facets (for a more complete description of the PAC model one can refer to (22) chapter 8, pp 161):

- The Presentation implements the perceivable behaviour of the agent. It is related to some presentation technique(s) of the PTC.
- The Abstraction implements the competence of the agent (i.e. its expertise) in an essentially media independent way. It maintains the abstract state of the agent and defines an interface with other agents. It may be related to some conceptual object(s) of the IFC. The abstraction facet defines the location for performing semantic delegation.
- The Control part of an agent is in charge of two functions: linkage of the Abstraction part of the agent to its Presentation portion, and maintenance of the relationships of the agent with other agents. The linkage serves two purposes: formalism transformations between the Abstraction and the Presentation portions of the agent, and data mapping between the abstract facet and the presentation facet. Relationships between agents may be static or dynamic. Dynamic relationships are required when agents are dynamically created/deleted. Relationship maintenance by the control part of an agent covers the communication and the synchronization mechanism between this agent and its cooperating partners.

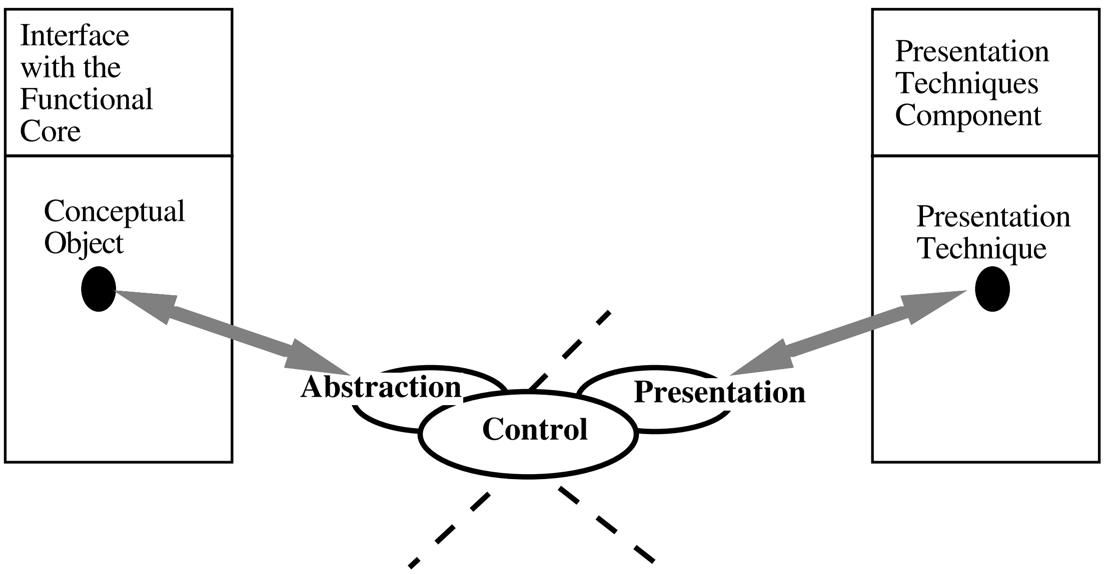

Fig. 4: A PAC agent of the Dialogue Controller. Dashed lines represent possible relationships with other agents. Dimmed arrows show the possible links with the surrounding components of the dialogue controller.

Figure 4 shows how a PAC agent relates to other PAC agents and to the surrounding world of the dialogue controller: conceptual objects in the IFC and presentation techniques in the PTC. If we consider the dialogue controller as a whole:

- the set of Abstraction parts of the various agents defines the internal state of the interaction,
- the set of Presentation parts defines the external state of the interaction, and
- the set of Control parts defines the mapping functions between the internal and the external state. Some properties of these functions are defined in (14).

This general description of the dialogue controller does not provide enough insight about how to define the agents for a particular interactive system. This is the topic of the next section.

INSIDE THE DIALOGUE CONTROLLER: FROM MODEL TO REALITY

The picture shown in Figure 4 is an idealized view of the reality. Experience shows that some agents do not require all the facets advocated by the model. Early experiences with PAC has lead us to define a taxonomy for PAC agents as well as guidance for the abstraction/refinement process of agents. The result of our reflection is presented in the following two paragraphs.

A dimension space for PAC agents

PAC agents may be classified along two dimensions: linkage with conceptual objects and facets.

Linkage with conceptual objects

An agent may or may not be linked to a conceptual object. When linked to a conceptual object, it participates in a chain of data transformations and data mappings. This chain ends with the agent whose presentation part delivers the conceptual object to the user.

Agents that are not linked to any conceptual object are not domain dependent but they support the accomplishment of the task. They are a form of semantic enhancement delegated into the user interface. For this reason, we call them "syntactic agents". For example, in a visual programming environment system, the functional core maintains the abstract tree of the program being built whereas an agent in the dialogue controller may be used to trace program execution.

Facets of a PAC agent

A PAC agent always includes a control part. Since it links an agent to its environment, a control part necessarily plays a key role in the existence of an agent. PAC agents, however, need not include either an Abstraction facet or a Presentation facet.

Empty Abstraction occurs in two circumtances:

1) the agent, which is used as a presentation object only, has no particular competence. It may result from an extension of the toolkit unable to provide the appropriate facilities for defining a new interaction technique;

2) the agent is in direct correspondence with a conceptual object. In order to avoid duplication of information, the abstract competence of the agent is left in the conceptual object. In such a case, the abstraction is limited to the identification of the conceptual object.

Agents may have no presentation. As in the case of *view controllers* in Serpent (23), they are used as computational units to maintain relationships between multiple agents. In the next section, we discuss a number of such relationships.

Rules for identifying PAC agents

Agents may be used to represent combinations of relationships and levels of abstraction. We provide a set of rules that may help in the design process of building dialogue controllers in terms of agents. A more complete description can be found in (24).

Distributed Syntax

Use an agent to cement actions distributed over multiple agents into a higher abstraction.

It is often the case that the specification of a command involves actions distributed over multiple agents. This is particularly true for CAD application systems where a palette, which is an agent, displays the possible options and where a window, which is yet another agent, is used as a scratch area to create and edit an artefact (e.g. a picture in a graphic editor). For example, drawing a circle in an interactive system such as MacDraw implies selecting the circle icon in the palette then drawing the circle in the scratch area. The combination of the two actions has a meaning: that of creating a circle but the palette and the scratch area are not aware of each other. They would otherwise be dependent on each other and thus the first one would not be reusable without the existence of the second one. In addition, multi-thread dialogue authorizes logically connected actions to be interleaved with "foreign" actions. As a consequence, there is a conflicting need for maintaining "mutual ignorance" between two agents, for remembering logically connected actions and for combining them into a higher abstraction (e.g. a command). The multi-agent framework provides a natural way for satisfying such requirements: introducing an intermediary agent which serves as a cement. Figure 5 shows the general solution.

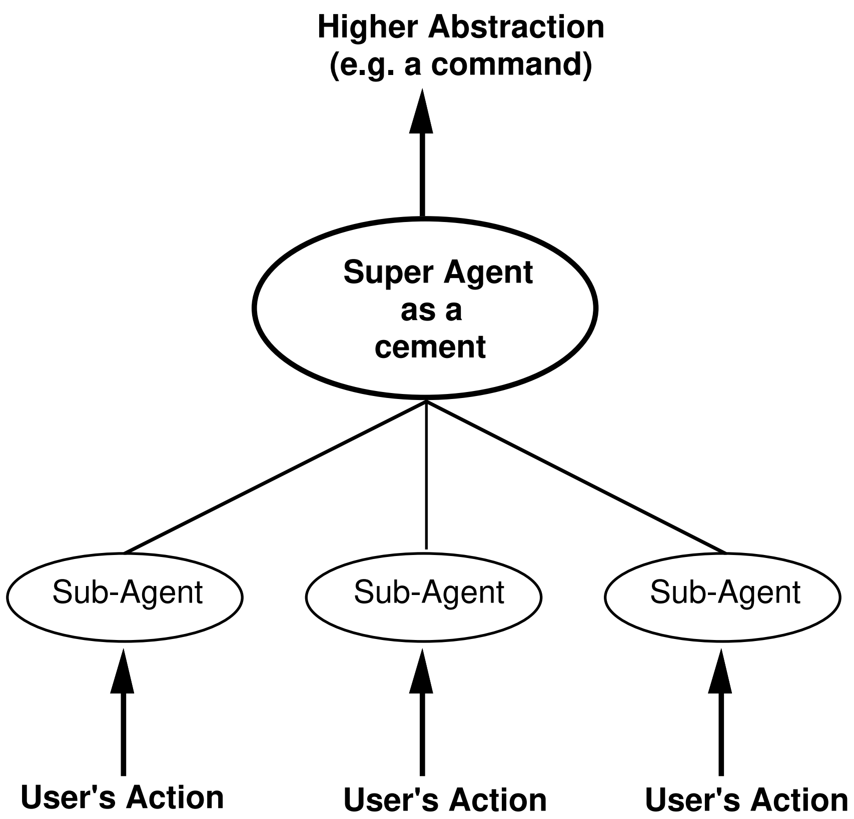

Fig. 5: Use of an agent to cement logically connected and distributed actions into a higher abstraction.

The cement agent behaves like a syntax analyzer which controls the local automata maintained by its subagents. By doing so, input for syntax analysis may come in any order from multiple sources. If, in addition, the cement agent is able to handle the construction of several commands simultaneously, it provides an easy way to implement multi-thread dialogues.

Multiple Views

Use an agent to maintain visual consistency between multiple views.

One semantic concept of general interest is that of "focus of attention". In a document preparation system, the "focus of attention" is a document. The problem with the existence of multiple views on the same concept is the maintenance of visual consistency. Any action on one view should be reported in the other views.

According to the software principle which stresses "mutual ignorance" to enhance reusability, agents which implement the views of a semantic concept should not know each other. As shown in Figure 6, a "Multiple view" agent is introduced to express the logical link. Any action with visual side effect on a view is reported to the Multiple view which broadcasts the update to the other siblings.

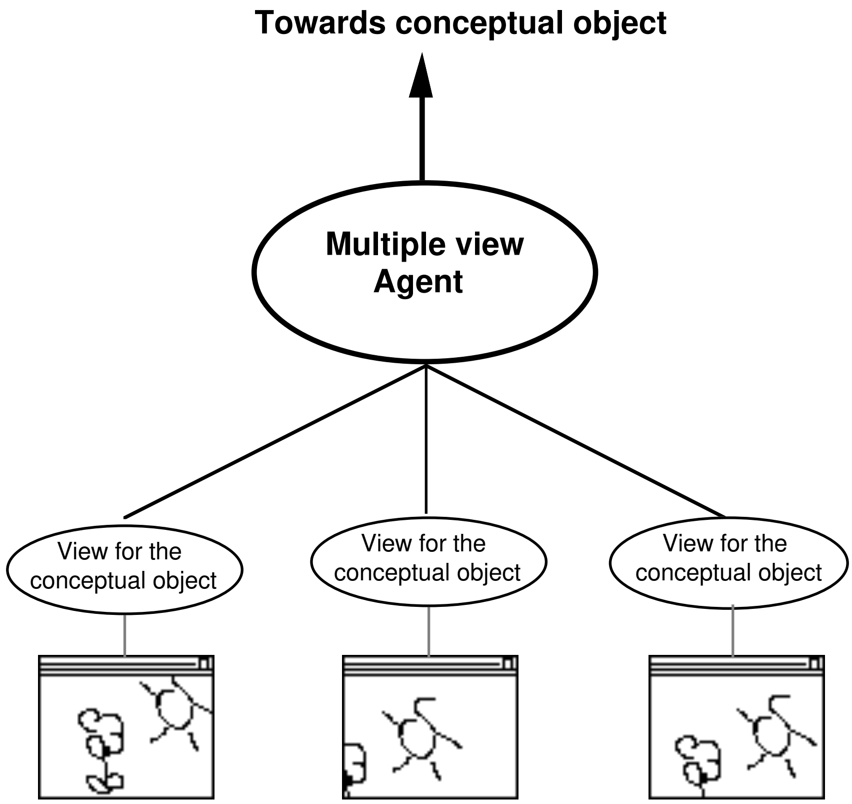

Fig. 6: Use of an agent to maintain visual consistency between multiple views of the same conceptual object.

Conveying composed conceptual objects

Use isomophic agents to reflect the composition of conceptual objects.

It is often the case that conceptual objects are composed of conceptual objects. For example, a program is composed of modules which in turn is composed of procedures, etc. If the relation of composition must be conveyed to the user, then one can define a similar composition in terms of agents in the dialogue controller. A hierarchy of agents isomorphic to the hierarchy of conceptual objects is thus defined.

6. CONCLUSION

We have presented some preliminary results related to our heuristic approach to the design of interactive software architectures. Our contribution is three-fold:

1) We have clarified the description of the overall architecture of an interactive system making explicit the boundaries with the functional core and the presentation toolkit, two components considered by programmers as unavoidable constraints.

2) We have refined the dialogue controller into a number of agents modelled along two dimensions: facets and levels of abstractions. The combination of these two axes is a convenient way of describing phenomena observed from current practices.

3) We have identified a number of heuristic rules which provide useful insights and guidance for the definition of relationships between agents.

In the near future, we will formalize our design heuristics rules and develop an expert system which will be able to automatically suggest the agents appropriate for a particular system.

ACKNOWLEGMENTS

This paper was influenced by stimulating discussions with our colleagues and observers of the IFIP WG 2.7, with members of the AMODEUS project: G. Abowd, Bala, L. Bass, M. Beaudouin-Lafon, M. Harrison, I. Newman, P. Dewan, J. Larson, C. Unger, H. Stiegler. A. Chabert of the IHM group of LGI and V. Normand of Bull-Imag also deserve special thanks for their contribution to our model.

REFERENCES

(1) OSF/Motif, Programmer's Reference Manual, Revision 1.0; Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 02142, 1989.
(2) K. SCHMUCKER. MacApp: An Application Framework; Byte 11(8), 1986, pp. 189-193.
(3) B. F. WEBSTER. The NeXT Book; Addison Wesley, 1989.
(4) R.W. SCHEIFLER, J. GETTYS. The X Window System; ACM Transactions on Graphics 5(2), April, 1986, pp. 79-109.
(5) User Interface Management Systems; G. E. Pfaff ed., Eurographics Seminars, Springer-Verlag, 1985.
(6) D.A. DUCE, M.R. GOMES. Hopgood, Lee (eds): User Interface Management and Design; Proceedings of the Workshop on UIMS and Environments, Lisbon, june 1990; Springer Verlag Publ., 1991.
(7) L. BASS, R. LITLTLE, R. PELLEGRINO, S. REED, R. SEACORD, S. SHEPPARD AND M. R. SZEZUR. The Arch Model: Seeheim Revisited; User Interface Developers' Workshop; April26, 1991, Version 1.0.
(8) J. COUTAZ. UIMS, Promises, Failures and Trends; HCI'89, People and Computers V, A. Sutcliffe, L. Macauley eds., Cambridge University Press, 1989, pp. 71-84.
(9) J. COUTAZ. PAC, an Implemention Model for Dialog Design; Interact'87, Stuttgart, September, 1987, pp. 431-436.

(10) A. GOLDBERG. Smalltalk-80: The Interactive Programming Environment; Addison-Wesley Publ., 1984.
(11) ESPRIT BRA 3066. Assimilating Models of Designers, Users and Systems (AMODEUS), Technical Annex, CEC, Brussels, 1989.
(12) R. YOUNG, P. BARNARD. The Use of Scenarios in Human Computer Interaction reserach: Turbocharging the tortoise of cumulative science; In Proceedings of the CHI+GI'87 Conference (Toronto); ACM, 1987, pp. 291-296.
(13) N. HAMMOND, P. BARNARD, J. COUTAZ, M. HARRISON, A. MACLEAN, R. YOUNG. Modelling User, System and Design: Results of a Scenarios Matrix Exercice; Panel in Proceedings of the CHI'91 Conference (New Orleans, 28 April-2 May); ACM, 1991, pp. 377-380.
(14) M. HARRISON, H. THIMBLEBY (eds). Formal Methods in Human Computer Interaction; Cambridge University Press, 1990.
(15) P. BARNARD. Cognitive Resources and the Learning of Human Computer Dialogs; in Interfacing Thought, Cognitive Aspects of Human Computer Interaction, J.M. Caroll Ed., MIT Press Publ., 1987, pp. 112-158.
(16) R.M. YOUNG, T. GREEN, T. SIMON. Programmable user models for predictive evaluation of interface design; In Proceedings of the CHI'89 conference, 1989, ACM press, pp. 15-19.
(17) M. HARRISON, G. ABOWD. Formal methods in Human Computer Interaction; Tutorial #27, Computer Human Interaction Conference, 1991.
(18) D. A. NORMAN, S. W. DRAPER. User Centered System Design; Lawrence Erlbaum Associates, Publishers, 1986.
(19) P. TANNER, W. BUXTON. Some Issues in Future User Interface Management Systems (UIMS) Development. IFIP Working Group 5.2 Workshop on User Interface Management, Seeheim, West Germany, November, 1983.
(20) COUTAZ, S. BALBO. Applications: a dimension space for user interface management systems; In Proceedings of the CHI'91 Conference (New Orleans, 28 April-2 May); ACM, 1991, pp. 27-32.
(21) J. Bass, J. Coutaz: Developping Software for the user interface; Addison Wesley, 1991.
(22) J. COUTAZ. Interfaces Homme-Ordinateur: Conception et Réalisation; Dunod, 1990.
(23) L.J. BASS, E. HARDY, K. HOYT, R. LITTLE, R. SEACORD. The Serpent run time architecture and dialogue model; Carnegie Mellon Technical Report, CMU/SEI-88-TR-6, January, 1988.
(24) Software Architecture Modelling, RP2, Deliverable D2, september, 1990.