Chapter 3: Software Architecture Modelling: Bridging Two Worlds Using Ergonomics and Software Properties

L. Nigay, J. Coutaz

1. Introduction

The process of designing and constructing user interfaces is critical for 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. In addition, the increasing complexity and size of software systems require sound engineering principles and frameworks to formally structure the design process into multiple but consistent perspectives.

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 is an hybrid multi-agent software architecture model that represents the organization of the components of an interactive software. As any architectural framework, it consists of the description of an organisation of computational elements and their interactions [Shaw 95].

Software tools for the construction of user interfaces will not eliminate architectural issues as long as the construction of user interfaces requires programming. Developers and maintainers of interactive systems need to rely on models:
- for identifying software components,
- for organising their interconnections,
- for reasoning about components and interconnections,
- for verifying ergonomic and software properties,
- for modifying and maintaining them in a productive way.

Software architecture may also be used to communicate a design solution to another development team such as the programmers. In this case, the description must be unambiguous. A model may reduce misinterpretation risks.

In this chapter, we present an agent-based architectural model PAC-Amodeus, for the purpose of assessing software designs. In order to clarify the role of a software architecture model, we first identify its use in a design and development process. In Section 3 we present our PAC-Amodeus model: the principles of the PAC-Amodeus model and guidelines for applying it. In Section 4, we show how PAC-Amodeus can be used in practice: we present a PAC-Amodeus software design solution of a WWW browser and its assessment in terms of ergonomic and software properties.

2. Software architecture and life-cycle: point of contact of two worlds

As shown in Figure 1, software engineering structures design and implementation into 6 phases: requirements definition, specification, implementation, testing, installation and maintenance [ANSI 83]. A more complete description of each phase can be found in [Bass-Coutaz 91, chapter 1]. 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. Ideally requirements are defined in cooperation with the end-users.
2. Specification consists of high level design (i.e., external specifications) and internal design (i.e., internal specifications). High level design is concerned with the external behavior 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. Internal design 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.
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.

Software architecture is concerned with the Specification phase of the software engineering life cycle. As mentionned in the introduction, its role is to help the designer to identify the software components that implement the user interface portion of an interactive system. Thus, within the specification phase, a software architecture model deals with internal design.

Software architecture modelling starts with the design solutions selected in the design space and leads to the implementation of these solutions. Like the other modelling approaches, it enriches the design space with specific properties. At this level, salient properties might include software properties such as efficiency and reusability as well as ergonomic properties such as multithread dialogue and observability. As shown in Figure 2, software architecture modelling is a design activity at the turning point between two worlds: the user interface design field and the programming field. Because of its interlinking location, software architecture must take into account properties of the two worlds: ergonomic properties of the designed user interface and properties of the future implemented software. Consequently, it requires that the software designer establishes sound trade-offs between conflicting properties from multiple requirements: user-centered design requirements, software engineering requirements and ease of implementation (implementation cost and time constraints).

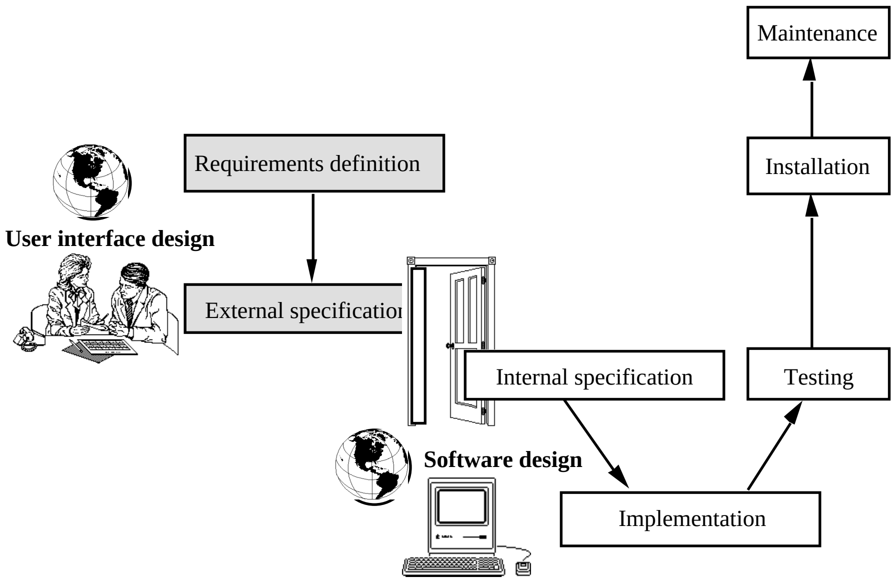


Figure 1 Software engineering life-cycle.

In [Gram-Cockton 96], we denote the properties derived from the user and the system perspectives as external and internal properties respectively:

- External properties: "The usability of an interactive system is linked to the quality of the dialog, and quality shall here be expressed through the number of measurable properties of the dialog." [Gram-Cockton 96, chapter 2] "...user-centered properties of interactive systems ... promote a high quality from the perspective of the users." [Gram-Cockton 96, chapter 2]

- Internal properties: "Internal properties are quality attributes of a system as seen from the developer's perspective, just as the external properties are system quality attributes as seen from the user's point of view." "Internal properties require a complete life cycle view. It is important to recognize that these properties are relevant from the conception of a system, beyond construction to modification and maintenance until its final demise." [Gram-Cockton 96, chapter 3]

A given software architecture model does not verify all of the external and internal properties. Each software architecture model is suitable for a sub-set of properties. By “suitable” we mean that the model helps to either verify or assess a property. In the following section, we present PAC-Amodeus using the notion of internal and external properties as an analytic tool.

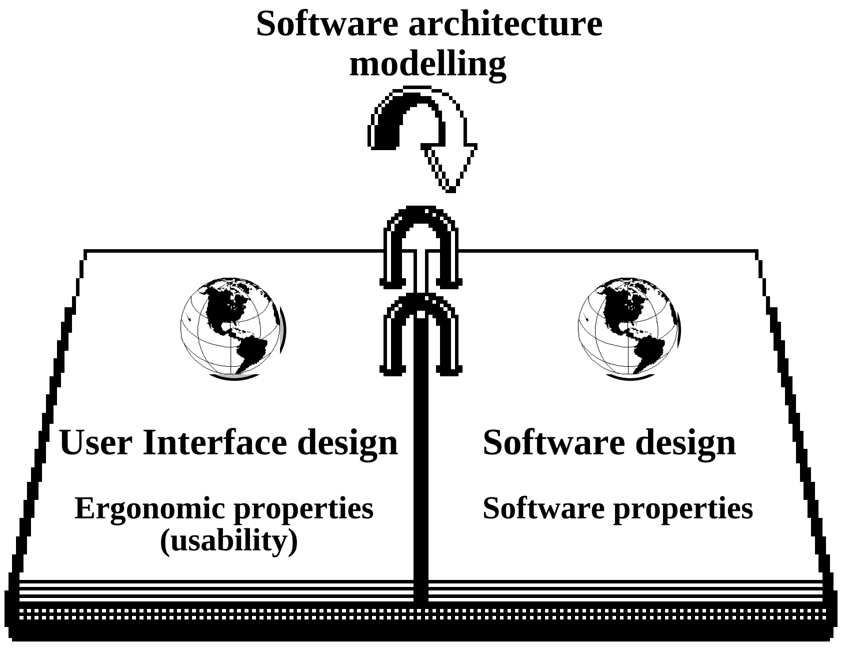

Software architecture modelling
User Interface design
Ergonomic properties (usability)
Software design
Software properties

Figure 2 Software architecture at the turning point of two worlds.

3. Model and Method with PAC-Amodeus

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 support the external specifications of that system. 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. A method is a set of directions (or procedures) for using a particular model that represents the artefact being designed and built.

In this section, we first describe the model in terms of its the software components and their interactions. We then discuss the external and internal properties that the model can support. By doing, so we emphasize the links between the external and internal properties. We finally conclude the section by providing guidelines for applying the model.

3.1. PAC-Amodeus, a software architecture model for interactive systems

As shown in Figure 3, 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. 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. In constrast to 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 [Coutaz 87]. A more detailed description of PAC-Amodeus can be found in [Nigay 91] [Nigay 94]. The following paragraphs provide a synthesized definition of the various components illustrated in Figure 3.

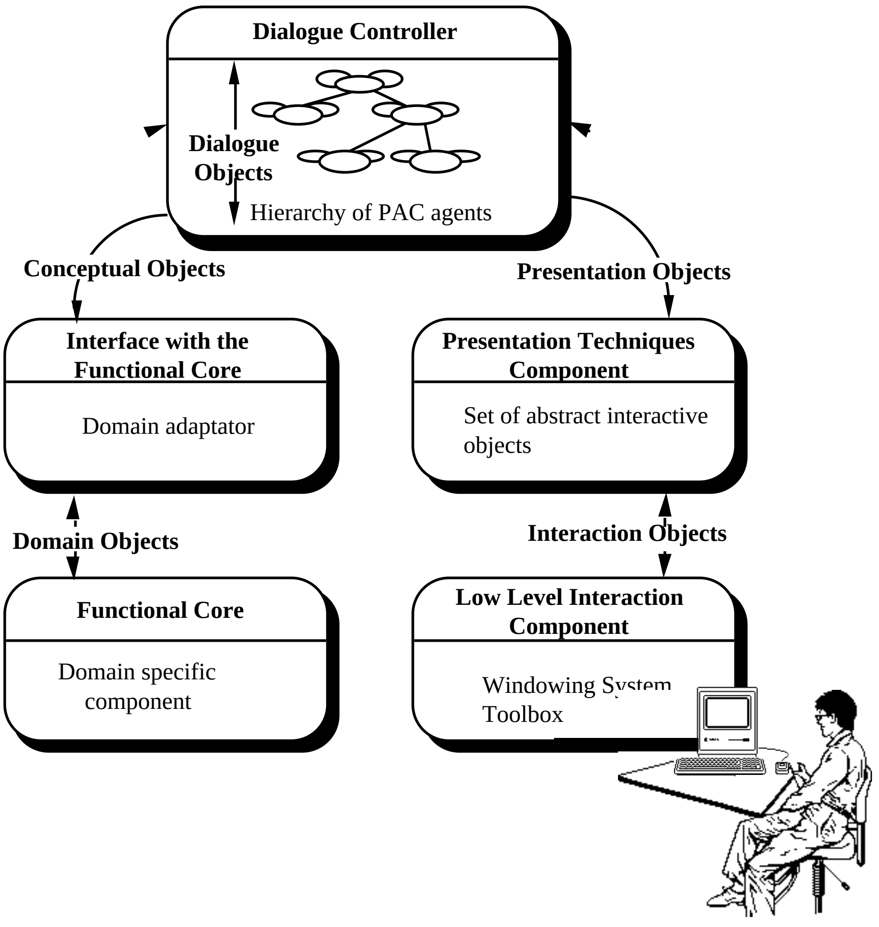


Figure 3 The PAC-Amodeus software components.

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. These define perspectives on domain objects intended to match the user's mental representation of domain concepts. In other words, 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 adaptation 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 subtle. 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^{o} 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. PAC-Amodeus decomposes the Dialogue Controller into a set of cooperative PAC agents [Coutaz 87].

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:

- An 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.
- 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 as shown in Figure 4. 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:

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

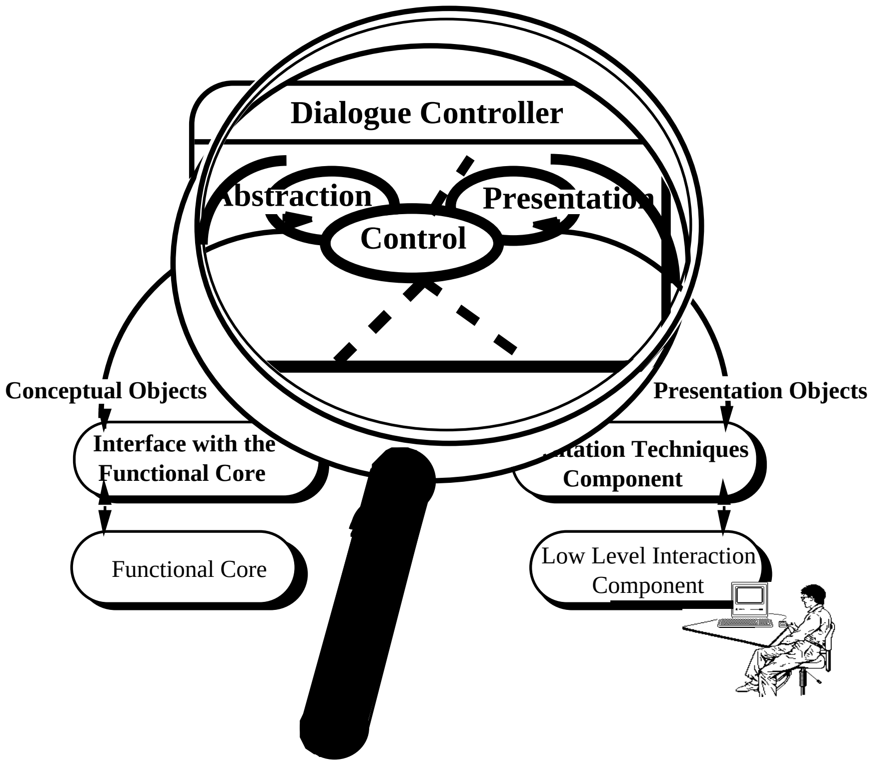


Figure 4 Zoom within the PAC-Amodeus Dialogue Controller, a PAC agent. Dashed lines represent possible relationships with other agents. Plain lines show the possible links with the surrounding components of the Dialogue Controller.

In summary, in PAC-Amodeus, the overall functional partitioning of an interactive system should be defined according to the following rule: the Low Level Interaction

Component should be both device and interaction language dependent; the Presentation Techniques Component is device independent but still language dependent; the other components of the interactive system, including the Dialogue Controller, should be both device and language independent. The Dialoge Controller should be refined in terms of PAC agents. The hierarchical organization of the Dialogue Controller in terms of PAC agents is motivated by the necessity of modelling computation at various levels of abstraction, the necessity of expressing the fusion and fission phenomena as well as parallelism at various levels of granularity. In addition to the vertical information flow within the PAC hierarchy, communication can also occur horizontally with the IFC and the PTC at various levels of abstraction through the Abstraction and Presentation facets of the PAC agents.

Having presented the main features of the PAC-Amodeus model, we are now able to discuss its relationships with internal and external properties.

3.2. User-centered design and software design: a point of contact using properties

In this section, we present the benefits of the PAC-Amodeus model in terms of external and internal properties. A list of external and internal properties can be found in [Gram-Cockton 96, chapter 2&3]. Here we present those that PAC-Amodeus deals with and that are useful for describing the case study. For each property, we point out whether the model carries the property or wether it supports its assessment.

3.2.1. External properties and PAC-Amodeus

In [Abowd 92] external properties are classified in two categories: properties that support interaction flexibility and properties that characterize interaction robustness. This initial set has been extended in [Gram-Cockton 96, chapter 2]. In this article, we examine the properties that relate to our case study only: device and representation multiplicity, multithreading, non-premptiveness, reachability, observability, and predictability.

“Device and Representation multiplicity” is defined as the capacity of the system to offer multiple input/output devices and interaction representations for communication. In PAC-Amodeus, the multiplicity of devices can be locally assessed in the Low Level Interaction Component (LLIC). The multiplicity of representations is implemented in terms of multiple presentation objects that are either managed in the Presentation facet of PAC agents in the Dialogue Controller (DC) or in the Presentation Techniques Component (PTC). PAC-Amodeus does not deliver “Device and Representation multiplicity” but supports its assessment by identifying the software components where the property should be implemented.

“Multithreading” of the user/system dialogue allows for support of more than one task at a time. “Multithreading” is delivered by PAC-Amodeus because of the agent decomposition of the Dialogue Controller (DC). Indeed an agent can be associated with one thread of the user's activity. Since a state is locally maintained by the agent, the interaction between the user and the agent can be suspended and resumed at the user's will. When a thread of activity is too complex or too rich to be represented by a single agent, it is then possible to use a collection of cooperating agents.

“Non-preemptiveness” refers to the degree of freedom the user has in deciding the next action to be performed. “Non-preemptiveness” has direct impact on the Dialogue Controller (DC). As explained above, the DC has the responsibility for task-level sequencing. Because PAC-Amodeus refines the DC in terms of small computational cooperative agents, it is easy to implement “non-preemptiveness”. Conversely, it may be difficult to assess the property since dialogue control is distributed over a multiplicity of Control facets.

“Reachability” is concerned with navigation through the system states. We make a distinction between “backward” and “forward” reachability. “Backward reachability”, such as undo, is useful for returning to some previous state. “Forward reachability” allows the user to reach any desired state from any current state. Each agent in the Dialogue Controller (DC) maintains a local state. The designer must therefore examine each agent to assess the property. Implementation of “reachability” may nevertheless be simpler because of its distributed implementation within every agent.

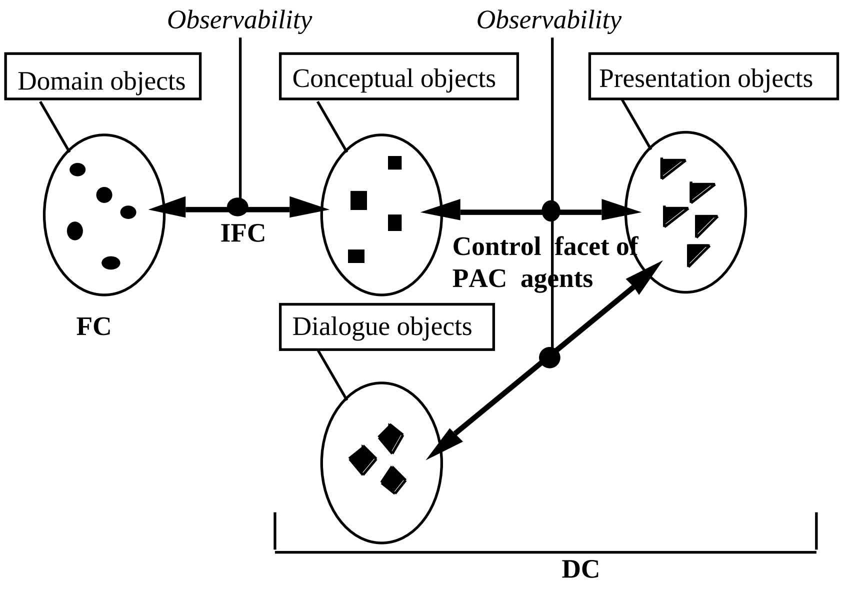


Figure 5 Observability property in the PAC-Amodeus model.

“Observability” allows the user to evaluate the internal state of the system from the perceivable representation of that state. It implies that the system makes relevant information available to the user. “Observability” is implemented in the Interface with the Functional Core (IFC), the Dialogue Controller (DC), and the Presentation Technique Component (PTC). Domain objects that are relevant for a given task must have a set of corresponding conceptual objects defined in the IFC. In turn, conceptual objects must be linked to presentation objects by the DC. In addition, each relevant piece of information (i.e.; dialogue object) maintained in the Abstraction facet of an agent that does not correspond to a conceptual object must also be related to a presentation object. Figure 5 summarizes our analysis of observability for PAC-Amodeus.

“Predictability” of an interactive system means that the user's knowledge of the interaction history is sufficient to determine the result of future interaction. The knowledge can be reduced to the current observable state of the system so that the user needs not remember anything that is not currently observable. “Predictability” is also closely related to the “consistency” property. It implies that the behavior of each agent in the Dialogue Controller must be consistent.

3.2.2. Internal properties and PAC-Amodeus

PAC-Amodeus is geared towards satisfying two internal properties: “modifiability” which is closely related to “maintainability” and “portability”.

“Modifiability” is the effort required to modify a program. We identify three steps in modifying a program:
- identify the component to be modified,
- modify the component,
- test the new component and its integration into the program.

Modifiability of the code level is a crucial property for supporting a user-centered iterative design. PAC-Amodeus incorporates the two adapter components of Arch, the Interface with the Functional Core (IFC) and the Presentation Techniques Component (PTC), in order to insulate the keystone component (i.e., the Dialogue Controller (DC)) from modifications in its unavoidable neighbours: the Functional Core (FC) and the Low Level Interaction Component (LLIC). For example, it is example possible to modify the input and output interaction techniques in the LLIC and PTC (devices and languages) without endangering the code of the Dialogue Controller.

“Portability” is the effort required to transfer a program from one hardware configuration and/or software system environment to another. The Presentation Techniques Component (PTC) acts as a mediator between the Dialogue Controller and the Low Level Interaction Component (LLIC) which is system environment dependent. It is therefore possible to change the underlying physical and software platform by only modifying the PTC.

The agent decomposition of the Dialogue Controller (DC) is a useful mechanism for satisfying additional internal properties: agent models in the DC stress a highly parallel modular organisation and distribute the state of the interaction among a collection of co-operating units. Modularity, parallelism and distribution are convenient mechanisms for supporting the iterative design of user interfaces and for implementing physically distributed applications:

- An agent defines the unit for functional modularity. It is thus possible to modify its internal behaviour without endangering the rest of the system.
- An agent defines the unit for processing. It is thus possible to execute it on a processor different from the processor where it was created. It is also possible to use instances of a class of agents to present a concept on distinct workstations. This property is essential for implementing groupware.

3.3. From model to reality: how to apply PAC-Amodeus

Our own experience with PAC-Amodeus reveals two interesting but dual observations:

- Software designers have few difficulties in understanding the role of the five software components of the arch. Thereforet, they have few problems in describing the functions and objects implemented in each component.
- Conversely, 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 demonstrates interesting properties with regard to interaction principles, it is too general to guide the software architecture design process. In particular, the refinement of the Dialogue Control component in terms of PAC agents neeeds to be made more precise. The heuristic rules presented next provide an operational apparatus for this refinement. These rules have been implemented in the form of an expert system, PAC-Expert [Nigay 94]. As mentioned in Section 2, they imply a bottom-up analysis starting from the external specifications of the system. They are organized along three issues: object and group content, group links and hierarchy revision. The concept of group can be mapped onto the “Objects group” defined in [May 96]: it corresponds to a presentation objects group as perceived (hopefully) by the user. For instance, a window on the screen forms a group. If the user interface is well designed, presentation objects in a group maintain a semantic relationship: they are related to the same task or to the same domain or dialogue object.

3.3.1. Object and group existence

Rule 1: Use an agent to implement an elementary presentation object.
We have observed that users of PAC-Amodeus have difficulty in bounding the recursive decomposition of the agent hierarchy. We recommend the following

bottom-up approach: refine the decomposition down to the elementary presentation objects as specified in the external specifications (e.g., a push button, etc.).

Rule 2: Use an intermediate agent to implement a group object.
We then recursively build upon the elementary agents by identifying group agents as shown in Figure 6. For example a tool palette is a group of push buttons, each of them being implemented as an agent.

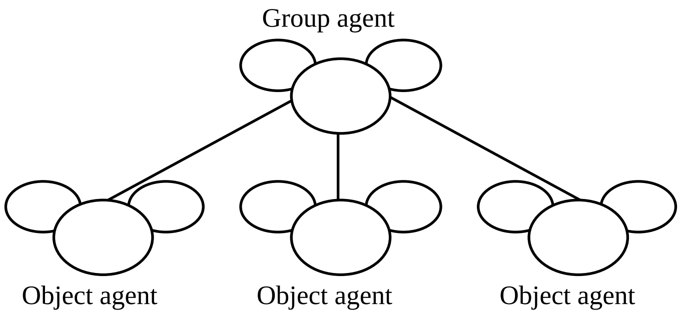


Figure 6 Object and group agents.

3.3.2. Group links

We identify two types of links: semantic and syntactic. Such links are not translated in terms of groups at the user interface level, otherwise Rules 1 and 2 can be applied. Rules 3 and 4 deal with semantic links while Rule 5 correspond to syntactic links.

Rule 3: Use isomophic agents to reflect the composition of conceptual objects.
It is often the case that conceptual objects are composed of conceptual objects. For example, a program is composed of modules which in turn is composed of procedures, etc. If the relation of composition must be conveyed to the user, then one can define a similar composition in terms of agents in the dialogue controller. A hierarchy of agents isomorphic to the hierarchy of conceptual objects is thus defined.

Rule 4: Use an agent to maintain visual consistency between multiple views.
According to the software principle which stresses “mutual ignorance” to enhance reusability, agents which implement the views of a semantic concept should not know each other. As shown in Figure 7, a “Multiple view” agent is introduced to express the dependency. Any action with visual side effect on a view is reported to the Multiple view which broadcasts the update to the other siblings.

Rule 5: Use an agent to synthesise actions distributed over multiple agents.

Group agents are related by a “syntactic link” when a set of user actions distributed over these agents can be synthesised into a higher abstraction (i.e., the fusion phenomenon). 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 synthesised by a cement agent into a higher abstraction (i.e., the command “create circle”). This agent, which maintains a syntactic link between its sub-agents, is called a “cement agent”. Again, if the tool palette and the drawing area are contained in the same window (a group) Rule 2 applies. Rule 5 would be applied if the tool palette were independent of the drawing area.

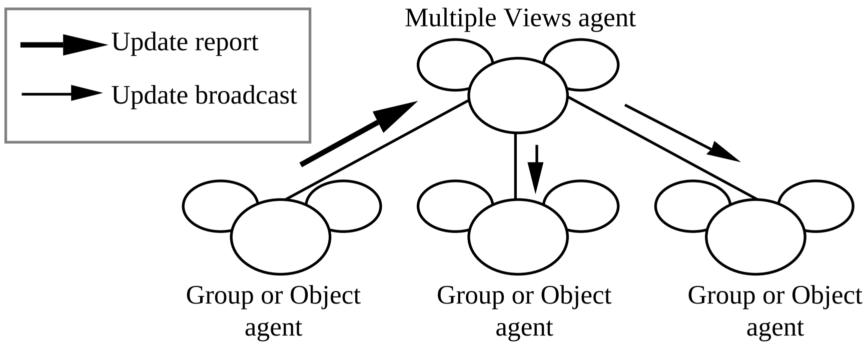


Figure 7 Multiple Views agent.

3.3.3. Hierarchy revision

Once the hierarchy of agents has been devised using rules 1 to 5, we recommend to analyze the hierarchy using the following rules.

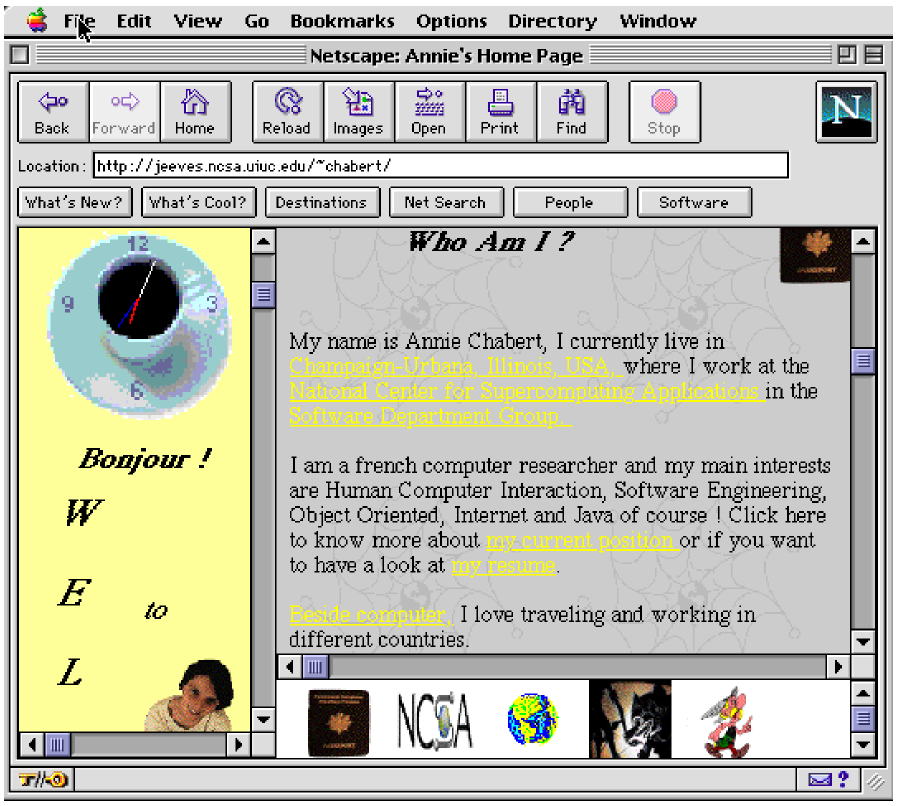
Rule 6: An interaction object is not an agent.
If the Presentation Techniques Presentation (PTC) levels can implement an agent in a straightforward way, then turn the agent into an interaction (or presentation) object and make it part of the Presentation facet of the parent agent (i.e., the facet gets connected to the interaction/presentation object). For example using a toolkit such as Motif™, a push button is not an agent because a push button is implemented by Motif™.

Rule 7: An agent and its single sibling can be combined into one agent.
If a parent agent has one single sibling and future evolution of the system does not lead to additional siblings and information transfer between agents is not for free, then it may be useful to combine the two agents into a single one.

We now illustrate the application of our heuristic rules with the description of a case study.

4. A case study: a WWW browser

Our case study is based on the Netscape [Netscape] user interface whose screen dump of Figure 8 shows an example. We show how to design the software architecture for this exemplar applying the model and the rules presented in Section 3. We then assess the resulting architectural design based on the internal and external properties presented in Section 3.2. In so doing, we show which external properties are satisfied in the Netscape user interface and which internal properties are supported by our PAC-Amodeus architectural design.



http://jeeves.ncsa.uiuc.edu/~chabert/

Who Am I ?

My name is Annie Chabert, I currently live in Champaign-Urbana, Illinois, USA, where I work at the National Center for Supercomputing Applications in the Software Department Group.

I am a french computer researcher and my main interests are Human Computer Interaction, Software Engineering, Object Oriented, Internet and Java of course ! Click here to know more about my current position or if you want to have a look at my resume.

Beside computer, I love traveling and working in different countries.

Figure 8 The Netscape user interface. The current page contains three frames [Netscape]. A frame is a mechanism that allows the page designer to display several HTML [HTML] pages within a single page.

4.1. Software design

In this section we explain how to generate the software design of the user interface of Figure 8 by applying the model. The first design step consists of identifying the functions and objects in the five software components advocated by PAC-Amodeus.

Section 4.1.1. describes the results of this design step. Having identified the content of the five components, we then recommend the refinement of the Dialogue Controller in terms of agents: Section 4.1.2. shows how we applied the rules in order to build the hierarchy of agents.

4.1.1. Overall architecture of the Netscape browser

As shown in Figure 9, the Functional Core (FC) hosts the functions that are dependent on the internet network connection (http protocol):

- connection to the distant web server,
- receipt of the HTML [HTML] pages.

In addition, the FC manages access to local files that can be directly loaded by the browser including user's preferences such as bookmarks, and possibly the cache memory. Cache memory is used to keep local copies of frequently accessed pages in order to reduce network load.

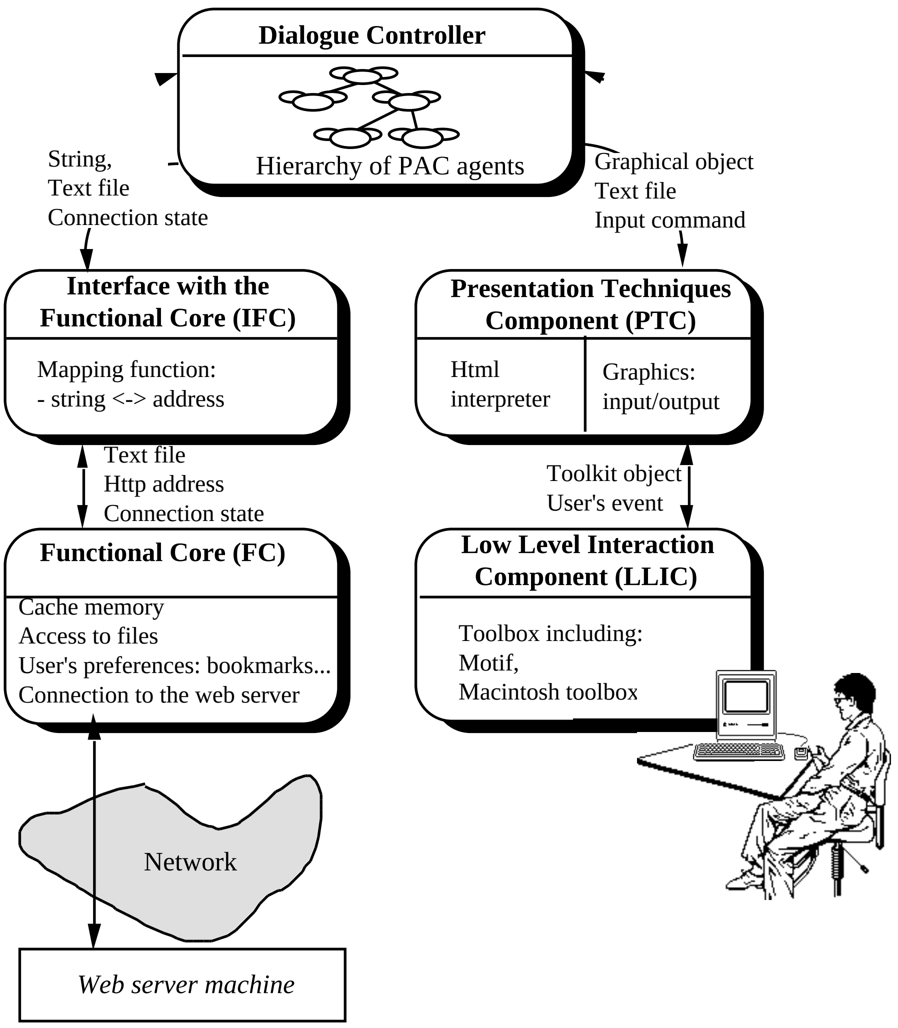


Figure 9 Overall architecture of the exemplar of Figure 8 developed using PAC-Amodeus.

The Interface with the Functional Core (IFC) serves as a buffer between the Dialogue Controller (DC) and the Functional Core (FC). It maintains a mapping between the concepts of the FC (e.g., http addresses and text files) and the concepts that belong to the DC (e.g., strings and pages). For example, the IFC receives a string from the DC that it translates into an http address understandable by the FC. The IFC allows communication between its two surrounding components by implementing a communication protocol. It is therefore possible to receive information through the network via the FC and to handle user events through the DC.

At the other end of the spectrum, the Low Level Interaction Component (LLIC) denotes the underlying software and hardware platform. It manages the graphical output and mouse-key events. As a result, this component is dependent on the underlying platform. For example, on the Macintosh™, this component embeds the Macintosh toolbox. On UNIX workstations, it includes the Motif™ toolbox.

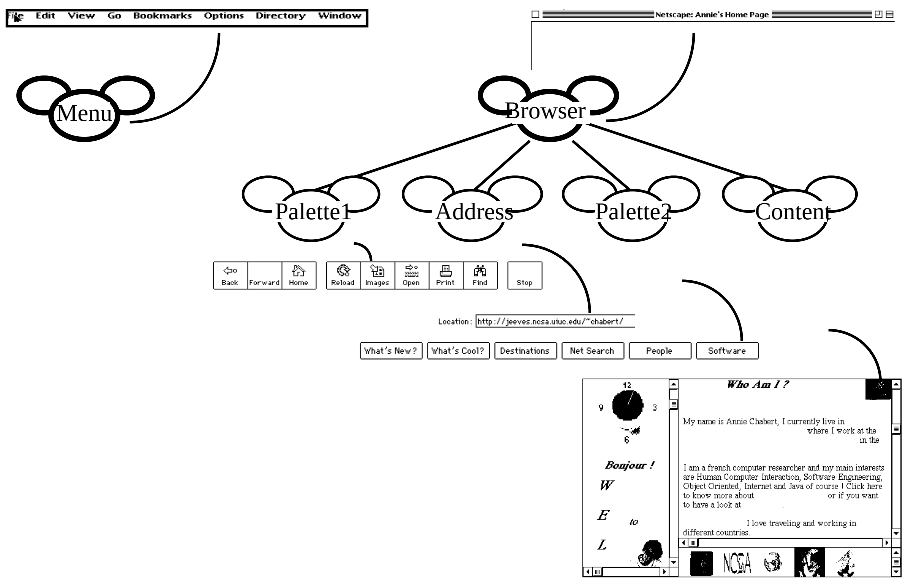
The Presentation Techniques Component (PTC) is split into two main parts: the graphical definition (input and output) and the HTML interpreter. The PTC receives presentation objects from the Dialogue Controller (DC) that are mapped onto interaction objects. Interaction objects are toolbox dependent. The PTC serves as a buffer between the underlying platform (LLIC) and the Dialogue Controller. It thus makes the DC platform independent.

Finally the Dialogue Controller is responsible for task sequencing. It is not dependent on the network protocol nor on the software and hardware platform. In particular, the DC is toolbox independent and HTML independent. We refine it in terms of PAC agents in the following section.

4.1.2. Refining the Dialogue Controller in terms of PAC agents

The hierarchy of PAC agents is derived using the rules presented in Section 3.3. We start with Rules 1 and 2: each presentation object, such as a button, is a PAC agent and each presentation group including palettes, menus and windows are agents. For example, the “Palette1” agent models the part of the window where the command buttons including “Back” and “Forward” are displayed. Its Abstraction facet maintains the list of corresponding commands. When the user selects one of the buttons, the corresponding event is received by the Presentation facet. The “Palette1” agent sends a message to the “Browser” agent which will in turn translate the command in terms understandable by the IFC: an http address. The Abstraction facet of the “Browser” agent will send this address to the IFC.

We then apply Rule 6 to minimize the number of levels in the hierarchy. Since toolboxes provide menus and buttons, we do not need agents to manage them. Using the user interface shown in Figure 8, we obtain the two hierarchies of agents presented in Figure 10.



My name is Annie Chabert, I currently live in where I work at the in the

I am a french computer researcher and my main interests are Human Computer Interaction, Software Engineering, Object Oriented, Internet and Java of course ! Click here to know more about or if you want to have a look at .

I love traveling and working in different countries.

Figure 10 First refinement step: result from the application of Rules 1, 2 and 6.

Because several browsers may be opened simultaneously, Rule 5 applies to maintain syntactic links between the multiple browsers. For example, images can be dragged and dropped between browsers. We apply Rule 5 to maintain those user's actions distributed over multiple browsers: Figure 11 presents the corresponding hierarchy.

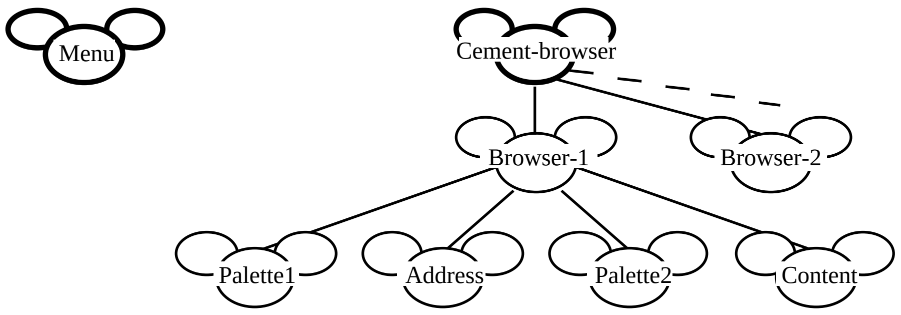


Figure 11 Second refinement step: result from the application of Rules 5.

The Control facet of the “Cement-browser” agent maintains a mapping between the current opened browsers and corresponding PAC-agents. This correspondence can be implemented as a table: when a new browser is created, the “Cement-browser” agent dynamically creates a new “Browser” agent and adds a new line in the table.

Finally, we identify syntactic links between the main menu (the “Menu” agent) and the opened browsers (the “Cement-browser” agent). For example the user can select an http address in the "Go" menu to load an already accessed HTML page. The selection will be received by the Presentation facet of the "Menu" agent and must thus be sent to the current active browser. A new "Cement-root" agent must then be designed to maintain such communication between the "Menu" agent and the "Cement-browser" agent. In order to do so, Rule 5 is again applied. Figure 12 presents the final hierarchy of agents that refines the Dialogue Controller.

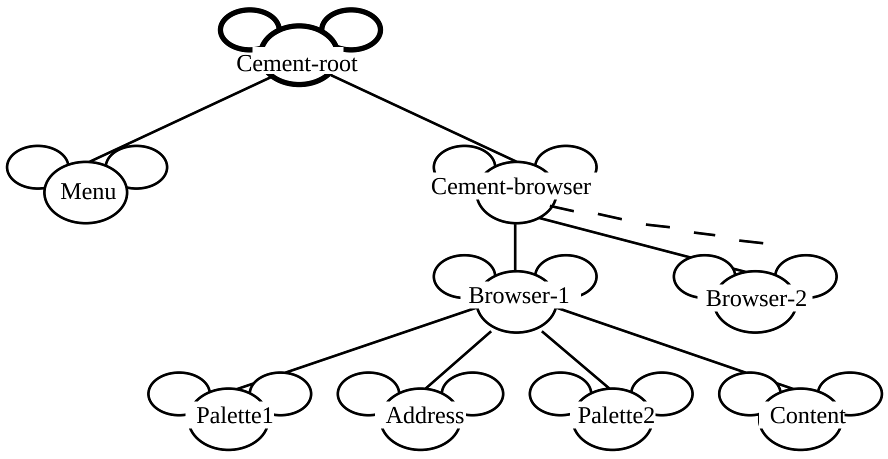


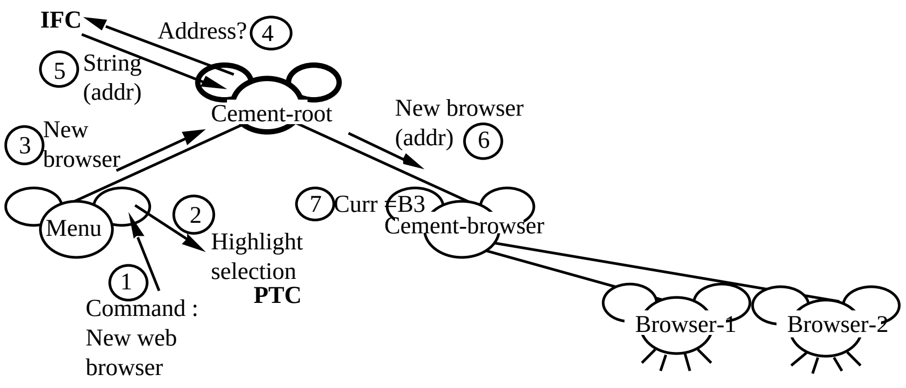
Figure 12 In the final refinement step; Rule 5 is applied.

4.2. Scenario: messages passing

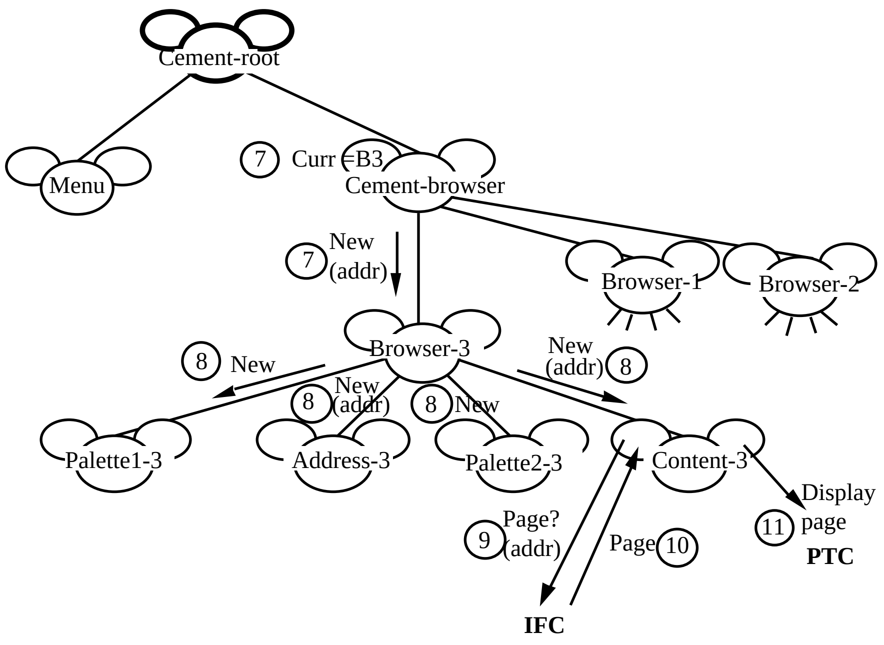
To better understand the roles of each agent, we explain the information flow through the hierarchy of agents in the context of two scenarios.

4.2.1. First scenario: dynamic instantiation of a browser

Figure 13 shows message passing through the hierarchy of agents in the context of the following scenario: the user has already opened two browsers. The user then selects the option "New Web Browser" in the "File" menu (1). The command is received by the Presentation facet of the "Menu" agent. It is important to note that the Dialogue Controller (DC) does not know how the user has specified the command: the DC is interaction technique independent. It receives semantic description of commands, not user's physical actions. For instance, the user can select the option in the menu by using the mouse or by typing the short cut " N" using the keyboard. Therefore a speech interface can be envisioned without modifying the DC. Spoken utterances, however, such as "Create a new browser", requires the modification of both the Low Level Interaction Component (LLIC) and the Presentation Techniques Component (PTC). The DC is left unchanged provided that the introduction of speech does not generate extra tasks such as negociations.



(a) Before the creation of new agents



(a) After the creation of new agents.

Figure 13 Opening a new browser: messages passing through the hierarchy of agents.

The Presentation facet of the “Menu” agent, which processes the user’s mouse selection (1), produces a partial immediate feedback: it highlights the selection. The “Menu” agent, which cannot perform any more processing, sends the command to its parent agent, the “Cement-root” (3).

The Abstraction facet of the “Cement-root” agent sends a message to the Interface with the Functional Core (IFC) asking for a first page (4). If a “Home” page has been specified by the user, the Functional Core (FC), via the IFC, will provide the corresponding http address to the DC (5). If there is no home page specified in the FC, the Abstraction facet of the "Cement-root" agent will receive the Netscape http address. Having received an address, the Control facet of the "Cement-root" agent sends a message to the "Cement-browser" agent in order to create a new browser. This message contains the address (6).

The "Cement-browser" agent dynamically creates a new "Browser-3" agent (Figure 13-b) and adds a new line in the mapping table that contains the identification of the newly created PAC agent (7). Meanwhile, the Abstraction facet updates the local variable that contains the identification of the current active browser (7).

In turn, the newly created "Browser-3" agent initializes its local history in its Abstraction facet and dynamically creates its sub-hierarchy (8). Initially, the new "Palette1-3" agent contains the two buttons "Back" and "Forward" set to inactive state. The state of the buttons is maintained by the Abstraction facet of "Palette1-3" while the presentation of the buttons is defined by the Presentation facet. The new "Address-3" agent displays the address provided by its parent agent. Finally, the Abstraction facet of the new "Content-3" agent sends a request to the IFC asking for the page that corresponds to the address (9). The Abstraction facet of "Content-3" will first receive messages from the IFC that describe the current internal state of the FC (e.g., "Connect: Contacting host..."). These states are displayed by the Presentation facet of "Content-3". When the Abstraction facet receives the text file (10), it is reflected back to the PTC via the Presentation facet of "Content-3" (11). The PTC is then able to interpret the new page and displays it. Because the address is now validated, the "Content-3" agent sends the address back to its parent. Through the hierarchy ("Cement-browser" -> "Cement-root" -> "Menu"), the address is sent back to the "Menu" agent that updates its "Go" and "Window" menus.

4.2.2. Second scenario: browsability using hypertext links

In this scenario, we consider browsability using hypertext links. Two browsers are opened. The user drags a link from a browser to the second one. The corresponding page is loaded in the second browser. The first browser remains unchanged and is still active.

As shown in Figure 14, the selection of the link is received by the Presentation facet of "Content-1" (1). Its Presentation facet performs a partial immediate feedback by changing the color of the selection (2) while its Control facet sends the corresponding address to its parent(3), the "Browser-1" agent. In turn, "Browser-1" sends the address to its parent "Cement-browser" (4). "Cement-browser" maintains the selected address in its Abstraction facet (5).

Drag events are processed locally by the PTC. When the user releases the mouse button, the corresponding event is received by the Presentation facet of the "Content-2" agent (6) which performs a lexical highlight feedback. The Control facet of

“Content-2” sends the event to the “Cement-browser” agent via the Control facet of the “Browser-2” agent (7 and 8).

Now, the “Cement-browser” agent is able to combine the two user's actions, action1 and actions 2 into a meaningful command. It asks “Browser-2” to open the new address that has been selected (10). As in the previous scenario, the “Browser-2” agent modifies its local history in its Abstraction facet and broadcasts the new address to the “Palette1-2” agent, to the “Address-2” agent and to the “Content-2” agent. The “Palette1-2” agent may have to modify the state of its “Back” button. The “Address-2” agent displays the new address and the “Content-2” agent sends a request to the IFC in order to obtain the new page to be displayed. When the page is received, it is displayed by the PTC. Because the address is correct, the “Content-2” agent sends it back to its parent as a confirmation message. As opposed to the previous scenario, the validated address is not sent to the “Menu” agent because the “Browser-2” agent is not active. But if the user selects the window of the second browser in order to make it active, the whole history of accessed pages maintained by the “Browser-2” agent will be sent back to the “Menu” agent.

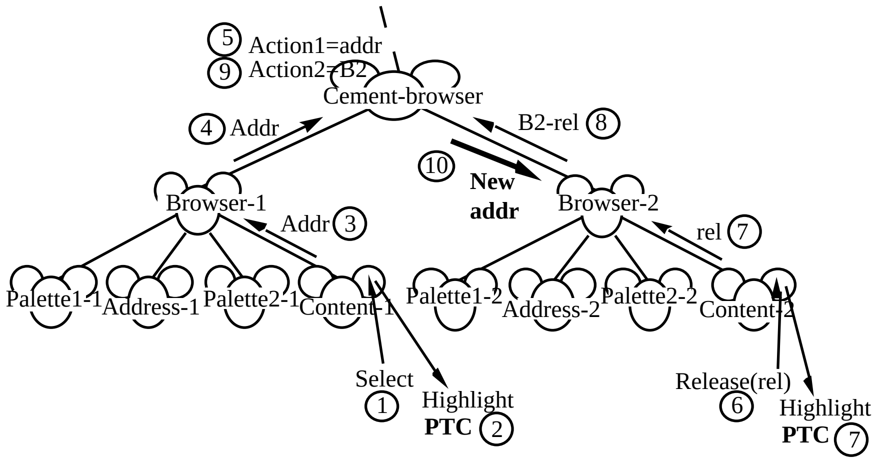


Figure 14 Browsing using links: messages passing through the hierarchy of agents.

4.3. Properties

Having designed the software architecture of the case study, we now consider our solution in the light of the external and internal properties selected in Section 3.2.

4.3.1. External properties

“Device and Representation multiplicity” is supported in the Low Level Interaction Component (LLIC) and the Presentation Techniques Component (PTC). For instance, in the first scenario (Section 4.2.2.) we present two ways of specifying the command "New Web Browser": the selection of a menu option using the mouse or by using a keyboard short-cut. This is one simple example of "device and representation multiplicity". If we add new interaction techniques such as speech input, the designer will have to modify the LLIC and the PTC. The Dialogue Controller will remain unchanged.

"Multithreading" is carried by agent decomposition of the Dialogue Controller. In the case study, the user can freely switch between browsers. Because we have designed a sub-hierarchy of agents per browser and a "Cement-browser" agent, the property is clearly satisfied.

"Non-preemptiveness" is visible at the Dialogue Controller (DC) level; the DC has the responsibility of task-level sequencing. At any time, every agent is ready to receive events from the Presentation Techniques Component (PTC) and from the Interface with the Functional Core (IFC). For example, the user can stop the process of loading a page at any time: because the IFC implements a communication protocol between the Functional Core (FC) and the DC, it is possible to receive information through the network via the FC and to handle user events (i.e., the "Stop" command) via the PTC: this is managed within the DC organized as a hierarchy of PAC agents.

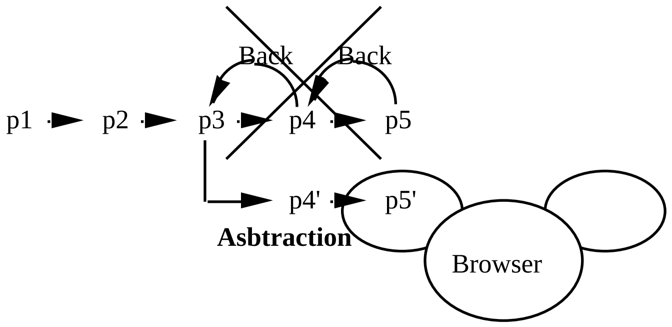


Figure 15 Interaction trajectory managed by the Abstraction facet of the "Browser" agent.

In order to assess the "reachability" property, the designer must examine the Abstraction facet of every agent: the implementation of the interaction state is distributed in the Abstraction facet of multiple agents. For example, let us consider the Abstraction facet of the "Browser" agent. It maintains the history of the accessed pages. It describes an interaction trajectory as shown in Figure 15: each state is an accessed page. By closely looking at the management of this history, we conclude that the "reachability" property is not satisfied because when the user comes back to a previous state and starts from there a new interaction trajectory, the previous trajectory is lost. It is no longer possible to come back to a previous state. To satisfy “reachability”, the designer must modify the Abstraction facet of the “Browser” agent.

As shown in Figure 5, “observability” can be easily checked in the Interface with the Functional Core (IFC) and in the Control facet of the agents that populate the Dialogue Controller (DC). For example, one can check that the internal states of the Functional Core (FC) are sent to the DC in order to be made perceivable to the user: to examine data passing, the designer must look into the IFC. In addition, within the DC, one can check that the internal state in the Abstraction facet of an agent is made perceivable to the user. In order to do so, we must look into the Control facet responsible for data passing between the Abstraction facet and the Presentation facet. Uing the “Browser” agent as an example, one can easily conclude that the history is not completely observable by the user. Only one part of the history is made perceivable to the user in the “Go” menu. For example, when the user selects links in frames [Netscape], the accessed pages are not presented in the “Go” menu. Instead, this information is maintained in the Abstraction facet of the “Browser” agent in order to return to a previous page within a frame. Such information displayed on screen would be very useful. Figure 16 schematically presents the problem related to the observability property in the “Browser” agent.

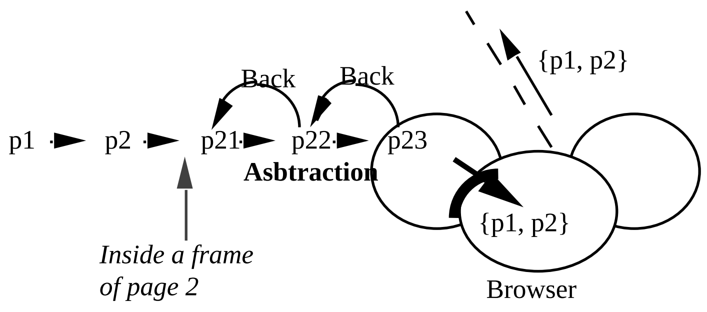


Figure 16 Observability and the “Browser” agent.

As explained in Section 3.2, the “predictability” property is closely related to the “consistency” property. To assess this property the designer must check that the behavior of the agents in the Dialogue Controller (DC) is consistent. For example we consider the management of frames within a page as shown in Figure 8. Each frame will be managed by an agent namely the “Frame” agent that is a sibling of the “Content” agent. From the user's perspective, such a frame will be perceived as small browser within a browser. But the behavior of the “Frame” agent is not consistent with the behavior of the “Content” agent. For example it is possible to drag a picture or a link from one browser to another one (scenario 2) but it is not possible to do so between two frames. This is a case of non-consistent behavior: when the user drags a link between two frames he cannot predict the correct result.

4.3.2. Internal properties

Modifiability of the software is clearly satisfied because of a high modularity of our software design and of the clearly identified links between modules. For example in the context of a user-centered iterative design we can modify the presentation of an agent without endangering the rest of the code. In the previous section we gave several examples of local modifications within a facet of an agent.

Portability of the software is satisfied because the Presentation Technique Component (PTC) makes the rest of the code independent of the underlying platform. The Dialogue Controller which is the more complex code is independent of the toolkit and the windowing system. This property is crucial for a WWW browser such as Netscape which runs on several platforms.

5. Conclusion

With the advent of new technologies such as WWW, the user interface portion of interactive systems is becoming increasingly large and complex. Architectural modelling is therefore becoming a central problem for large, complex systems. In addition, software architecture modelling is a design activity at the turning point between two worlds: the user interface design field and the programming field. Because of its interlinking location, software architecture must take into account properties of the two worlds: ergonomic properties of the designed user interface and properties of the future implemented software.

In this chapter we have focused on one software architecture model, PAC-Amodeus and a method to apply it. We have also explained how PAC-Amodeus can help the designer to satisfy and assess the ergonomic and software properties. The model, the method and the properties have been illustrated by a case study, a WWW browser.

6. Acknowledgements

This work has been supported by project ESPRIT BR 7040 Amodeus II. It was influenced by stimulating discussions with the IFIP WG2.7 “User Interface Engineering” members, and our Amodeus partners. Many thanks to G. Serghiou for reviewing the chapter.

7. References

[Abowd 92]

G. Abowd, J. Coutaz, L. Nigay, Structuring the Space of Interactive System Properties. In Proceedings of the IFIP TC2/WG2.7 on Engineering for Human-Computer Interaction, Ellivuori, Finland, 1992, 113-128.

[ANSI 83]

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

[Bass 91]

L. Bass, R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard and M. R. Szezur. The Arch Model: Seeheim Revisited, User Interface Developers' Workshop, April 26, 1991, Version 1.0.

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

[Coutaz 87] J. Coutaz. PAC, an Object Oriented Model for Dialog Design. In Proceedings of Interact'87, North Holland, 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.

[Gram-Cockton 96]
C. Gram, G. Cockton Ed., Design Principles for Interactive Software. Chapman&Hall, 1996.

[HTML]
http://www.w3.org

[May 96]
J. May, S. Scott, P. Barnard, Structuring Interfaces: a psychological guide. The ICS project, September, 1996. (http://www.shef.ac.uk/~pc1jm/guide.html)

[Netscape]
http://home.netscape.com

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

[Nigay 94]
L. Nigay, Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales. PhD dissertation, University of Grenoble, France, 1994, 315 pages.

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

[Shaw 95]
M. Shaw, D. Garlan, Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall, 1995.