PAC-Expert:
Towards an Automatic Generation of Dialogue Controllers

L. Nigay, J. Coutaz
LGI-IMAG, University of Grenoble
BP 53 X
38041 Grenoble, Cedex, France

Mai 1992

paru également comme Document D18, Projet BRA Amodeus 3066

Résumé

Ce rapport décrit PAC-Expert, un système expert qui inclut une méthode de conception d'architecture logicielle répondant au modèle PAC-AMODEUS. A partir de la description du comportement externe d'un système interactif, PAC-Expert produit l'architecure PAC-AMODEUS correspondante. La représentation graphique de l'architecture constitue la spécification interne du système qui doit, selon les préceptes du génie logiciel, précèder la conception interne détaillée.

Mots-clés

Modèle PAC-AMODEUS, PAC-Expert, modélisation d'architecture logicielle, interface homme-machine.

Summary

This report describes PAC-Expert, an expert system that embeds a method for designing software architectures according to the PAC-AMODEUS model. From the description of the external behaviour of an interactive system, PAC-Expert is able to generate a PAC-AMODEUS software architecture for that system. The graphical expression of this architecture constitutes the internal specification of the system before detailed internal specifications of the system must be produced.

Keywords

PAC-AMODEUS model, PAC-Expert, Software architecture modelling, computer-human interaction.

Table of contents

Introduction

The process of designing and constructing user interfaces is critical to building systems that satisfy the customer's needs, both current and future. This process includes the original design of the interface, the implementation of the system, and the modifications to the operational system. These modifications are endemic in interactive systems. Since the user interface can account for approximately 50 per cent of total life cycle costs [Myers 89], the software engineer has a vested interest in constructing a user interface that both satisfies the customer and is constructed using the best available tools and techniques.

Tools that support the development of user interfaces vary widely in complexity and power ranging from user interface toolkits to user interface generators. Although powerful, user interface generators cannot produce everything. Therefore, the user interface must be tuned to the specific case at hand. In turn, customization requires programming, and good programming practice necessitates an architectural framework such as PAC-AMODEUS.

PAC-AMODEUS, based on PAC [Coutaz 87] and refined in the AMODEUS project, is a multi-agent model that represents the organization of the components of an interactive software. PAC-Expert is the companion tool which embeds a method for designing PAC-AMODEUS architectures. Based on the description of the external behaviour of an interactive system, PAC-Expert is able to generate the representation of the software architecture for that system in terms of the PAC-AMODEUS model.

The structure of this working paper is as follows: Section 1 presents the life cycle of an interactive system and indicates where in the design process PAC-Expert comes into play. Section 2 discusses the complementary roles of models and methods. This dictinction will be used to relate PAC-AMODEUS with PAC-Expert. Section 3 summarizes the main features of PAC-AMODEUS and Section 4 is devoted to the description of PAC-Expert per se. An example of a session with PAC-Expert is given in Appendix A. Appendix B summarizes our test experiments using the AMODEUS examplars.

1. PAC-Expert and the software engineering life cycle

In software engineering, design and implementation are structured into 6 phases: definition, specification, implementation, testing, installation and maintenance [ANSI 83]. Basically,

1. *Requirements definition* is a formal or semi-formal statement of the problem to be solved. It specifies the properties and services that the system must satisfy for a specific environment under a set of particular constraints. Requirements are defined in cooperation with the customer.

2. *Specification* consists of high level design (i.e., external specifications) and internal design (i.e., internal specifications). High level design is concerned with <u>the external behavior</u> of the computer system. This behavior is described in terms of functionalities as perceived by the user of the future system. For each function, valid inputs and outputs are specified as well as error conditions. <u>Internal design</u> determines a software organization that satisfies the specification resulting from high level design. Internal design covers the definition of data structures, algorithms, modules, programming interfaces, etc.[1]

[1] In software engineering, the term "design" covers the specification phase. Since we are concerned with drawing a distinction between designing the user interface (as in HCI) and designing the software

3. *Implementation* is the expression of the internal specification in terms of a set of programming languages and tools.

4. *Testing* involves debugging individual modules as well as performing their integration.

5. *Installation* consists of placing the software system into production.

6. *Maintenance* deals essentially with changes along with their side-effects in the software life cycle.

A more complete description of each phase can be found in [Bass-Coutaz 91, chapter 1, pp. 1-9].

PAC-Expert is concerned with the *Specification phase* of the software engineering life cycle. Its role is to provide help for identifying the software organization of the code that implements the user interface portion of an interactive system. Thus, within the specification phase, PAC-Expert is related to internal design.

Figure 1: Where PAC-Expert comes into play in the software design cycle.

As shown in Figure 1, input to PAC-Expert is a description of the external behavior of the interactive system. As discussed above, this description results from the high level design phase (external specifications). The output of PAC-Expert is a representation of a PAC-AMODEUS software architecture which satisfies the specified external behavior. This representation is a first step towards detailed internal specifications. In turn, detailed specifications serve as input for the implementation phase. Within the framework of AMODEUS, the agent language developed at York is the natural candidate for expressing the internal detailed specifications for PAC-AMODEUS architectures [Abowd 91]

Although based on a general framework, the current version of PAC-Expert deals with Graphical User Interfaces only (i.e., GUI technology). More advanced user interfaces which allow the combination of multiple modalities, will be considered in a further study.

2. Model and Method with PAC-AMODEUS and PAC-EXPERT

Models and methods are closely related notions. Generally speaking, a method provides a guide that assists the design and implementation process of an artefact (e.g., the architecture of an interactive system). In order to be manipulated, the artefact must have a representation. In turn, the representation is the reflection of a model suitable for reasoning in the problem space. Thus, a method is a set of directions (or procedures) to use a particular model which underlies the representation of the artefact being designed and built.

In software engineering, we must make a further distinction between models and methods used during the design phase of a software, and models and methods applicable to the implementation phase. PAC-AMODEUS and PAC-Expert sit at the frontier between software design and software implementation.

---

that implements the user interface, we will use the term specification for this phase of the life cycle.

PAC-AMODEUS is a model applicable to the design of interactive software architectures. Designing a software architecture consists of defining the software units of the interactive system that satisfy the external specifications of that system. In PAC-AMODEUS, these units are defined based on the notion of agent. An instance of a PAC-AMODEUS architecture is a representation of the software architecture of a particular interactive system; this instance results from the application of a design method associated with the PAC-AMODEUS model.

Our own experience with PAC-AMODEUS reveals that software designers have trouble identifying the appropriate agents which, at run time, will support the behaviour described by the external specifications of the interactive system. Although PAC-AMODEUS has interesting properties with regard to interaction principles, it is too general to guide the software architecture design process. The goal of PAC-Expert is to bridge the gap between external specifications and internal specifications. It does so by embedding the PAC-AMODEUS method.

3. PAC-AMODEUS, a software architecture model for interactive systems

As shown in Figure 2, the PAC-AMODEUS model is a refinement of the Arch/Slinky Model [Bass 91]. In turn, the Arch/Slinky model defines itself as a “Seeheim revisited” model.

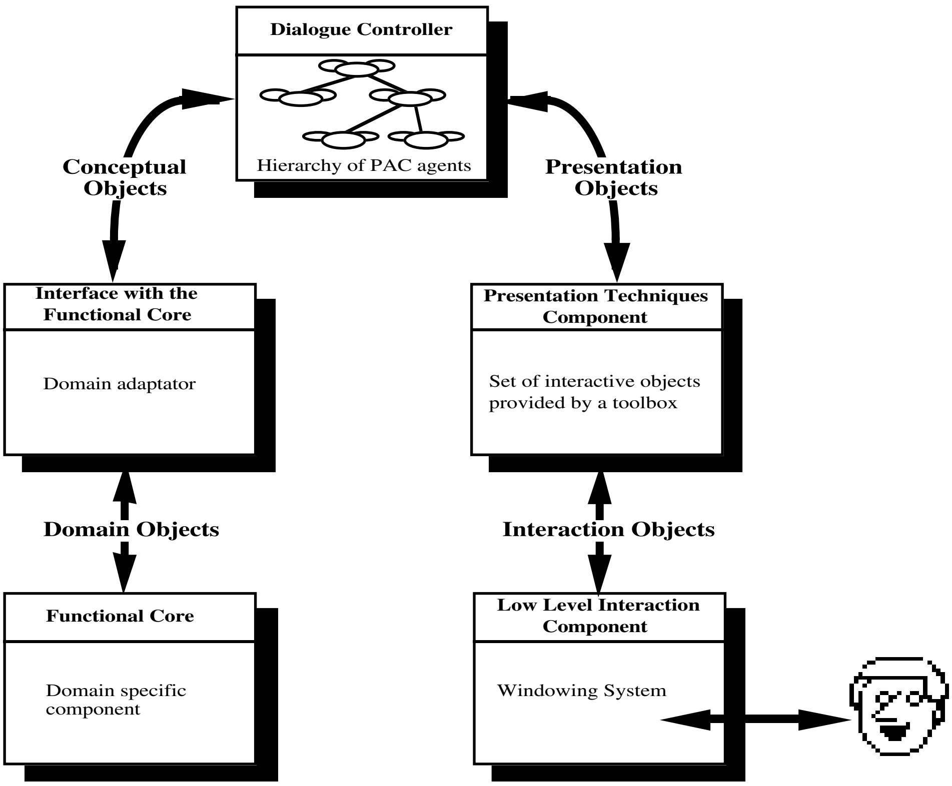

Figure 2: Components of an interactive system and the interface between them.

PAC-AMODEUS reuses the main components advocated by Arch/Slinky: an interactive system is comprised of the functional core, the interface with the functional core, the dialogue controller, the presentation technique component, and the low level interaction component. At the opposite of Arch/Slinky which does not provide any hint about how to organize the dialogue controller, PAC-AMODEUS refines this component in terms of PAC agents. The role of PAC-Expert is to automatically identify these agents as well as their relationships from the specification of the external behavior of the interactive system.

A more detailed description of PAC-AMODEUS can be found in [Nigay 91a]. The following paragraphs provide a synthesized definition of the various components illustrated in Figure 2.

The Functional Core (FC) implements the domain specific concepts. Ideally, the representation used to model domain concepts in the functional core is independent from the rendering of these concepts to the user. In general, this representation is driven by computational considerations.

The Interface with the Functional Core (IFC) serves as a domain adaptor between the functional core and the dialogue controller. Data exchanged with the functional core are the domain objects that the functional core exports towards the user [Coutaz 91]. Data exchanged with the dialogue controller are conceptual objects, that is perspectives on domain objects. Such perspectives are supposed to match the user's mental representation of domain concepts. They transform comptutational objects into abstractions driven by considerations for the user's conceptual model.

At the other end of the spectrum, the **Low Level Interaction Component** (LLIC) denotes the underlying platform, both software and hardware. The LLIC includes basic interaction facilities such as a windowing system, a spoken language shell, and toolkits such as Motif™ and OpenLook™. It manages the physical interaction with the user: physical actions from the user are modelled as typed events which are time-stamped, queued, and analysed. Those events that are part of lexical tasks such as window moving and window resizing, are processed locally by the LLIC. In the case of spoken-utterances, the LLIC may include mechanisms for confirmation allowing the user to intercept wrong recognitions or alleviate ambuiguities. From the developer's perspective, the LLIC is an unavoidable component. It exists just like a constraint exists.

The Presentation Techniques Component (PTC) serves as an adaptor between the Dialogue Controller and the Low Level Interaction Techniques. Whereas the Interface with the Functional Core performs transformations at the conceptual level, the PTC performs adaption at the rendering level. The communication with the Dialogue Controller is expressed in terms of Presentation objects. Presentation Objects convey data to be rendered to the user.

Presentation objects are translated in terms of interaction objects. The distinction between presentation objects and interaction objects is subtile. A presentation object may be modelled as an abstract interaction object or may be a new interaction technique built from LLIC components. Thus, the PTC defines a layer for portability between LLIC's components as well as a layer for extending LLIC services.

For example, when the user interface of a system is expressed in terms of ready for use interaction objects, then the mapping between presentation objects and interaction objects is 1-to-1: presentation objects exist to ensure portability with regard to the LLIC component. On the other hand, if the user interface of the system has very specific requirements, then new classes of interaction techniques must be defined: presentation objects are used as extensions of interaction objects.

As an example of extension performed inside the PTC, let us consider the Presentation Object that renders the concept of temperature in the form of a thermometer. The LLIC component includes graphic primitives, buttons and menus but does not provide thermometers. Thus, a new class (thermometer) will be defined in the PTC layer. This presentation object class will be expressed in terms of the basic interaction objects such as Draw-Line, Draw-Rectangle. It may as well be mapped into a voice message such as: "Current temperature is x° degrees Celsius".

The Dialogue Controller (DC) is the keystone of the model. It 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 architecture. An 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 Interface with the Functional Core (IFC), and from the user via the Presentation Technique Component (PTC). In addition to task sequencing, bridging the gap between an IFC and a PTC requires data transformation and data mapping:

1) A IFC and a PTC use different formalisms. 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 IFC conceptual objects and PTC presentation objects. A conceptual object may be rendered with multiple presentation techniques. Therefore, consistency must be maintained between the multiple views of the conceptual object. Such management is yet another task of the dialogue controller.

Experimental results suggest that task sequencing, formalism transformation, and data mapping 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 objects into higher level events until the IFC is reached. Conversely, concretizing decomposes and transforms high level data from the IFC into low level information. The lowest level of the dialogue controller is in contact with presentation objects. 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 applies the usual abstraction/refinement paradigm used in software engineering. In addition to the refinement/abstraction axis, we introduce the "facet" axis. Facets are used to express the different but complementary and strongly coupled computational perspectives of an agent. These perspectives are similar to those identified for the whole interactive system:

1) the functional core or Abstract facet (i.e., the A facet) defines the competence of the agent in the chain of abstracting and concretizing. It may be related to some conceptual objects in the IFC;

2) the dialogue controller facet (i.e., the C facet) controls event sequencing inside the agent, maintains a mapping between the IFC facet and the presentation facet of the agent;

3) the presentation facet (i.e., P facet) is involved in the implementation of the perceivable behaviour of the agent. It is related to some presentation object in the PTC component.

Having presented the main features of the PAC-AMODEUS model, we are now able to describe its companion tool: PAC-Expert.

4. Specifications of PAC-Expert

4.1. General presentation

PAC-Expert supports the software designer in devising the Dialogue Controller portion of the user interface. From the external specifications of the system, PAC-Expert produces the representation of a hierarchy of PAC agents which composes the Dialogue Controller for that particular system. This representation is not an executable code nor a formal description but a graphical representation of the architecture. As discussed in Section 2, PAC-Expert is intended to bridge the gap between software design and software implementation. The generated picture can be used as the representation which comes before detailed (formal) internal specifications can be performed. As a result, PAC-Expert is independent from implementation issues.

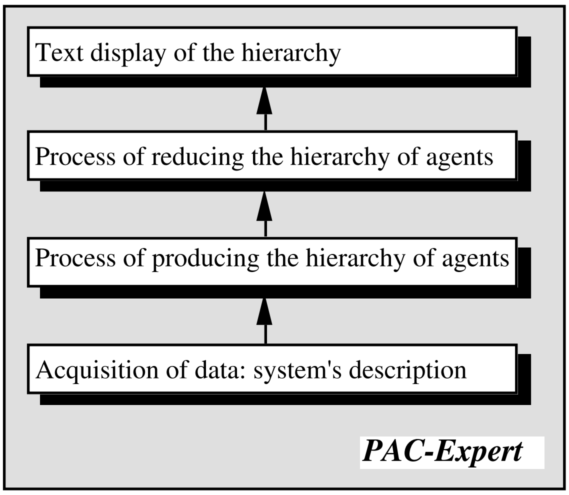

Figure 3: The different computation phases in PAC-Expert.

Figure 3 shows the various computation phases that structure the execution of PAC-Expert. These phases operate in a pipe-line manner: from inputs provided by the software designer, a hierarchy of agents is generated. This hierarchy is displayed in a third phase. Consequently, PAC-Expert consists of three sequential separate processes:

1. Acquisition of the system description,

2. Generation of a Dialogue Controller in terms of PAC agents,

3. Layout of the PAC hierarchy.

Paragraph 4.2 describes the external specifications of PAC-Expert i.e., the behaviour of PAC-Expert as perceived by the software designer. Paragraph 4.3 discusses its internal organization.

4.2. External Specifications

The perceivable behaviour of PAC-Expert is concerned with the nature of input commands as well as with the layout of the results. Note that the HCI design of PAC-Expert was out the scope of RP2. The intention was to test the feasibility of the approach, not to develop a full-fledge tool devoted to teach the PAC-AMODEUS method.

4.2.1. Input to PAC-Expert

Input acquisition is driven by PAC-Expert. Questions, expressed in natural language, are asked to the software designer (let's call him "Soft-D"). In general, Soft-D is requested to answer "yes" or "no" or to provide names to denote interaction techniques such as windows and domain concepts. Although the interaction is system-driven, Soft-D is able to detour the main dialogue through escape commands.

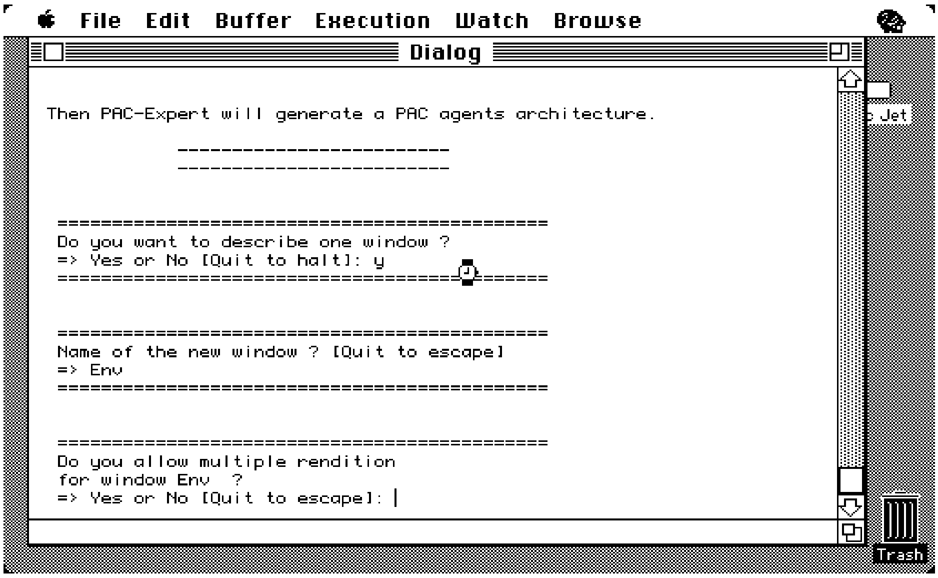

Figure 4a: Snapshot from PAC-Expert. Specifying the existence of a window.

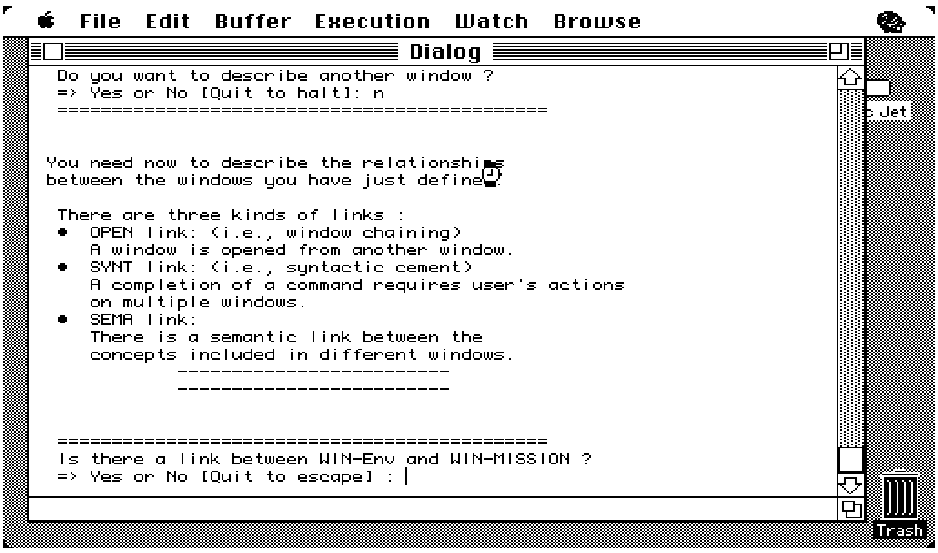

Figure 4b: Snapshot from PAC-Expert. Specifying links between windows.

PAC-Expert asks two types of questions about the system whose architecture is to be inferred:

1. questions about presentation issues, such as the existence of windows for the future system, allow PAC-Expert to identify the locus of interaction between the system and the user;

2. questions about conceptual issues allow PAC-Expert to identify the conceptual objects

that the Functional Core of the system is to export towards the user. Knowledge about conceptual objects is necessary to identify intermediate agents which do not necessarily have all of the facets advocated by the PAC model. One can find in [Nigay 91a] a taxonomy of PAC agents that can be found in a Dialogue Controller.

Figures 4a and 4b give an example of a dialogue between PAC-Expert and Soft-D. This dialogue is structured into three successive steps: window existence, window content, and window links.

1. Window existence allows Soft-D to declare the existence of the set of windows that will be used for the future system. For each window, Soft-D is requested to provide a window name and to specify whether the window can be cloned multiple times. Once the set of windows has been set up, the description of windows content can start.

2. Windows content is specified through a set of questions about the presence/absence of a menu-bar, the presence/absence of complex concepts such as compound concepts whose structure must be made explicit to the user of the future system. Once windows content has been specified, links between windows must be described.

3. Basically, links between windows indicate how windows are opened from other windows. They express the structure of the task space of the user at the interaction level.

4.2.2. Output from PAC-Expert

As discussed above, PAC-Expert generates a hierarchy of PAC agents. This hierarchy is laid out textually or graphically. Figures 5a and 5b show examples of textual layout. The layout can be performed for the complete hierarchy (as in Figure 5a) or at a specific level of abstraction (as in Figure 5b).

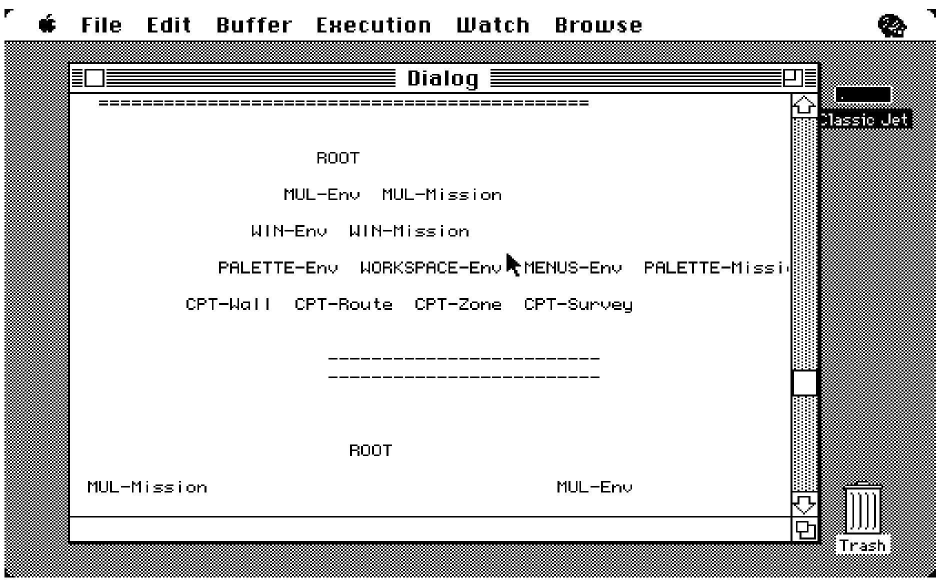

Figure 5a: Snapshot from PAC-Expert. The text display of a complete PAC hierarchy produced for a particular system.

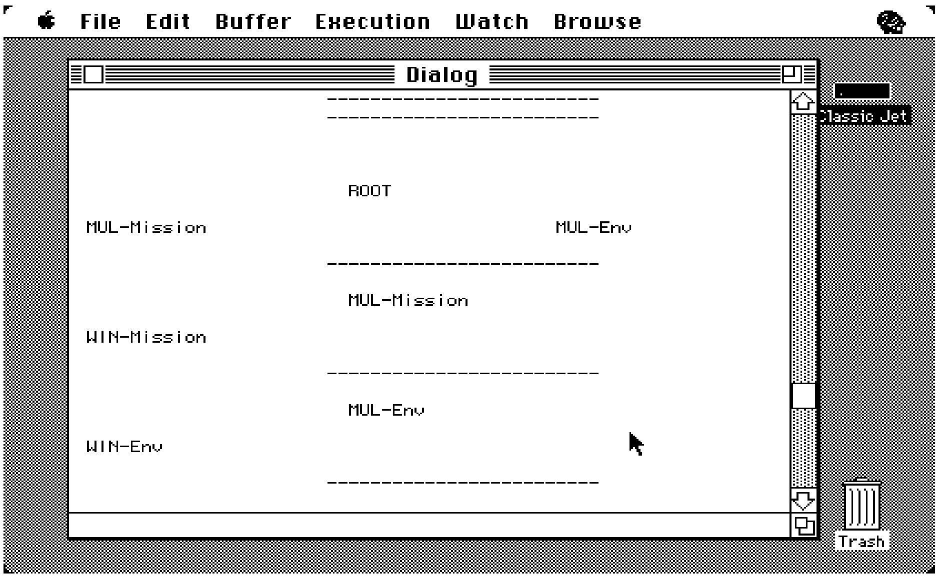

Figure 5b: Snapshot from PAC-Expert. A text display for each level of the PAC hierarchy generated for a particular system.

Textual layout is crude and not editable. A utility tool called PAC-Display[3] is available to display a graphical and editable representation of the generated PAC architecture. As shown in Figure 6, each agent is represented according to the PAC graphical formalism. In addition, an agent representation can be resized, moved, etc. More important, by clicking on an agent representation, a bubble pops up and justifies the role of the agent.

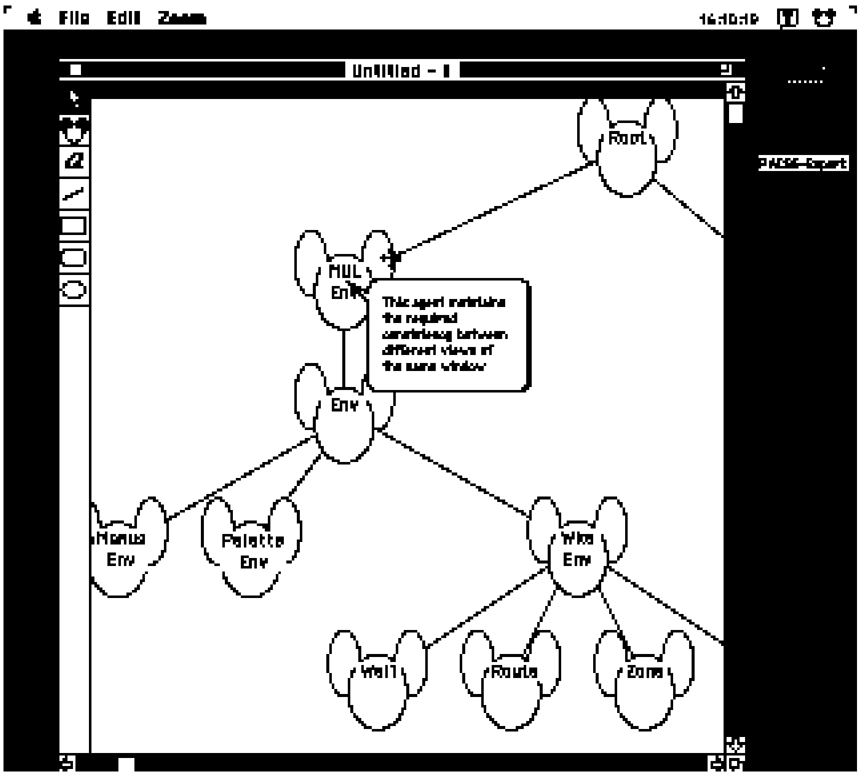

Figure 6: Snapshot from PAC-Display. The graphical representation of a PAC architecture generated by PAC-Expert for a particular system.

3 PAC-Display has been developed by Arno Gourdol using the HIK application framework [Gourdol 91].

Due to implementation restrictions imposed by the development environment, PAC-Display is not integrated into PAC-Expert. PAC-Expert produces an ASCII file which describes the generated PAC architecture. This file is then used as input to PAC-Display. Figure 7 summarizes the link between PAC-Expert and PAC-Display.

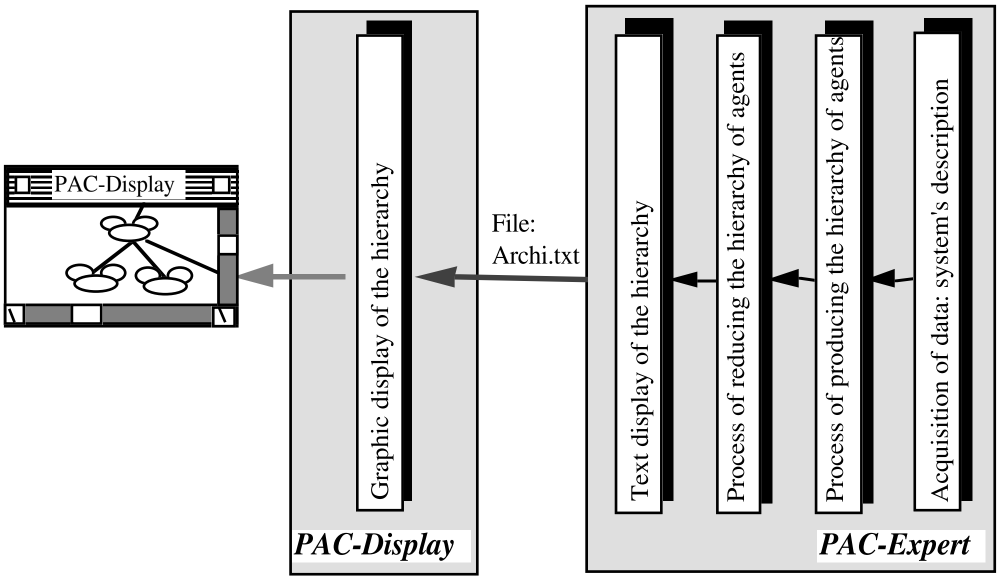

Figure 7: Link between PAC-Expert and the utility program PAC-Display.

4.3. Internal specifications: the rules

PAC-Expert is available on the Macintosh. It is implemented in CLIPS, a language designed for developing Expert Systems [Giarratano 88]. A CLIPS program consists of rules and facts. Given the current facts, the CLIPS inference engine determines which rules to fire.

In this section we discuss the rules programmed in CLIPS that generate PAC architectures for dialogue controllers. These rules model our expertise as software designers in applying the PAC-AMODEUS model. In [Nigay 91b] paragraph 4.2, we provide a set of rules that help in identifying the agents appearing in dialogue controllers.

There is a mapping between the rules and the questions that organize the dialogue with Soft-D. The rules drive the questions. Consequently, there are rules concerned with window existence, window content, and links between windows.

Remark:
We are aware that the following presentation and justifications suffer from a lack of rigor. This is acceptable for the purpose of an internal working paper. In the near future, we will be working on formalizing these definitions.

4.3.1. Window Existence

Generally speaking, a window is a rendering surface for displaying information on the physical screen. In our modelling technique, we make a distinction between main-dialogue-thread windows which display concepts exported from the functional core from convenience windows such as dialogue boxes and forms used in subdialogues to inform the user that an abnormal condition has occurred or to offer him the opportunity to enter the parameters of a command.

Rule 1: Model a “main-dialogue-thread window” as an agent.

The Presentation facet of an agent of type "main-dialogue-thread window" manages the window itself (that is the interaction technique "window" offered by the LLIC component), including the title and the windowing commands such as the resize and move functions. In addition, if the agent is a leaf in the PAC hierarchy, its presentation facet also manages the presentation of the concepts it renders to the user.

If a "main-dialogue-thread window" agent is a leaf in the PAC hierarchy, its Abstraction facet maintains an abstract representation of the concepts it renders or at least the links between these concepts.

Remark:
We observe that, in general, the presentation of a hierarchy of concepts is displayed within the window technique associated with the "main-dialogue-thread window" agent. Although general, this property is not always true. It may be the case that the selection of the representation of a subconcept in the "main-dialogue-thread window" agent opens a new window. In turn, this window, which displays functional core concepts, is modelled as a child "main-dialogue-thread window" agent.

Rule 2: Use an agent to maintain visual consistency between multiple views.

If multiple views about the same concept are allowed and if each view is modelled as an agent, then a Multiple View parent agent is introduced to express the logical link between the children view agents. Any user action with semantic and visual side effect on a view agent is reported by the view agent to its parent, the Multiple View agent, which in turn broadcasts the update to the other siblings.

4.3.2. Window Content

It is often the case that the user interface presents a list of the classes of concepts from which the user can create instances. For example, a drawing editor includes the classes circle, line, rectangle, etc. In general, these classes are gathered into palettes or tear-off menus. Let's call "tool palettes" such presentation techniques.

Rule 3: Model a tool palette as an agent.

The Abstraction facet of a tool palette agent contains the list of the classes of instanciable concepts.

In general, the Presentation facet of a tool palette agent is built from interaction techniques offered by the LLIC component. It is in charge of the local lexical feedback when user actions occur on the physical representation of the concept classes (e.g., reverse video of the selected icon). These actions are then passed to the Control facet.

The Control facet of a tool palette agent maps user actions to the list maintained in the abstraction facet. It transforms these actions into a message whose level of abstraction is enriched (e.g., a mouse click on the “circle” icon is translated into the message "current editing mode is circle").

Tool palette agents provide a good basis for extensibility, reusability and modifiability.

Remark:
We must make a distinction between tool palette agents which represent classes of concepts from main-dialogue-thread-window agents which represent instances of concepts.

Rule 4: Model the menu bar of a window as an agent.

As for tool palette agents, menu bar agents provide a good basis for extensibility, reusability and modifiability.

Remark:
Pull-down menus are embedded in the menu bar agent. However, a tear off menu which has the role of a tool palette, becomes the presentation part of a tool palette agent when it is teared off.

Rule 5: Model the editable workspace of a window as an agent.

A window may contain an area where the user can edit concepts. In this case, this area should be modelled as a "workspace" agent.

A workspace agent is responsible for interpreting (1) the user actions on the background of the window, (2) the user actions on the physical representations of the editable concepts when these concepts are not managed by any special purpose agent, (3) messages from the children agents when those agents represent editable concepts. In addition, a workspace agent may be in charge of maintaining graphical links to express logical relationships between the editable concepts. In summary, a workspace agent has the competence of a manager of a set of concepts.

At the opposite, a non editable area of a set of concepts is not modelled as an agent. It is embedded in the "main-dialogue-thread window" agent which displays these concepts.

Rule 6: Model a complex concept as an agent.

A complex concept may be either a compound concept, or it may have a structured perceivable representation, or it may have multiple renditions.

1. A compound concept is built from sub concepts and this construction must be made perceivable to the user. In general, a hierarchy of agents in the user interface maps the composition of the coumpound concept.

2. When involving a set of rendering items, a simple concept may be modelled as an agent. For example, the concept of wall in the mobile robot scenario, whose presentation is built up from a line, a hot-spot (to select it), and a popup menu (to invoke an operation on the wall), can be modelled as an agent.

3. An elementary concept may be presented in multiple locations. In addition, each presentation may be different. One agent is introduced to maintain the consistency between the multiple presentations.

4.3.3. Window Links

Window links fall into three categories: open links, syntactic links, and semantic links.

Rule 7: If the access to a “main-dialogue-thread window” is allowed from another “main-dialogue-thread window”, a father-child relation is modelled by an agent.

We call “open link”, the relation that describes the possibility for the user to open a “main-dialogue-thread window” from another one. This relationship is controlled by a dedicated agent.

Rule 8: A dedicated agent is introduced to cement actions distributed over multiple agents.

Windows agents are related by a “syntactic link” when a set of user actions distributed over these windows can be synthesized into a higher abstraction. For example, to draw a circle, the user selects the “circle” icon in the tool palette agent, then draws the shape in the workspace agent. These distributed actions are synthetised by a cement agent into a higher abstraction (i.e., the command “create circle”). This agent maintains a syntactic link between its subagents.

Rule 9: An agent is introduced to maintain a semantic relation between concepts included in distinct windows.

This rule is identical to the previous one except that the competence of the Semantic agent is to maintain a semantic relation and not to cement user's actions.

5. Validating PAC-Expert

PAC-Expert has been tested using AMODEUS examplars as well as two full scale systems designed and developed by LGI. For each system, we have analyzed:

1. The architecture produced automatically by PAC-Expert;

2. The architecture that has been effectively designed by hand (and implemented) before PAC-Expert was available.

Interestingly, by comparing hand-designed architectures with automatically-generated architectures, we have been able to detect a defect in one of the manual architectures (i.e., Compo, the authoring system developed for Renault but not considered in AMODEUS studies).

Appendix A provides a detailed description of the experiment with the mobile robot system. Appendix B summarizes the various tests.

Conclusion

PAC-Expert embeds a method for designing software architectures according to the PAC-AMODEUS model. From the description of the external behaviour of an interactive system, PAC-Expert is able to generate a PAC-AMODEUS software architecture for that system. The graphical expression of this architecture constitutes the internal specification of the system before detailed internal specifications of the system can occur. These detailed specifications may be expressed using the agent language developed at York, leading eventually to an executable system or at least allowing the automatic checking of a number of desirable system properties. Thus, there is a natural, waterfall link between the work performed in York and Grenoble.

PAC-Expert will be improved along the following lines: (1) formalization and (2) extension of the PAC-AMODEUS model, (3) links with ICS, (4) user interface.

1. Formalization of PAC-AMODEUS: the agent formalism clarifies the definition of the notion of agent. It does not however solve the problem of identifying the agents involved in an architecture. Although we have made significant progress in this direction, we need to refine our understanding in the agent taxonomy. From a well-informed categorization of software architecture concepts, we can strive for a formal definition of our PAC agents making precise their role and relationships in the various layers of abstraction. Formalizing PAC concepts is a necessity for asseying PAC-AMODEUS as well as for implementing an effective and reliable PAC-Expert.

2. Extension of PAC-AMODEUS: hitherto, we have considered software architectures for the GUI technology only. We need now to investigate the applicability of PAC-AMODEUS to multimodal user interfaces. An early experience with multimodality allowed us to identify new difficulties such as the location for the fusion of modalities, the necessity for an explicit model of the “dialogue” (whatever that means) in order to improve the robustness of the interaction.

3. Links with “ICS-Expert”: as PAC-AMODEUS is embedded in PAC-Expert, so is ICS. Although the finalities are distinct, the ICS and PAC expert systems ask similar questions to the designer. A comparative study that would identify the similarities and differences may have interesting outcomes.

4. The user interface of PAC-Expert: the current version of PAC-Expert does not aim at being an effective tool. Instead, it is a prototype that demonstrates the interest of the approach. As a result, PAC-Expert has not been “user centered designed”. The interaction is system driven and takes the form of a sequences of questions. As an alternative, the input to PAC-Expert could be the external specifications that the designer is supposed to produce before entering the internal specification phase.

Acknowledgements

We wish to thank Arno Gourdol for the development of PAC-Display which, from the output of PAC-Expert, is able to draw editable PAC architectures on the display. PAC-Display has been implemented from a reusable application skeleton, HIK, developed by Arno as well.

References

[Abowd 91]
G. Abowd, .

[ANSI 83]
ANSI/IEEE Standard 729-1983, Software Engineering Standards. IEEE, New york, 1989.

[Bass 91]
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.

[Bass-Coutaz 91]
L. Bass, J. Coutaz, Developing Software for the User Interface. SEI Series in Software engineering, ed. Addison-Wesley, New York.

[Coutaz 87]
J. Coutaz, PAC, an Implemention Model for Dialog Design; Interact'87, Stuttgart, September, 1987, pp. 431-436.

[Coutaz 91]
J. Coutaz, S. Balbo, Applications: A Dimension Space for User Interface Management Systems. In Proceedings of CHI'91 Conference, New Orleans, April 27-May 2, 1991, pp.27-32.

[Giarratano 88]
J. C. Giarratano, CLIPS User's Guide. Version 4.2 of Clips. A Product of the Mission Planning and Analysis Division's Artificial Intelligence Section, NASA Johnson Space Flight Center. Volume 1, 2 and 3, 1988.

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

[Myers 89]
B. A. Myers, Tools for Creating User Interfaces: An Introduction and Survey. IEEE Software 6(1), 1989, pp. 15-23.

[Nigay 91a]
L. Nigay, A Case Study of Software Architecture for Multimodal Interactive System: a voice-enabled graphic notebook, Report, October 1991, section 3, pp. 7-15.

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

[Norman 86]
D. A. Norman, S. W. Draper, eds User centered System Design. Lawrence Erlbaum Associates, Hillsdale, N. J., 1986.

Appendix A, The Mobile Robot Examplar

A.1. Brief presentation of the system

The mobile robot system allows an operator to interact with a mobile robot through a user interface implemented on a SUN workstation. The robot is designed to navigate through a collection of obstacles within a closed environment.

The operator's tasks can be divided into three categories: environment description, mission specification, and mission control.

Environment description consists of providing the robot with an initial description of the structure of the environment (i.e., the geography). This description includes non-movable objects such as walls, and movable objects such as furniture. The robot is also given a description of named places in the environment, as well as routes between these places. The specification of a mission describes the set of surveillance and navigation tasks that the robot will have to accomplish. During mission control, the operator observes and controls the robot's behaviour from the workstation.

In the following paragraphs, we limit the discussion to the <u>environment description portion of the system</u>. We present the PAC hierarchy as designed by hand. We then show the PAC hierarchy produced by PAC-Expert and compare the two solutions.

A.2. Effective Architecture

Figure 8 presents the PAC architecture that has been implemented in C on a SUN workstation. At the bottom of the hierarchy, unitary agents such as wall and route agents model the elementary concepts wall and routes maintained in the Functional Core. These agents are grouped to form an environment managed inside a workspace. This workspace is linked to a specific environment, a Window agent. Since an environment may be opened several times, a Multiple Views agent controls the visual consistency between the multiple views of a given environment.

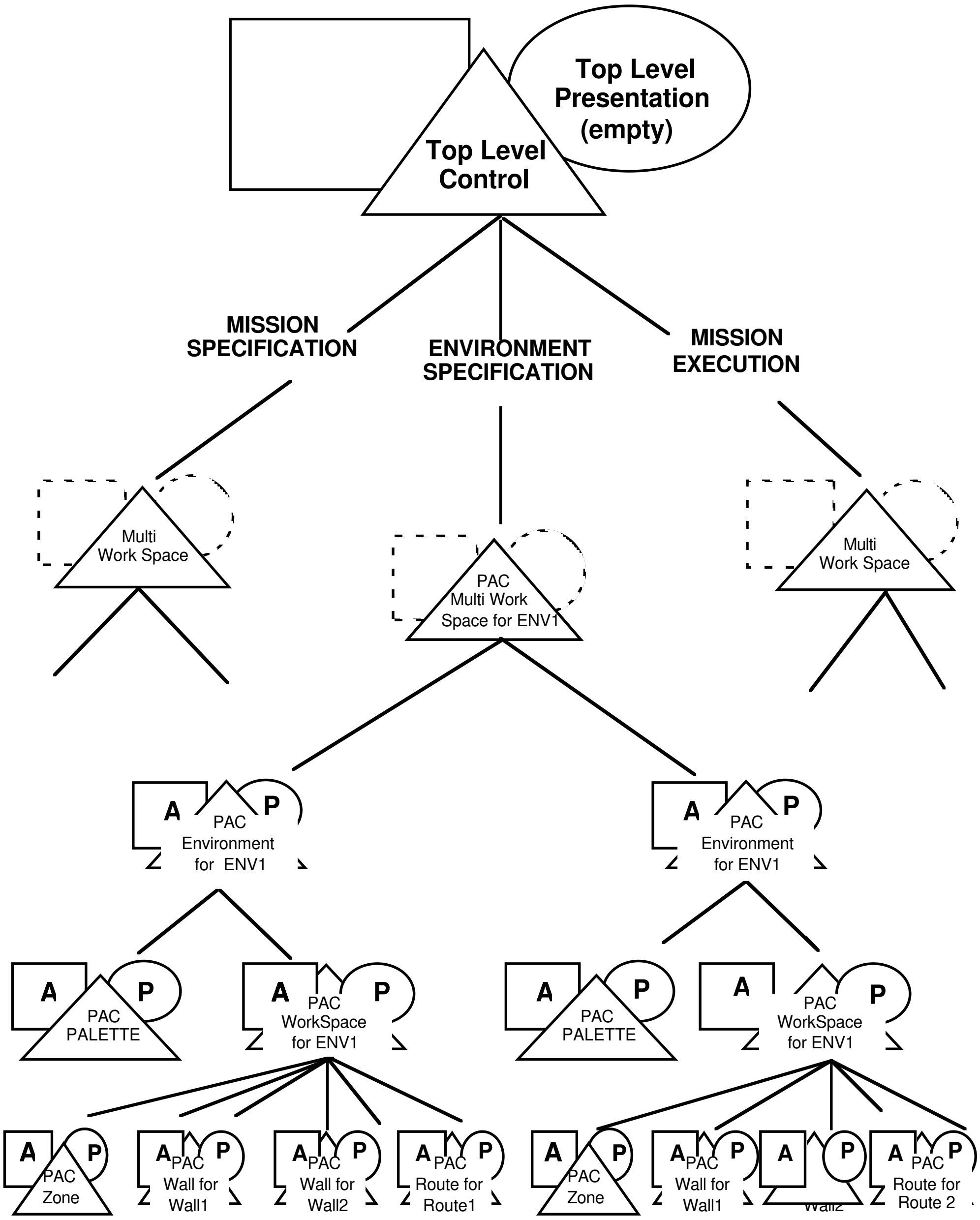

Figure 8 : The PAC architecture implemented for the environment specification part of the mobile robot system.

A.3. The Architecture Produced by PAC-Expert

Figure 9 shows the dialogue between PAC-Expert and the software designer to enter a description of the external behaviour of the system. Figure 10 presents a picture of the resulting architecture. We observe that result is identical to the architecture designed by hand.

```
CLIPS> (run)
*
***********************
********************
*********************************
*********************************
********

          -------------------------
          -------------------------

You are going to describe the User Interface
of your interactive system.
To do so, you will have to answer a number of questions.
These questions are concerned with
- window existence,
- window content,
- window links.
PAC-Expert will generate a PAC agent architecture.

          -------------------------
          -------------------------

 ==================================================
 Do you want to describe one window ?
 => Yes or No [Quit to halt]:  y
 ==================================================


 ==================================================
 Name of the new window ? [Quit to escape]
 =>  Env
 ==================================================


 ==================================================
 Do you allow multiple rendition
 for window Env  ?
 => Yes or No [Quit to escape]: y
 ==================================================


 ==================================================
 Is there a palette of tools inside
 window Env  ?
 => Yes or No [Quit to escape]: y
 ==================================================


 ==================================================
 Does window Env contain at least
 one menubar (i.e., one or several) ?
 => Yes or No [Quit to escape]: y
 ==================================================


 ==================================================
 Does window Env contain
 a zone that can be modified by the end-user ?
 => Yes or No [Quit to escape]: y
 ==================================================



 ==========================================================
```

Which complex concepts are rendered in
window Env ?

A complex concept is:
1) Either a compound concept (i.e., made of sub-concepts).
2) Or a concept rendered in another window.
3) Or a concept which requires a compound presentation
(i.e., made of sub-presentations).

Do you want to describe another complex concept
=> Yes or No [Quit to escape] : **y**
==

==
Name of the new complex concept ? [Quit to escape]
=> **Wall**
==

==
Which complex concepts are rendered in
window Env ?

A complex concept is:
1) Either a compound concept (i.e., made of sub-concepts).
2) Or a concept rendered in another window.
3) Or a concept which requires a compound presentation
(i.e., made of sub-presentations).

Do you want to describe another complex concept
=> Yes or No [Quit to escape] : **y**
==

==
Name of the new complex concept ? [Quit to escape]
=> **Zone**
==

==
Which complex concepts are rendered in
window Env ?

A complex concept is:
1) Either a compound concept (i.e., made of sub-concepts).
2) Or a concept rendered in another window.
3) Or a concept which requires a compound presentation
(i.e., made of sub-presentations).

Do you want to describe another complex concept
=> Yes or No [Quit to escape] : **y**
==

==
Name of the new complex concept ? [Quit to escape]
=> **Route**
==

==
Which complex concepts are rendered in
window Env ?

```
A complex concept is:
1)        Either a compound concept (i.e., made of sub-concepts).
2)        Or a concept rendered in another window.
3)        Or a concept which requires a compound presentation
      (i.e., made of sub-presentations).

Do you want to describe another complex concept
=> Yes or No [Quit to escape] : n
=====================================================

==============================================
Do you want to describe another window ?
=> Yes or No [Quit to halt]: n
==============================================


   ============= PRODUCED HIERARCHY =================

         MUL-Env

        WIN-Env

       PALETTE-Env  WORKSPACE-Env  MENUS-Env

     CPT-Wall  CPT-Route  CPT-Zone

          -------------------------
          -------------------------
           MUL-Env

WIN-Env

          -------------------------
           WIN-Env

WORKSPACE-Env       MENUS-Env       PALETTE-Env

          -------------------------
           WORKSPACE-Env

CPT-Route       CPT-Zone       CPT-Wall

          -------------------------
           MENUS-Env

          -------------------------
           PALETTE-Env

          -------------------------
           CPT-Route

          -------------------------
           CPT-Zone

          -------------------------
           CPT-Wall

          -------------------------
    =====================  END  ======================
    =================== SAVING ======================

265 rules fired
CLIPS> (dribble-off)
```

Figure 9: The dialogue to describe to the environment specification part of the mobile robot system. The end-user's answers are printed in boldfalce. The last part is the generated architecture presented in a textual form. Each agent has a name whose prefix expresses the role or competence of the agent.

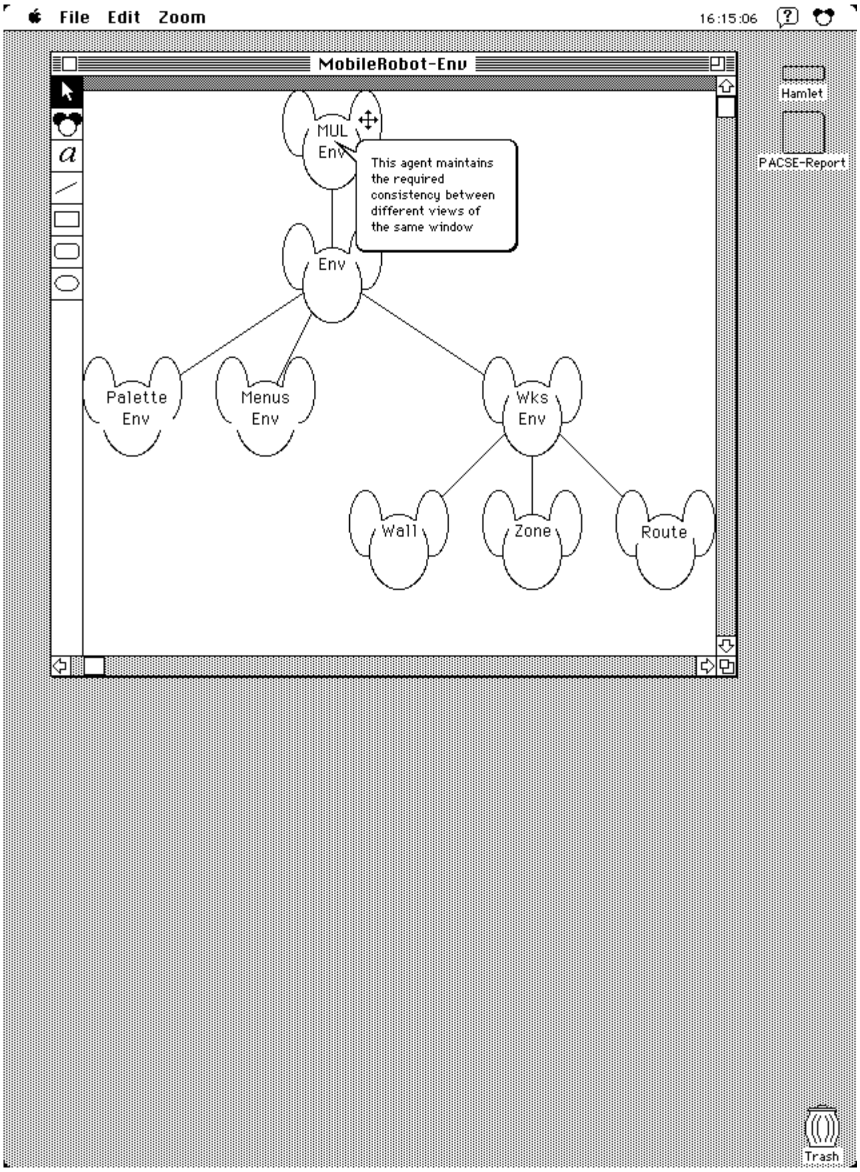

Figure 10: The architecture generated by PAC-Expert and presented graphically by PAC-Display. By moving the mouse on an agent, a bubble shows up to tell the role or competence of the agent.

Appendix B, PAC-Expert experiments

The table of Figure 11 summarizes our experiments with PAC-Expert applied to the AMODEUS examplars as well as to private exemplars and systems. Columns show the rules presented in section 4 whereas lines correspond to the test examplars. A "bomb" at the intersection of a column C and a line L indicates that PAC-Expert has fired Rule C to generate the architecture for examplar L.

Rules / Systems	Window	Multiple View	Palette	Menubar	Work space	Complex Concept	Opening Link	Syntactic Cement	Semantic Link	Reduce	Root Agent
Renault project: Applications generator	10	2	2	4	4	7	10				
Renault project: Authoring langage	6	3	3		3	6	5				1
ThermoPac	2			1	1	4	1				
Conferencing System: What does Undo do ?	3			2	2			1	1		1
Mobile Robot: Environments	2	1	1	1	1	3	1			1	
Mobile Robot: Misssions	5	1	1	1	1	7	4	1	2	1	
The Macintosh Finder	3					4	3				

Figure 11: Validation of PAC-Expert using examplars. The number at the bottom right corner of each square indicates the number of times the rule has been fired.