Architecture Models for Interactive Software

JOELLE COUTAZ

Laboratoire de Génie Informatique (IMAG)
BP 53 X
38041 Grenoble Cedex, France
UUCP: joelle@imag.imag.fr
Tel. (33) 76 51 48 54

Abstract: The definition of an appropriate software architecture is an important problem in user interface design. Without an adequate architectural model, the resulting software may be hard to modify and maintain. Even worse, it may be unable to take the user's characteristics into account.

This paper presents an overview of three significant architectural models and proposes an evaluation. It shows that the monolithic sequential views of interactive software do not match the cognitive processes involved in computer-human interaction. Early experiences with PAC, an object-oriented model, tend to prove that parallel multi-agent views are well-suited to the opportunistic behaviour of the user and to iterative design.

Keywords: User interface design, user interface software architecture, interactive software model, object-oriented model.

1 INTRODUCTION

Computer-human interaction has been described by diverse models from cognitive psychology and computer science.

Models from cognitive psychology such as ACT* [Anderson 83] and the theory of action [Norman 86] offer a way to understand the nature of the cognitive processes involved in computer-human interaction. Other models such as GOMS [Card 83] and the Model Human Processor [Card 83] provide a general behaviorist framework that is useful in predicting human performance. Engineering models such as the Keystroke Level Model [Card 83] and formal grammars [Reisner 82] are useful in making predictions about a particular design or evaluating alternatives between doubtful syntaxes. Although models from cognitive psychology are useful "tools for thought" [Newell 86], they do not lead smoothly to the practical requirements for implementing interactive systems.

Models from computer science are aimed at providing implementers with a framework for building interactive software. Although such a framework is intended for developers, it should be designed for the ultimate benefit of the end-user. Unfortunately, the gap between

user modelling in cognitive psychology, and formalisms and techniques in computer science impinges the transfer of knowledge between the two disciplines. As a consequence, reasonable user interfaces cannot be implemented at the first attempt. Prototyping is a mandatory step.

This paper is concerned with the foundations for prototyping interactive systems. It presents an overview of three significant architectural models: in paragraph 2, the Language Model based on linguistic principles, then, in paragraph 3, the Input/Output Model based on data flow and abstraction techniques, and, in paragraph 4, the Multi-agent Model inspired from stimulus-response systems and close to the notion of object. These models are successively evaluated with special care for two fundamental criteria: the integration of human factors principles and support for easy repair. In paragraph 5, we show how these requirements are satisfied by our own contribution, the PAC Model, an exemplar of Multi-agent Models.

2 THE LANGUAGE MODEL

Language models such as those of Seeheim [Pfaff 85] and of Foley and Van Dam [Foley 84] draw upon the analogy of computer-human interaction as a dialogue between individuals. A dialogue relies on the usage of a common language.

2.1 Principles

According to linguistic models, the language used by a computer and a user is defined by three components: semantics, syntax and lexicon.

- Semantics defines the meaning of sentences. It models the concepts and the knowledge in a particular domain. In the area of computer-human interaction, semantics describes the concepts (i.e. classes of objects) that the user and the system may reference during the interaction. For example, if the task consists in the study of thermodynamics, classes involved in the interaction would include temperature and heat as well as operations that allow the user to identify the relations between these two notions.

- Syntax defines the construction of sentences from a set of predefined syntactic units. A syntactic unit is an element which cannot be decomposed further without loosing its meaning. For example, the graphical symbols which respectively represent a thermometer and a burner do not express the idea of temperature and heat when they are decomposed in terms of graphics primitives. As in linguistics, a syntactically valid sentence is not necessarily semantically meaningful. For example, the sentence built from the movement of a thermometer in the physical space is syntactically correct. However, it has no semantic effect until the thermometer is placed on top of a functioning burner.

- Lexicon defines the construction of syntactic units from a vocabulary. In the area of computer-human interaction, this vocabulary includes any item that can be produced with input and output devices: characters from a keyboard, elementary sounds from a synthesizer, or graphics primitives from a library. For example, graphics primitives allow the implementer to produce a thermometer and a burner on the screen.

Modelling computer-human interaction as a language leads to structuring an interactive software as an organization of three components. As shown in figure 1, an interactive system

is comprised of the Application which implements the semantics, the Dialogue Controller which deals with syntactical issues and the Presentation which is in charge of the lexical aspects of the interaction. The Application Interface defines the view that the Dialogue Controller has of the Application.

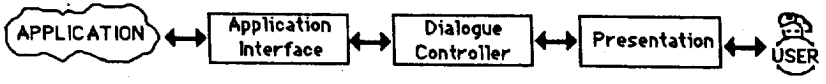

Figure 1: Organizing interactive software according to the Language Model.

2.2 Evaluation

The Language Model presents three advantages: it provides the implementer with a simple framework, it supports the iterative design of user interfaces, and it is general enough to be applied to any type of applications.

- *A simple framework.* The notions of semantics, syntax and lexicon are well understood by computer scientists. They can therefore be easily applied during the design process of a software architecture.

- *Support for iterative design.* The underlying modularity of the model allows for the modification of one component (e.g. the Presentation) without endangering the rest of the system.

- *Generality.* The Language Model does not make hypotheses concerning the nature of the application or the implementation tools. In addition to its use for organizing software, the Language Model has influenced the definition of design methods such as CLG [Moran 81] and that of Mark Green [Green 85]; it has also served as a basis for User Interface Management Systems (UIMS) which automatically generate a user interface from a high level description [Olsen 83, Jacob 84, Wasserman 85, Hayes 85, Schulert 85].

The Language Model presents three major drawbacks: priority is given to the form of the dialogue, the processing of the form is centralized and the interaction language is decomposed into two subsets.

- *Priority to the form of the dialogue.* By nature, a language view of the interaction puts the emphasis on the form, and is totally oblivious to the dynamics. This prejudice is reasonable for a compiler which must process fully specified static input expressions. However, in computer-human interaction, the form of an input expression may evolve in parallel with the production of an output expression; multiple input expressions may even be specified simultaneously. Unlike a compiler which performs a well-defined sequence of processing, the user interface must support parallelism and should allow for partial input expressions to be interleaved with the production of output expressions. Such phenomena are typical, for example, of multi-thread dialogues.

- *Centralization of processing.* Concentrating semantic, syntactic and lexical processing is not compatible with interactivity. Experience shows that some very simple lexical feedback sometimes requires a minimum of semantic knowledge. Direct manipulation interfaces which require immediate and informative feedback, fall in this category. For example, the boundary of a rubber box may depend on data elaborated by the application. In addition, experience demonstrates that the frontier between the Application (the semantics) and the User Interface (the syntax and the lexicon) is not always a sharp

well-defined limit. Instead of partitioning an interactive system into three layers of abstraction, a continuum of abstractions seems to better match the representation process of the human being.

- *Distinction between input and output languages.* Some applications of the Language Model, in particular UIMS's, or some of its interpretations [Foley 84, p. 220], make a distinction between the language used by the end-user to specify input expressions and the language used by the computer system to produce output expressions. The result of this dichotomy is that outputs cannot be reused as input data. In other words, cut-and-paste facilities are totally discarded. A second consequence of this distinction is the functional asymmetry of UIMS's which are only able to support the input language. By doing so, the Application is in charge of producing output expressions and half of the dialogue slips away from the control of the UIMS. This situation may blur the fundamental distinction between domain dependent functions and presentation policies.

3 THE INPUT/OUTPUT MODEL

Whereas the Language Model deals with the form of the interaction, the Input/Output Model (I/O Model) puts the emphasis on data communication and data transformation.

3.1 Principles

Data transfer and data processing may be modelled in two ways: either as layers of abstract machines [Coutaz 85a] (as shown in figure 2) or as a set of cooperating servers [Lantz 86] (as shown in figure 3).

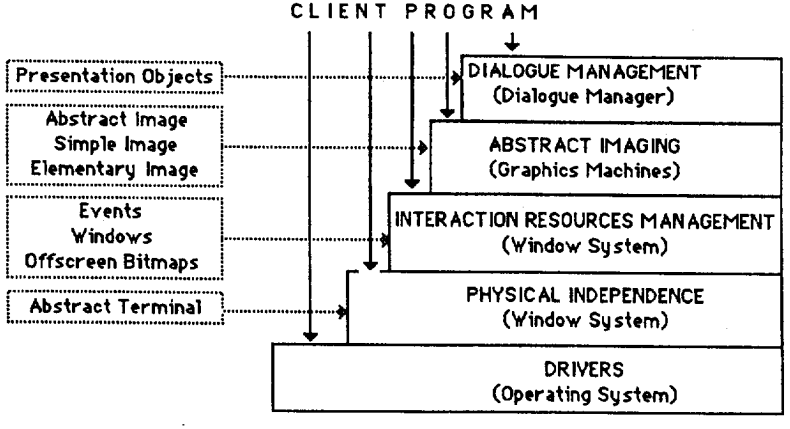

Figure 2: The Input/Output Model: the layered view.

At the bottom of the hierarchy, drivers handle physical devices. They are usually part of the operating system. At the next layer, physical independence comes in the form of an abstract terminal which is usually integrated into a window system.

Higher in the hierarchy, the window system manages the resources involved in the interaction (e.g. screen, keyboard, etc.). Concepts that a window system makes visible to

client programs include windows, offscreen bitmaps, events, etc. The window system behaves as the server of the workstation.

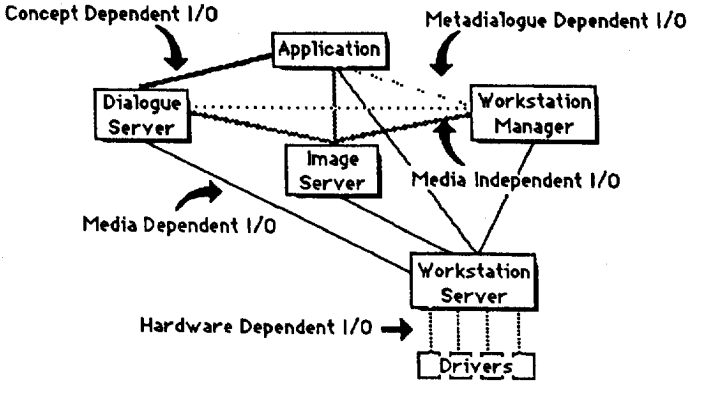

Figure 3: The Input/Output Model: the modular view.

Next in the hierarchy, abstract image machines provide client programs with services for laying out and picking up information from the screen. These services may in turn include several levels of abstraction: elementary images built from graphic primitives such as "draw a line", simple images defined from graphic macros such as GKS segments [ISO 85], Macintosh QuickDraw regions [Rose 86] or PostScript paths [Adobe 85], abstract media independent structured images such as PHIGS structures [ISO 86] or Boxes [Coutaz 85 b].

At the next level, the dialogue manager controls the interaction through a collection of presentation objects. With regard to the application, it defines an abstract dialogue which hides the way application concepts are presented to the user as well as the details of the interaction with the user. To express itself to the user, the dialogue manager can exploit the services provided by an image server or it may directly call the workstation server.

At the top of the hierarchy, the application defines a server for concepts in a particular domain. In a multitasking environment, the workstation manager is a particular application. The domain task of the workstation manager is to launch or to stop the execution of other applications, to allow the user to change his focus of attention between applications, etc. For doing so, it defines a metadialogue. The Unix Shell and desktop managers are examples of workstation managers.

3.2 Evaluation

The I/O Model identifies the software components that make possible the usage of a workstation. It provides the developer with a global view of data transformations from the

user's actions to the application's concepts, and conversely, from the application's data structures to the physical primitives of the hardware.

Although the layered and the modular views of the interaction help to understand the general software organization of a workstation, it does not provide any hint about how to implement a dialogue manager. Whereas window managers and abstract image machines are commonly available, dialogue managers such as application skeletons [Bass 88, Coutaz 87a] or UIMS's, are still a domain of active research.

In summary, the I/O Model distributes data communication and data processing better than the Language Model but the level of abstractions of this distribution is still too coarse to guide the developer in organizing the details of the interaction. The Multi-agent Model presented in the next paragraph provides a solution to this problem.

4 THE MULTI-AGENT MODEL

4.1 Principles

The Multi-agent Model is based on the functioning of stimulus-response systems. Such systems are organized as a collection of agents which react to a given set of external phenomena (i.e. stimuli) and which in turn produce new stimuli. Similarly, the Multi-agent Model structures an interactive system into a collection of specialized agents which produce and react to events. An event acts as a stimulus. It carries information which depends on an event class. It is received by any agent interested in the event class. An agent is a complete information processing system: it includes event receivers and event transmitters, a memory to maintain a state, and a processor which cyclically processes input events, updates its state, and may produce events or change its interest in input event classes.

The Multi-agent Model which stresses parallel processing in a highly modular organization and which uses an event-based communication mechanism shows similarities with the concept of an object: an object class defines a category of agents where class operators and attributes respectively model the instruction set and the state of an agent, and where an event class denotes a method. An object and an agent are both highly specialized processing units, and both "decide" their own state: a state is not manipulated by others but results from processing triggered by others.

4.2 Evaluation

The Multi-agent Model substitutes a highly parallel modular organization to the dichotomous sequential structure of the Language Model. The centralized management of the presentation and of the state of the interaction conveyed by the Language Model is fully distributed among

a collection of cooperating agents. Parallelism and distribution are convenient for supporting the iterative design of user interfaces, for implementing physically distributed applications, and for handling the opportunistic behaviour of the user [Hayes-Roth 79]:

- *Support for iterative design.* An agent defines the unit for modularity. It is thus possible to modify its behaviour without endangering the rest of the system. The centralization of the Language Model makes this adjustment a difficult and dangerous task.

- *Support for distributed applications.* An agent defines the unit of processing. It is thus possible to execute it on a processor different from the processor where it was created. This property is essential for the implementation of collaborative applications such as Colab [Stefik 87].

- *Support for the opportunistic behaviour of the user.* Opportunistic behaviour in problem solving is characterized by a non sequential and a non pure hierarchical approach. When considering computer-human interaction, this behaviour appears in the form of interleaved input expressions: the user initiates some action on the presentation of a concept (called a presentation object), then switches to another object, then may return to the first object and complete the initial action. By doing so, the user simultaneously handles several threads of reasoning. In order to match the user's cognitive process, the user interface must also be multithread.

Multithread dialogues can easily be implemented along the lines of the Multi-agent Model: to each thread of the user's activity, one can associate an agent specialized in this activity. Since a state is locally maintained by the agent, the interaction between the user and the agent can be suspended and pursued at the user's will. When a thread of activity is too complex or too rich to be represented by a single agent, then it is possible to use a collection of cooperating agents. The PAC model [Coutaz 87b] briefly presented in the following paragraph, specifies how to organize this cooperation.

5 THE PAC MODEL

5.1 Principles

PAC structures an interactive system in three parts: the Presentation, the Abstraction and the Control:

- the Presentation defines the Image [Norman 86] of the system, that is the input and output behaviour of the system with regard to the user;

- the Abstraction contains the functional core of the system; it implements the concepts of the task domain;

- the Control maintains the consistency between the two facets, the Presentation and the Abstraction.

This first level of description of an interactive system presents some similarities with the Language Model: the Abstraction corresponds to the Application of the Language Model; the Control includes some of the functionalities of the Application Interface but as shown further, its central location brings extra responsibilities; finally, the PAC Presentation does not match

the rigid framework of a single syntactic Controller and a single lexical Presentation. A PAC Presentation is implemented as a collection of agents specialized in man-machine interaction. For this reason, they are called "interactive objects" or "interactors".

An interactive object is also a PAC structure. It is characterized by

1. an Image which defines a perceivable behaviour (its Presentation, visible to the user),
2. functions and functional attributes (its Abstract side, visible to other software components), and
3. the maintainance of the links between the abstract and concrete sides (the Control).

The example of figure 4 illustrates the structure of a PAC burner:

1. the Presentation includes a shape (that of a burner) with a switch that the user can press on and off with the mouse. When on, the switch turns to black. It shows effluvia moving from the top when the burner is hot enough, and it follows the mouse movements when the user moves the burner around.
2. the Abstraction is comprised of an integer value (hot) and two booleans (isOn, and isSomethingOn). The burner has no knowledge of the interpretation that other software components may make of these abstractions. Similarly, these abstractions are unaware of the way they are presented to the user.
3. the Control maintains the consistency between the Presentation and the Abstraction. It is the decision maker. In particular, it chooses the translation technique to be used between the abstract and the concrete views of an object. In the example, when the user presses the switch, the mouse event processed by the Presentation provokes the switch to change color and the Control is warned of the modification. In turn, the Control translates the modification into the negation of the isOn abstract item and, if related to other interactive objects, the Control may warn them of the status change.

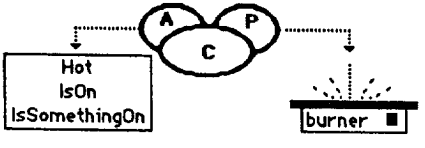

Figure 4: Modelling a burner as a PAC interactive object.

5.2 The recursivity of the PAC Model

A PAC interactive object may be elementary (as the burner) or composite. A composite object is a structured agent whose behaviour depends both on itself and on its composing objects. Its Presentation and its Abstraction inherit from the Presentation and the Abstraction of the components but include their own characteristics. Its Control maintains the consistency between the abstract and the presentation sides. In addition, the Control manages the collaboration or the dependency between the components.

By applying techniques of refinement or composition, an interactive system may be organized as a hierarchy of PAC entities. As shown in figure 5, an interactive system is a PAC composite object. At the top of the hierarchy, the Abstraction corresponds to the Application, the Presentation organizes the Image of the system, and the Control acts as a middleman between the two views.

From the point of view of the top level Control, the Application is a producer/consumer of values which model the concepts of the task domain. These abstract representations are convenient for the Application but may not help the user in accomplishing his task. Raw abstract data are thus replaced by intermediaries which are adapted to the user's task: interactive objects. One of the tasks of the top level Control is to link application concepts with abstract items of interactive objects. As a result, any modification performed by the Application to a concept is automatically translated by the top level Control into the formalism understood by the abstractions of the related interactive objects. In turn, the Controls of these objects propagate the modification to their own Presentation part as well as to their dependent interactive objects. This process ends when elementary objects are encountered.

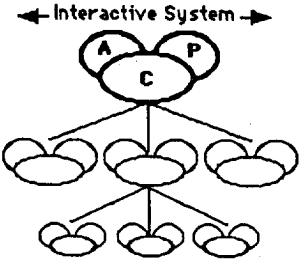

Figure 5: An interactive system is a PAC composite object.

Conversely, user's actions are first interpreted by the Presentation of elementary objects, translated through the elementary Control into the elementary abstract formalism, then propagated upwards until some Application concept is reached. The following example illustrates the functioning of an interactive system structured as a hierarchy of PAC entities.

5.3 Thermo, an Example of a PAC Interactive System

The goal of the Thermo system is to show the relationship between the notions of temperature, heat, and volume of water. For doing so, a real world metaphor has been applied [Hutchins 86]: a glass with a thermometer indicates the current value of the temperature, a burner is used as a source of heat, and water may be poured into the glass by pushing the "water" button of a dispenser. As in the real world, the water falls from the dispenser, effluvia may be observed on the surface of the burner when it is hot, bubbles appear in the glass when the water is boiling, and the glass and the burner may be moved

around with the mouse. As an extension of the real world, a plot is used to show the user the evolution of the temperature. Figure 6 shows the Image of such a system.

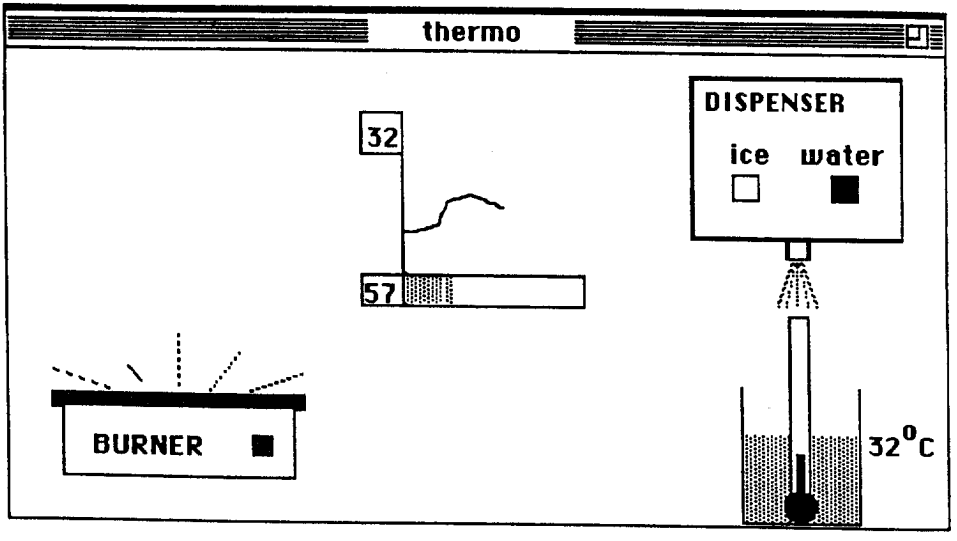

Figure 6: The Image of the Thermo system.

Figure 7 describes the PAC architecture of Thermo. At the top of the hierarchy, the Abstraction (i.e. the Application) implements a mathematical model of a set of thermodynamics laws. The Presentation has no special feature. It can be considered as non-existent. The top Control fulfills its usual role; Its function will be developed in 5.4.3. The component objects can be classified in two categories: the real world metaphor objects (specific to the domain) and the general purpose objects (such as menus and dialogue boxes). The structure of the burner has been described in 5.1. A similar structure is used for the dispenser and the glass with its thermometer. Each structure tends to imitate the behaviour of an object from the real world. Any behaviour is nevertheless dependent on constraints enforced by the environment.

In the case of Thermo, the environment is the cohabitation of a burner, a dispenser, and a thermometer in a glass. In order for this cohabitation to satisfy the physical laws of the environment (e.g. the gravitation law), it needs to be modelled by an agent. The role of the agent ENV in figure 7 is to enforce environmental laws on the individuals, the dispenser, the burner and the glass. It has no Presentation per se but its Abstraction implements some gravitational law and its Control is the arbitrator between the real world objects. For example, when the user moves the glass with the mouse, the PAC glass notifies ENV of its new location and ENV checks whether the glass is in the domain of attraction of another object (e.g. the burner). If so, ENV tells the glass to go to the attraction point (e.g. the top location

of the burner).

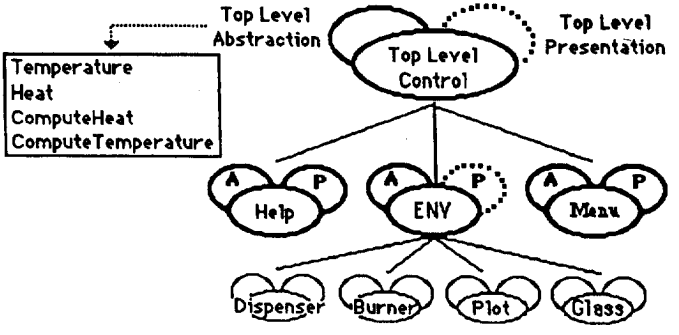

Figure 7: the PAC architecture of the Thermo system.

The Thermo system is a toy example. However, it demonstrates the interest of the PAC Model presented in the following paragraph.

5.4 The Interest of the PAC Model

In addition to the advantages of the Multi-agent Model, PAC presents three interests: it provides the software designer with a recursive framework, it distributes the semantics and the syntax at various levels of abstraction, and it explicitly introduces the notion of Control.

5.4.1 A recursive framework

PAC is in the direct line of modular techniques. However, it requires the software designer to think in terms of active specialized agents rather than in terms of passive modules. In addition, each agent must be seen as the association of two formalisms (that of the Abstraction and that of the Presentation) which cohabit through a common translator (the Control): one formalism is aimed at the internal software requirements whereas the other is adapted to the user. These two formalisms have no knowledge of each other. They must always refer to the Control for expressing requests or status changes. This systematic behaviour defines a framework for the construction of user interfaces that is applicable in a consistent way at any level of abstraction. As a result, semantic, syntactic and lexical processing are distributed across a hierarchy of entities rather than forming hermetic layers that do not match the cognitive organization of human knowledge [Anderson 83].

5.4.2 Distribution of the semantics and of the syntax at various levels of abstraction

Distributing semantics and syntax allows for the exchange of information at the desired level of abstraction. It also suppresses the strict boundary between the Application and the User Interface.

Data exchanged at the desired level of abstraction. The top level Abstraction defines the semantics of the application domain. It describes a competence which is independent of any syntactic consideration. As a result, the top level Abstraction must communicate with the top level Control in its own terms. For example, if it manipulates integers, it should be able to receive and send integer values. Similarly, semantic errors should always be expressed symbolically. Errors are concepts! They should never be implemented as static character strings in the top level Abstraction. If so, the expression of error messages cannot be modified easily. Instead, the final form of semantic errors should be elaborated by interactive objects which, given the symbolic form and the type of user, will determine the adequate formulation.

Figure 8 shows how links can be maintained in the top level Control in order for the top level Abstraction to communicate in its own terms. The abstract concepts, Temperature and Heat, are respectively linked to the abstract item Temp of an interactive object of class Glass and to the abstract item Hot of an interactive object of class Burner. Similarly, isOn and isSomethingOn are respectively linked to the functions ComputeHeat and ComputeTemperature. Any modification to these boolean values triggers the associated functions. Conversely, any modification to the temperature or heat concepts is translated into a modification to the associated abstract items.

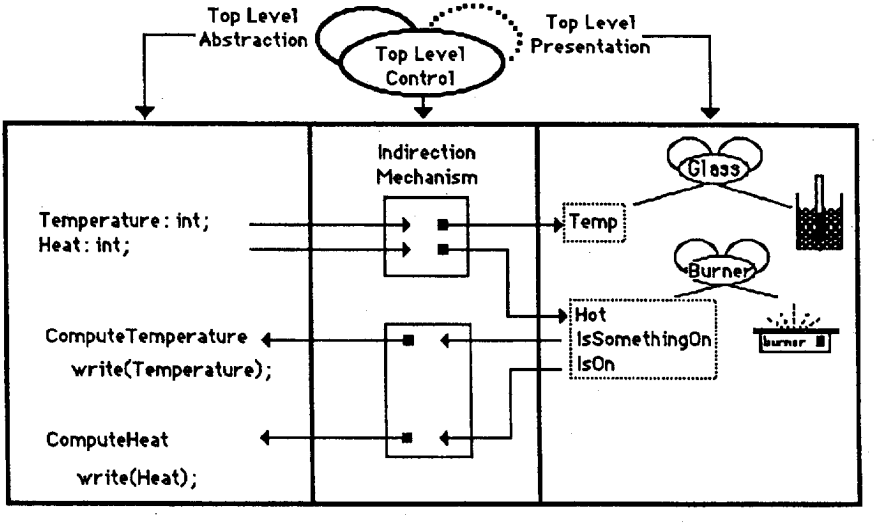

Figure 8: The top level Abstraction communicates with the top level Control in its own terms.

Suppression of the boundary problem between the Application and the User Interface . In some domains, the boundary between the functional core and the User Interface is difficult to identify. PAC which encourages the distribution of knowledge, provides a solution to this

problem. The PAC distribution which is accomplished in a systematic and structured way does not lead to some disorganization which would make iterative adjustment impossible. Instead, it allows for semantics repair, for semantics delegation, and for the cohabitation of several domains. To illustrate these issues, we will cite examples from IRENE.

IRENE is an expert system for configuring Ethernet local area networks [Coutaz 88]. It has been built with ELOISE, a kernel for implementing expert systems whose architecture has also been designed according to the PAC model. IRENE behaves like a collaborator: at the user's will, it automatically generates a configuration or checks the user's proposal.

- *Semantics repair.* The usage of a system may reveal the absence of a concept which would be helpful in accomplishing a given task. For example, a user of an early version of IRENE complained that he had no general indication of the semantic validity of his work. Although messages produced by IRENE are necessary for accomplishing the task, they do not provide the user with a synthesized view of the current situation. In the design of the top level Abstraction, we forgot to define a function that would count the current number of problems. The PAC architecture of IRENE let us insert a new PAC interactive objects, the Status-Man, whose Abstraction consists in counting the symbolic messages issued by the top level Abstraction (i.e. the expertise) and whose Presentation is illustrated in figure 9. This semantic repair has been performed in one day without modifying the code of the expertise and without endangering the overall organization of the system.

- *Semantics delegation.* Semantics delegation consists in downloading knowledge from a high level Abstraction into some lower level interactive object. This facility may reduce data transfer between levels of abstraction and thus, improve performance. One useful application of semantics delegation is the transfer of abstract knowledge from a data base into forms [Collet 87]. Forms are interactive objects whose fields sometimes require an immediate checking from the data base. The PAC Model authorizes the Abstraction of a form to perform such a verification in a natural way. In the system IRENE, all of the forms exploit the semantics delegation technique.

- *Cohabitation of several domains.* Semantics distribution permits interactive objects to express not only the semantics of the task domain but also that of a totally different domain. This situation is frequent in the implementation of real world metaphors. The Thermo system illustrates such a situation where the domain of thermodynamics (implemented in the top level Abstraction) cohabits with that of the gravitation (implemented in the Abstraction of ENV).

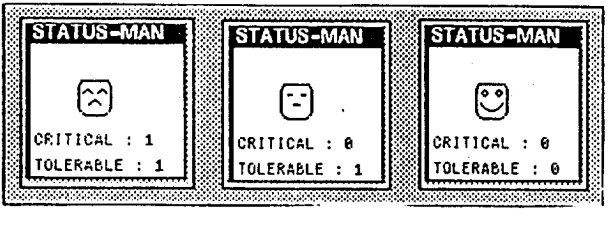

Figure 9: the Presentation of the Status-Man in the system IRENE.

5.4.3 An explicit notion of Control

In the next paragraph, we will briefly present other models which, as PAC, are based on the

notion of object or agent. All of them stress the distinction between functional issues and presentation policies but none of them indicates how to accomplish the separation. The notion of Control in the PAC Model serves as an explicit bridge between two worlds whose role and formalisms are distinct. It is used as a translator/linker and as an arbitrator.

A Control as a translator and a linker. An Abstraction and a Presentation are two perspectives of the same "thing". As mentioned earlier, these perspectives are expressed in a formalism chosen for the purpose they serve; They never reference each other explicitly. This feature guarantees the independence of the implementation of the two views. As a consequence, one can modify the behaviour of a perspective or substitute a perspective without endangering the second one. This type of adjustment is made feasible because of the Control which includes translation facilities between two formalisms and which solves indirect references. Figure 8 illustrates one possible technique with the use of correspondence tables. In IRENE, the top level Control maintains dynamic links since new interactive objects and abstract entities are dynamically created. It also switches from the frame-based representation used by the top level Abstraction and implemented in an OPS-5 [Brownston 85] like language, to the object-oriented PAC interactors implemented in LOOPS [Bobrow 83].

In order to reuse objects, indirection may also be applied between PAC objects. For example, in Thermo, the burner, the glass, the plot and the dispenser never reference each other. The world they represent is modelled by an agent, ENV, which determines their behaviour with regard to each other.

In addition to facilitate software reusability, the mechanism of indirection offers a convenient way for providing the user with multiple representations of the same concept. For example, in Thermo, the notion of temperature is simultaneously represented by two interactive objects: the plot and thermometer of the glass. For doing so, the variable Temperature from the top level Abstraction is linked to the abstract item Temp of a Glass and to an abstract item of a Plot. As a result, any modification of Temperature is automatically propagated by the top level Control to the associated interactive objects.

A Control as an arbitrator. Since a Control is the central processor of a PAC entity, it is a good candidate for arbitrating conflicts. Conflicts may happen between the Abstraction and the Presentation of a PAC agent but also between the components of a composite PAC agent. Conflicts that frequently arise in multithread dialogues and direct manipulation user interfaces concern resource allocation and graphical constraints.

Resources involved in "modern" user interfaces include the screen and the central processor unit. In such user interfaces, several objects may need to express themselves simultaneously. For doing so, they must access the CPU as well as the screen. For example, in Thermo, effluvia must move away from the top of the burner while water falls from the dispenser and fills up the glass. In order to animate the scene, each interactive object must be guaranteed to get the CPU. In addition, since interactive objects may overlap, they need to access the screen in the right order. Such resource allocation can be performed by a Control. In Thermo, CPU allocation is part of the top level Control, and screen refreshing is controlled by ENV. This example shows that each Control has some local effects which in turn can be supervised by higher Controls in the hierarchy. This observation leads to the general notion of constraint checking which, according to the PAC Model, can be distributed at various levels of abstraction.

5.5 Related Models

A comparative analysis between Multi-agent Models such as MVC [Goldberg 84], Interviews [Linton 86], GWUIMS [Sibert 86], and PAC, can be found in [Coutaz 87b, Coutaz 88]. To summarize, MVC distinguishes the output presentation from inputs whereas PAC concentrates them. Decoupling has the advantage of flexibility whereas concentrating may result in better performance when inputs and outputs are tightly coupled. MVC has no explicit notion of arbitrator and linker as PAC does. Interviews stresses the distinction between the abstractions (the subjects) and the presentation policies (the views) but does not use any notion of Control. GWUIMS introduces bridge objects between Abstract objects and Representation objects but, as Interviews and to the opposite of PAC, is not recursively applicable.

6 CONCLUSION

The I/O Model represents the interaction as a sequence of data exchanges and data processing performed at various levels of abstractions (i.e. from the physical actions of the user to the concepts of the Application). It provides the software designer with a synthetic view of the software components involved in man-machine interaction but it does not give any hint about how to organize the dialogue manager.

The Language Model fills the gap left by the I/O Model by concentrating on the representation of the dialogue. It is inspired from the linguistic models which structure a language into semantics, syntax and lexicon. Although the Language Model provides the software designer with a simple framework and leads the way to iterative design, its centralized organization enforces the sequential processing of input and output expressions. It is thus uncompatible with the opportunistic behaviour of the user.

Multi-Agent models stress modularity, distribution, and concurrency in the form of cooperating specialized actors. By doing so, they make possible multithread dialogues, facilitate iterative adjustments, and tend to match the cognitive processes involved in computer-human interaction. Experience shows that object-oriented techniques are well-suited to the implementation of such models.

Acknowledgment. This work sets the foundation for the user interface of project GUIDE, an object oriented distributed system, designed at LGI/IMAG. Systems ELOISE and IRENE were made possible by a "Program Grant" of Xerox France. They have been implemented by a team from DESS-GI (University of Grenoble) in collaboration with James Crowley of LIFIA/IMAG.

REFERENCES

[Adobe 85]
Adobe: PostScript Language Reference Manual, Addison Wesley, Reading, Mass., 1985.

[Anderson 83]
J.R. Anderson: The Architecture of Cognition; Harvard University Press, Cambridge, Massachusetts, 1983.

[Bass 88]
L. 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.

[Bobrow 83]
D.G. Bobrow, M. Stefik: The Loops Manual; Tech. report KB-VLSI-81-13, Knowledge Systems Area, Xerox, Palo Alto Research Center, 1981.

[Brownston 85]
L. Brownston, R. Farrell, E. Kant, N. Martin: Programming Expert Systems in OPS5, An Introduction to Rule-Based Programming; Addison Wesley Publ., 1985.

[Card 83]
S. Card, T. Moran, A. Newell: The Psychology of Human-Computer Interaction, ISBN 0-89859-243-7, Lawrence Erlbaum Associates, Publish., 1983.

[Collet 87]
C. Collet: Les Formulaires Complexes dans les Bases de Données Multimédia; Thèse de Doctorat de l'Université Scientifique Technonologique et Médicale de Grenoble, novembre 1987.

[Coutaz 85b]
J. Coutaz: A Layout Abstraction for User System Design; ACM SIGCHI, January 1985, pp. 18-24. Paru également dans Carnegie-Mellon University Technical Report, CMU-CS-84-167, December, 1984.

[Coutaz 85a]
J. Coutaz: Abstractions for User Interface Design; IEEE Computer, 18(9), September, 1985, pp. 21-34.

[Coutaz 87a]
J. Coutaz: The Construction of User Interface and the Object Oriented Paradigm; ECOOP'87, The European Conference on Object Oriented Programming, Paris, June, 1987, pp. 135-144

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

[Coutaz 88]
J. Coutaz: Interface Homme-Ordinateur: Conception et Réalisation; thèse de doctorat d'état de l'Université Joseph Fourier, BP 53X, 38041 Grenoble Cedex, France, december, 1988.

[Foley 84]
J. D. Foley, A. Van Dam: Fundamentals of Interactive Computer Graphics; Addison Wesley, Publ., 1984.

[Goldberg 84]
A. Goldberg: *Smalltalk-80: The Interactive Programming Environment; Addison-Wesley Publ., 1984.*

[Green 85]
M.W. Green: The Design of Graphical Interfaces; PhD Thesis, Tech. Report CSRI-170, Computer Systems Research Institute, University of Toronto, Canada, M5S 1A1, April, 1985.

[Hayes 85]
P.J. Hayes, P. Szekely, R. Lerner: Design Alternatives for User Interface Management Systems Based on Experience with Cousin; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April, 1985, pp. 169-175.

[Hayes-Roth 79]
B. Hayes Roth, F. Hayes-Roth: A Cognitive Model for Planning; Cognitive Science, 3, 1979, pp. 275-310.

[Hutchins 86]
E. L. Hutchins, J. D. Hollan, D. A. Norman: Direct Manipulation Interfaces; User Centered System Design; Lawrence Erlbaum Associates, Publishers, 1986, pp. 87-124.

[ISO 85]
International Organization for Standardization: Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description; ISO IS 7942, July, 1985.

[ISO 86]
International Organization for Standardization: Information processing systems - Computer graphics - Programmer's Hierarchical Interface to Graphics (PHIGS) functional description; ISO DP 9592, October, 1986.

[Jacob 84]
R.J.K. Jacob: An Executable Specification Technique for Describing Human-Computer Interaction; Advances in Human Computer Interaction, H.R. Hartson, ed. Alex Publishing Co., 1984.

[Lantz 86]
K.A. Lantz: On User Interface Reference Models; ACM SIGCHI Bulletin, 18(2), pp. 36-44.

[Linton 86]
M. Linton, C. Dunwoody: Partitioning User Interfaces with Interactive Views; Computer Systems Laboratory, Stanford University, Stanford, CA 94305-2192, communication privée, April, 1986.

[Moran 81]
T. Moran: The Command Language Grammar: a representation for the user interface of interactive computer systems; International Journal of Man-Machine Studies, 15, 1981, pp. 3-50.

[Newell 86]
A. Newell, S. Card: Straightening Out Softening UP: Response to Carroll and Campell; Human Computer Interaction, 2(3), 1986, pp. 251-267.

[Norman 86]
D. A. Norman, S. W. Draper: User Centered System Design; Lawrence Erlbaum Ass. Publ., 1986.

[Olsen 83]
D.R Olsen, E.P Dempsey: Syngraph: A Graphical User Interface Generator; Computer Graphics, July 1983, pp. 43-50.

[Pfaff 85]
User Interface Management Systems; G. E. Pfaff ed., Eurographics Seminars, Springer-Verlag, 1985.

[Reisner 82]
P. Reisner: Further developments toward using formal grammar as a design tool; Proceedings of Human Factors in Computer Systems, Gaithersburg, Maryland, March, 1982, 304-321.

[Rose 86]
C. Rose et al.: Inside Macintosh; Addison Wesley Publ., 1986.

[Schulert 85]
A.J. Schulert, G.T. Rogers, J.A. Hamilton: ADM - A Dialog Manager; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April 1985, pp. 177-183.

[Sibert 86]
J.L. Sibert, W.D. Hurley, T.W. Bleser: An Object Oriented User Interface Management System; SIGGRAPH'86, 20(4), 1986, pp. 259-268.

[Stefik 87]
M. Stefik, G. Foster, D.G. Bobrow, K. Hahn, S. Lanning, L. Suchman: Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings; *Communications of the ACM, 30(1), January, 1987, pp. 32-47.*

[Wasserman 85] A. Wasserman: Extending State Transition Diagrams for the Specification of Human-Computer Interaction; IEEE Transactions on Software Engineering, 11(8), August, 1985.