Paru dans Proceedings CHI’97, ACM Publ., pp. 242-249

From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW

Gaëlle Calvary, Joëlle Coutaz, Laurence Nigay

CLIPS-IMAG, BP 53, 38041 Grenoble cedex 9, France
Tel. +33 76 51 48 54, Fax: +33 76 44 66 75
Email: {Gaelle.Calvary, Joelle.Coutaz, Laurence.Nigay}@imag.fr

ABSTRACT

This article reports our reflection on software architecture modelling for multi-user systems (or groupware). First, we introduce the notion of software architecture and make explicit the design steps that most software designers in HCI tend to blend in a fuzzy way. Building on general concepts and practice from main stream software engineering, we then present a comparative analysis of the most significant architecture models developed for single- and multi-user systems. We close with the presentation of PAC*, a new architectural framework for modelling and designing the software architecture of multi-user systems. PAC* is a motivated combination of existing architectural models selected for the complementarity of their “good properties”. These include operational heuristics such as rules for deriving agents in accordance to the task model or criteria for reasoning about replication, as well as properties such as support for style heterogeneity, portability, and reusability.

Keywords

PAC, interactor, agent, architectural style, software architecture modelling, CSCW, interactive systems.

INTRODUCTION

Architectural design of complex systems such as Computer Supported Collaborative Work (CSCW) can no longer emerge from pure craft skills. Ad-hoc unmotivated solutions, are certainly acceptable for throw-away prototypes. However, the iterative nature of user interface design leads developers to put too much effort into the prototype and thus, be tempted to turn the software into a product. Then, in the absence of an explicit architectural framework and sound design rationale, the resulting system is difficult to maintain and cannot evolve adequately.

One additional problem in Human Computer Interaction (HCI) is that off-the-shelf tools, such as application frameworks and user interface generators, do not make explicit the link between the functionalities they provide and their underlying architectural framework. The software architecture is lost in the resulting code. Typically, programmers must reverse-engineer the architecture of object-oriented application frameworks (e.g., MacApp) in order to reuse and extend the existing code appropriately. User interface generators do not alleviate the problem either. Worse, they tend to provide a false sense of confidence that software architecture is no longer an issue. Actually, the software designer must first understand the functional coverage of the generated code in order to devise what needs to be developed by hand. Second, developers have to discover how to integrate and coordinate the hand-coded portion with the generated code in a way that supports the system requirements. Without an architectural framework to structure the problem, it is difficult to achieve this task properly. Software Architecture (SA) modelling from main stream research in software engineering can contribute significantly to this problem analysis: it provides the right insights, it triggers the right questions and offers general tools for thoughts. Completed with practical experience in SA modelling for single-user interactive systems, it cannot be ignored for the software design of multi-user systems.

This article reports our reflection on software architecture modelling for multi-user systems (or groupware). It is structured in the following way: first, we introduce the notion of software architecture and make explicit the design steps that most software designers in HCI tend to blend in a fuzzy way. Building on general concepts and practice from main stream software engineering, we then present a comparative analysis of the most significant architecture models developed for single-user and multi-user systems. We close with the presentation of PAC*, a new architectural framework for modelling and designing the software architecture of multi-user interactive systems. PAC* draws on the contributions from main stream software engineering and blends together the best of single-user architecture modelling with the state-of-the art in multi-user systems.

MAIN STREAM SOFTWARE ARCHITECTURE MODELLING

The product of architectural design is a set of computational entities called "components" whose interactions are mediated by "connectors". In general, a software architecture is expressed in natural language complemented with a graph of labeled boxes (i.e., the components) and arrows (i.e., the connectors). Actually, there is more to software architecture modelling than identifying components and their connections. As in any design activity, the definition of a software architecture is the result of a process. In turn, a design process involves considering multiple concerns, thus developing multiple perspectives of the same design problem. The primary steps involved in the process of architectural modelling include:

- Identifying the functional breakdown of the system, i.e., organising the functional requirements of the system into simpler conceptual pieces;
- Defining the structure of the system, that is, providing a static description of the form of the system in terms of components and connectors. This structure may emerge from implicit hand-crafted knowledge or it can be inspired from architectural styles such as pipes and filters or layers of abstract machines. A style includes a vocabulary of design elements (e.g., pipes and filters), imposes configuration constraints on these elements (e.g., pipes are mono-directional connectors between two filter components), and determines a semantic interpretation that gives meaning to the system description [13];
- Allocating functions to the structure, i.e., mapping the functional breakdown onto the structural description of the system. One component may encapsulate multiple functional pieces. Alternatively, a function may be distributed across multiple components;
- Describing the coordination that is, the dynamic behavior of the architecture. This perspective on a system architecture is orthogonal to the specification of the structure. As components and connectors of a structural description express what is functionally significant, so the coordination model describes the important issues for the dynamic aspects of the system;
- Depending on the software development environment, the designer may be concerned with additional issues such as allocating structural components to processes and allocating processes to physical processors. Groupware development tools such as Groupkit [22] and Suite [9], tend to alleviate the problem of process and processor allocation.

While the application of principled design is widely recognised in the software community, there is very little material to help architectural designers to cope with a number of major difficulties. These include finding the right balance between multiple sources of requirements, identifying the right level of refinement to reason about the system properties, and coping with heterogeneity. As demonstrated by Shaw [24], styles have very specific properties. As a result, all of the system requirements may not be covered with a single style. In architectural design, heterogeneity is a necessary trouble. Typically, the user interface partition of an interactive system is built, partly from existing code with its own style(s) (inherited from toolboxes and interface builders), and partly from fresh code which may follow yet another style.

Architectural models developed for the software design of single user interactive systems have addressed some of these issues. Their contributions, which are to be considered for multi-user systems, are discussed next.

CONTRIBUTION FROM ARCHITECTURAL MODELLING FOR SINGLE USER SYSTEMS

Architecture models for single-user systems primarily provide frameworks for performing functional partitioning and for allocating functions to structural components using both system and user-centered properties. We have selected the most significant contributions in the field starting with the seminal Seeheim and Arch models. Then, we present the current trend with agent-based models, followed by PAC-Amodeus, a hybrid model that combines Arch with the PAC agent style.

Seeheim and Arch

The Seeheim model has provided the very first canonical functional decomposition for the UIMS technology [21]. We observe that, with the notion of Application Interface Model, Seeheim has identified an explicit location for resolving possible mismatches between the functional core and the user interface partition. Seeheim can be interpreted as a framework for pure functional partitioning or can be viewed as a structural decomposition where boxes denote monolithic software components dedicated to a single functional role, and where arrows represent some communication mechanism. For example, in the mid-eighties, user-system interaction was primarily viewed as a language-based dialogue. Consequently, the role of each component was roughly described as the semantic, syntactic and lexical aspects of the interaction, and the overall control structure of the system was assimilated as a pipe-line scheme. With the advent of direct manipulation, the semantic-syntactic-lexical interpretation of the Seeheim logical partitioning failed at supporting new requirements such as interleaving system feedback with user's inputs.

In 1991-1992, the Arch model revisited the functional coverage of the Seeheim partitions, introduced an additional adaptor for the presentation side, and its Slinky meta-model acknowledged the functional migration across the structural components [1]. With regard to functional coverage, Arch raised the level of abstraction of the user interface partitions. In particular, the token assembly role of the Seeheim Dialogue Component has been replaced with the responsibility for task-level sequencing. Token assembly, which is now performed by reusable software, has shifted to the low level Interaction Toolkit Component. To minimize the effects of future modifications, Arch has insulated the Dialogue Component from the diversity and variations of its functional partners, the Domain Specific and the Interaction Toolkit Components. By introducing the Domain Adaptor Component and the Presentation Component, Arch has provided explicit hooks for modifiability as well as for portability of the user interface.

Although Arch is a reliable reference, its overall decomposition is not always sufficient for reasoning about a particular software architecture design. Agent-based models, which promote refinement, tend to satisfy this need.

Agent-Based Models

Agent-based models structure an interactive system as a collection of specialised computational units called agents. An agent has a state, possesses an expertise, and is capable of initiating and reacting to events. Agents that communicate directly with the user are sometimes called interactors. The terms "interactor", "agent" and "interaction object" are sometimes used indifferently even if there is no direct interaction with the user. Agent models 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, for implementing physically distributed applications, and for handling multi-threaded dialogues.

A number of agent-based models and tools have been developed along these lines. MVC [14], PAC [5], ALV [15], the LIM [19] and York [10] models are typical agent-based styles. All of the agent-based styles and tools push forward the functional separation of concerns advocated by Seeheim. They generalise the distinction between concepts and presentation techniques by applying the separation at every level of abstraction and refinement. In order to do so, they distribute the functional separation of concerns among distinct co-operating agents. For example, in PAC (Presentation, Abstraction, Control), the facets of an agent are used to express different but complementary and strongly coupled computational perspectives of the same entity.

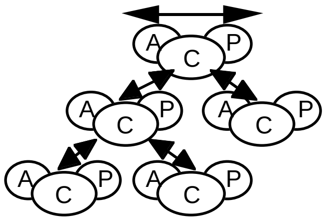


Figure 1: In PAC, the interactive system is modelled as a set of PAC agents whose communication scheme forms a hierarchy. Arrows illustrate the information flow (vertical flow between agents, horizontal flow between the facets of an agent).

A PAC agent has a Presentation facet (i.e., its perceivable input and output behavior), an Abstraction (i.e., its functional core), and a Control to express dependencies. The Control of an agent is in charge of communicating with other agents as well as expressing dependencies between the Abstract and Presentation facets of the agent. In the PAC style, no agent Abstraction is authorized to communicate directly with its corresponding Presentation and vice versa. In PAC, dependencies of any sort are conveyed via Controls. Controls serve as the glue mechanism to express coordination as well as formalism transformations that sit between abstract and concrete perspectives. In addition, the flow of information between agents transits through Controls in a hierarchical way (see Figure 1).

Agent-based models differ however in the way they perform the functional decomposition of an agent, and in the way they make explicit or not the I/O communication channels and the levels of abstractions. For example, in MVC, and in contrast to PAC, inputs and outputs are processed by two distinct facets (the C and the V, respectively) whereas there is no provision for expressing dependencies between the functional facets of an MVC agent.

More significant than their differences, agent styles have two important contributions:

- First, the capacity for the software designer to allocate a particular functional role at the adequate level of abstraction. Typically, domain dependent knowledge can migrate within the user interface portion to accomodate efficiency and the need for immediate semantic feedback;
- Second, the multi-faceted structure of an agent can be exploited in different ways. The issue is not that "my facets are better than yours", the point is that agents have multiple functional perspectives that should be exploited appropriately. As mentioned before, an architecture is an expression of what is significant. For example, in the AMF model, PAC agents have been augmented with facets to provide help or to log significant events to perform usability testing from observed behavior [24]. Other illustrations of dedicated facets to express significant functions will be illustrated with PAC*.

Agent styles model interactive systems in an homogeneous way: all of the functional aspects of the system are expressed using a single style. This homogeneity is desirable when the designer's goal is to reason about the system properties. Homogeneity is also acceptable when the style is conveyed by the implementation tool such as MVC within the Smalltalk development environment and when the entire system can be developed with the same tool. As mentionned above, heterogeneity is generally unavoidable. PAC-Amodeus has been designed to cope with this problem.

A hybrid model: PAC-Amodeus

PAC-Amodeus uses the Arch model as the foundation for the functional partitioning of an interactive system and populates the Dialogue Component with PAC agents: Arch supports the existence of reusable code and defines two adaptors for accomodating style heterogeneity, for anticipating changes and portability. On the other hand, Arch does not provide any guidance about how to structure the Dialogue Component in a way that is compatible with the user's task requirements. PAC supports task interleaving as well as multiple grains of task decomposition but fails at making explicit the link with existing styles. PAC-Amodeus gathers the best of the two worlds. Figure 2 shows the resulting functional breakdown.

As in Arch, PAC-Amodeus offers two-way information flows between the primary components of the arch. The nature of the connectors between the functional boundaries is left opened since it depends heavily on the case at hand. It may be procedure calls, pointers, message passing, or any other protocol suitable for the system requirements. Within the Dialogue Control component, we observe two information flows: the hierarchical traversal between PAC agents, and in contrast with the original PAC style, direct links with the Domain Adaptor (DA) and Presentation components (PC).

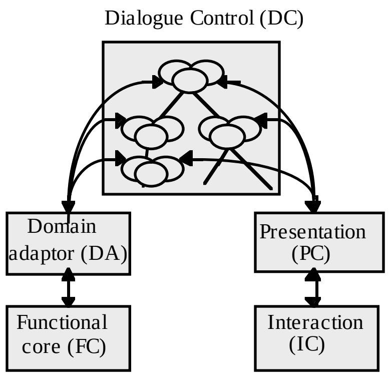


Figure 2: The PAC-Amodeus functional components. Arrows show information flow.

A PAC agent may be related to the DA and PC components through its Abstraction and Presentation facets respectively. Its Abstraction facet may be connected to one or multiple domain objects of the DA (or of the Functional Core if the DA does not exist). Similarly, a Presentation facet may be connected to one or multiple presentation objects of the PC (or to interaction objects of the Interaction component, if the PC does not exist). The design rationale for the "horizontal flow" is performance. Abstract information from the DA may not need additional processing from the parent agents. Similar reasoning holds for the presentation part. In this situation, flying through the PAC hierarchy would be both time consuming and useless.

We have identified a set of heuristic rules to help the designer in identifying the agents of the Dialogue Control. Each of these rules proposes a configuration of agents, or pattern, that fits a particular situation. A pattern describes a particular recurring design problem, proposes a pre-defined scheme for its solution, and includes heuristic rules for how and when to use it [12]. The complete set of PAC-Amodeus rules can be found in [17, 18]. We have selected one of them that will be used to illustrate PAC*.

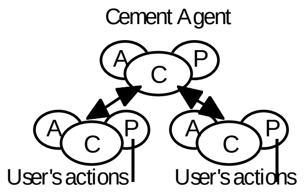


Figure 3: Pattern of PAC agents for "distributed actions".

If user's actions are distributed over multiple agents and if these actions are meaningful for the system, then a cement agent should be introduced to synthesise these actions distributed over multiple agents. This situation is illustrated in Figure 3. It primarily corresponds to editing systems. Typically, a palette of concepts is presented to the user along with an editing area. For example, to draw a circle, the user selects the circle icon from the palette then specifies the circle in the drawing area. These actions, which are distributed over two agents, are synthesized by the cement agent as a "draw circle" command along with its parameter values.

In summary, PAC-Amodeus is a hybrid model that:

- reuses the Arch functional decomposition for its known benefits;
- refines the key component of the architecture, (i.e., the Dialogue Control) in terms of PAC agents to comply with task analysis;
- includes patterns of agents that correspond to recurring situations;
- makes explicit the information flow but leaves open the nature of the connectors between the components;
- does not prescribe any refinement for the components of the arch other than the Dialogue Control, since those may be implemented from reusable code.

Having presented the background of software architecture modelling and its contribution to the software design of single-user systems, we need now to analyse its impact on groupware.

ARCHITECTURE MODELLING FOR GROUPWARE

Software architecture modelling for groupware must accomodate a large variety of requirements ranging from distributed systems and traditional HCI to more novel issues related to CSCW. The diversity and the novelty of the technical problems explain both the profusion of ad-hoc models and the lack of canonical models that would demonstrate sufficient genericity and coverage. In the following discussion, the coverage of an architectural model denotes its capacity to address the functional aspects of multi-user systems. Our "clover model", presented next, provides a high level partitioning for reasoning about the classes of functions a groupware may support. Using functional coverage as a criterion, we then analyse two significant models applicable to groupware systems: Dewan's model based on a layer style and ALV based on an agent style.

The Clover model

As shown in Figure 4 a), a groupware system covers three domain specific functions: production, coordination and communication [23].

- The production space denotes the set of domain objects that model the multi-user elaboration of common artefacts such as documents, or that motivate a common undertaking such as flying an airplane between two places. Typically, shared editors support the production space.
- The coordination space covers activities dependencies including temporal relationships between the multi-user activities. Workflow systems are primilarly concerned with coordination.
- The communication space supports person-to-person communication. Email and mediaspaces are examples of systems designed for supporting computer-mediated communication either asynchronously or synchronously.

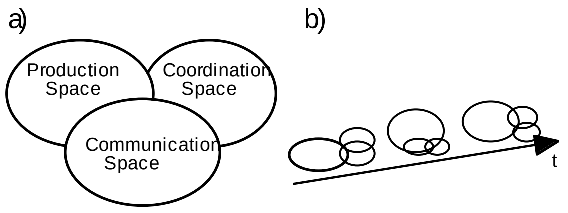


Figure 4: a) Groupware as a "functional clover" and b) its slinky property.

Our notions of production and coordination spaces correspond to Ellis' ontological and coordination models while our communication space complements Ellis' view of the functional decomposition of groupware [11]. Contrary to Ellis's model, user interface issues do not constitute a third functional aspect of groupware. Instead, it is orthogonal to *all* functional aspects of groupware. As for any domain specific function, the services provided by each of the three functional spaces must be accessible and observable through an appropriate user interface.

Interestingly, the relative importance of the three functional spaces depends on the particular groupware system at hand and may vary over time. Typically, shared editors favor production whereas communication functions are first class issues in mediaspaces. In addition, this functional (slinky) shift may vary over time (see Figure 4b). For example, at some point in the group activity, coordination is the focus of attention, possibly using computer-mediated communication to plan future common production.

Dewan's model

The "generic multi-user architecture" model proposed by Dewan [8] structures a groupware system into a variable number of levels of abstraction ranging from the domain specific level to the hardware level. Layers shared between users form the base of the system (e.g., layers S to L+1 in Figure 5a). At some point, the base gives rise to branches which are replicated for every user (see layers L to 0 in Figure 5a). Information flow between layers occurs vertically between adjacent layers along the input and output axis as well as horizontally between peer and non peer replicated layers for synchronizing states.

Dewan's model can be seen as an extension of Patterson's "zipper model" [20]: when a layer is replicated, all layers below it are necessarily replicated. This hypothesis does not comply with situations where multiple users, like in MMM [3], share the same physical workstation. On the other hand, this model offers a good basis for implementing various forms of coupling as well as for allocating functions to processes (e.g., reasoning about the granularity of parallelism, replication and distribution). For example, one can choose to execute the base and each branch within distinct processes. Similarly, without any automatic support from the underlying platform, the model helps reasoning about allocating processes to processors.

Genericity in Dewan's model comes from the notion of layer whose functional role and number can be adapted to the case at hand. A layer can be viewed as a level of abstraction or as a service that overlaps with other services as in SLICE[16]. Figure 5b) shows an instantiation of Dewan's model using the functional layers of Arch. Although generic, Dewan's model does not convey any structuring mechanism to reason about the functional aspects of the clover model.

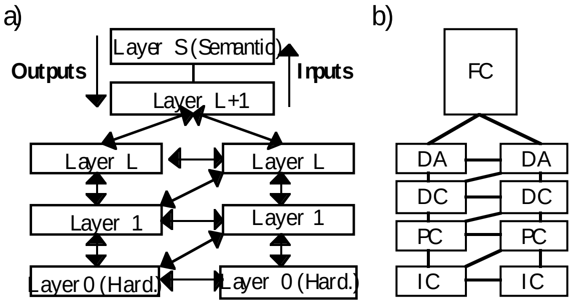


Figure 5: a) Dewan's generic architecture for multi-user systems. Layers S to L+1 are common to all users and not replicated. Layers L to 0 are replicated. Arrows denote information flow: the horizontal ones express the existence of some coupling between peer layers ; the diagonal ones denote some coupling between non peer components. b) An instantiation of Dewan's model using the functional layers of Arch.

The ALV model

ALV associates a personal interface component, a View, to every user and uses Link components to connect views to a shared Abstraction [15]. Links are in charge of expressing constraints between the views and the shared abstraction and of maintaining their mutual dependencies by a bidirectional propagation of events. Clearly, ALV addresses synchronous multi-user systems based on a centralized shared functional core. ALV can be seen as an instantiation of Dewan's model using three layers where the semantic level is mapped to the shared abstraction and where branches are comprised of links and views. Experimental evidence indicates that ALV primarily covers the production space based on direct manipulation and, to a lower extent, addresses the coordination space.

PAC*, A NEW FRAMEWORK FOR MULTI-USER SYSTEMS

Rationale

In Dewan's model, the number of functional layers is left opened. Experience shows that the five level breakdown of Arch offers an operational way of thinking about a system. As discussed above, Arch accomodates style heterogeneity, portability as well as code modifications. Whereas Dewan's architecture adopts a strict zipper model, PAC* conceptually authorizes multiple forks and joins along the levels of abstractions. But we have no experimental evidence of the soundness (or usefulness) of this generality.

In addition to Dewan, and drawing upon our experience with PAC-Amodeus, PAC* refines every Dialogue Control component of the architecture with agents. This systematic refinement could be pursued for the other components of the Arch but, as for PAC-Amodeus, we consider the other layers to be reusable code, or to be code developed with the style adequate for the case at hand. As demonstrated by PAC-Amodeus, an agent style is appropriate for modelling a Dialogue Control component in accordance to the task model. Therefore, in PAC*, agents of Dialogue Control components are derived using the same rules as those developed for PAC-Amodeus.

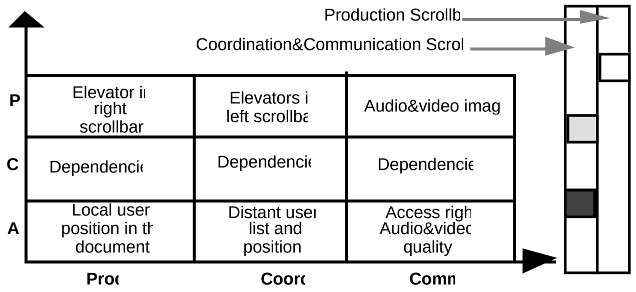
Going one step further than PAC-Amodeus agents, the functional role of each agent in PAC* is refined along two orthogonal axes: the PAC functional breakdown and the clover decomposition of groupware.

Functional decomposition of PAC* agents

Figure 6 illustrates the functional structure of a PAC* agent: the services that the agent provides for supporting the Production space are structured along the three PAC facets (the Presentation, the Abstraction, and the Control). Similarly, the services for Coordination and for Communication have their own Presentation, Abstraction and Control.

For example, let us consider an extended version of the multi-user scrollbar developed for the collaborative document editor SASSE [2]. The extended SASSE scrollbar supports both production, coordination, and communication. As shown at the bottom of Figure 6, a SASSE scrollbar is composed of two adjacent scrollbars:

- on the right, an ordinary scrollbar is owned by the local user to browse through the shared document: this component of the SASSE scrollbar supports personal production;
- on the left, a scrollbar includes one elevator per user currently working on the document. The elevators cannot be moved by the local user but they show the current location of the other users within the document: the left scrollbar supports coordination;
- by double clicking on an elevator of the left scroll bar, the local user can open an audio-video connection with the distant owner of the elevator: this extension of the SASSE multi-user scrollbar participates to the communication space.



Production Scrollb
Coordination&Communication Scrol
P
C
A
Elevator i right scrollbar
Elevators i left scrollba
Audio&video imag
Dependenci
Dependenci
Dependencie
Local user position in th document
Distant use list and position
Access righ Audio&videc quality
Proc
Coor
Comn

Figure 6: The functional decomposition of a PAC* agent illustrated with an extended version of the SASSE multi-user scrollbar.

The SASSE scrollbar can be modelled as a PAC* agent in the following way (refer to Figure 6):

- with regard to Production, the Abstract facet of the SASSE scrollbar denotes the position of the local user in the document; its Presentation refers to a single-user scrollbar as provided by most user interface toolkits. The Control facet includes two-way dependencies between the abstract position in the document and the position of the elevator of the right scrollbar;
- with regard to Coordination, the Abstract facet contains the users list as well as their position in the document, the Presentation corresponds to an extended version of an ordinary scrollbar. Referring to PAC-Amodeus rules, the extension may occur within the Presentation facet of the agent, or within the Presentation Component, or even in the Interaction Toolkit component. The Control facet is in charge of one way dependencies between the users list and their location, and the number of elevators and their position in the left scrollbar;
- with regard to Communication, the abstract facet may contain access control parameters in relation to privacy or resolution factors for specifying the quality of the audio and video services. The Presentation is the audio/video image of the distant user per se.

Communication in PAC*

Communication in PAC* inherits the communication rules devised for the models it is based on. As in Dewan's model, PAC* supports vertical communication between adjacent local layers as well as horizontal communications between remote peer layers. As discussed next, communication between replicated Dialogue Control components are performed at a finer grain than in Dewan's model.

Within Dialogue Control components, agents communicate according to the PAC-Amodeus rules: local agents communicate with each other in a hierarchical manner through their C facet; for every functional aspect of the groupware clover, they may communicate directly through their P and A facets with their local neighbours, respectively the local Presentation and the local Domain Adaptor Components. In addition to the local PAC-Amodeus based communication, agents may communicate with peer remote agents through their C facet.

Finally, within an agent, for every functional aspect of the groupware clover, the Abstract and Presentation facets communicate through their respective Control facet. Communication between the Production, Coordination and Communication views depends on the refinement chosen for PAC* agents.

Refinement of PAC* agents

As shown in Figure 7, a PAC* agent can be modelled as an encapsulated compact "Neapolitan PAC": the Presentation, Abstraction and Control facets are structured along the three functional aspects of the groupware clover.

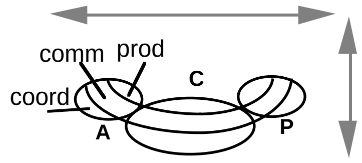


Figure 7: A PAC* agent as a bundled "Neapolitan PAC". Arrows show horizontal communication between the A, P, C as well as vertical communication between the Production, Coordination, and Communication views.

Alternatively, a PAC* agent can be decomposed as a cluster of three PAC agents, each agent being dedicated to one class of the functional clover.

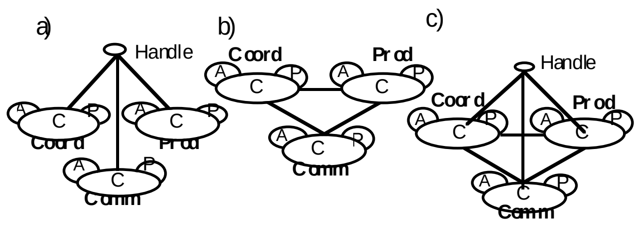


Figure 8: Refining a PAC* agent as clusters of PAC agents.

Figure 8 shows the possible breakdowns of a PAC* agent into a cluster of clover PAC agents. In 8 a), the agents of the PAC* cluster communicate through a single "handle". This handle is in charge of dependencies between the agents of the cluster as well as of the communications with the external world. In 8 b), the agents of the cluster communicate directly without any intermediary both between themselves and with the external world. Figure 8 c) shows a combination of direct and indirect communication. The choice between a bundled "Neapolitan PAC" and the various configurations of the PAC* cluster depends on the requirements of the system as well as on the development tools.

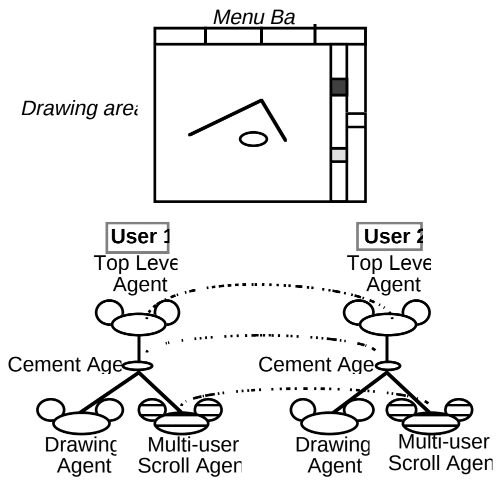


Figure 9: PAC* applied to the Dialog Control of a multi-user drawing editor using SASSE scrollbars. Stripes denote agents that support multiple aspects of the groupware clover. Dotted lines show possible direct communication between remote peer agents.

Figure 9 illustrates PAC* with a SASSE-like multi-user editor running on a heterogeneous environment. The top of the figure shows the view observed by each user. The overall architecture of the system is shown in Figure 5b: the Functional Core (FC) maintains an abstract representation of the shared document. Domain adaptors (DA) have been defined for each platform to resolve possible mismatches between the FC and the user interface portion of the system. The bottom of Figure 9 shows a zoom on the Dialogue Control components represented here for two users. Since every user has the same user interface running on remote workstations, Dialogue Control components are replicated. Agents of a Dialogue Control component are derived from the PAC-Amodeus rules (here, the cement agent rule applies to maintain consistency between the drawing area and the private scrollbar). Most agents support a single aspect of the groupware clover: they are pure PAC agents. The SASSE scrollbar, on the other hand, covers multiple functions of the clover. Therefore, it may be refined along the three clover perspectives as discussed above.

CONCLUSION

Architectural modelling is becoming a central problem for large, complex systems. Although interface builders tend to alleviate the problem, they are limited in scope and apply to mundane cases for which the user interface is often a second class component. Architecture design of user interfaces is not a luxury but a necessity and needs better support.

To respond to this requirement, we have developed PAC*, a motivated combination of existing architectural models selected for the complementarity of their "good properties". These include operational heuristics such as rules for deriving agents in accordance to the task model or criteria for reasoning about replication, as well as properties such as genericity and support for style heterogeneity, portability, and reusability. PAC* can be summarized in the following way: 1) it uses PAC-Amodeus as the basic functional breakdown of a system, 2) it draws upon Dewan's model for reasoning about replication, sharing, and communication between components, and 3) it populates Dialogue Control components, the keystone elements in the architecture of an interactive system, with agents that each covers the functional clover of groupware.

PAC* is being applied to the software architectural design of two ongoing projects: CoMedi, a prototype media-space for experimenting with both social and technical aspects of Computer Mediated Communication [6], and CATCHI, a software environment that supports the design of multi-user war games simulators. So far, PAC* has proven useful in triggering the right software design questions and in providing operational answers for most cases. We do feel however that PAC* patterns need to be devised to respond to recurrent but difficult problems in multi-user systems such as response time and fault tolerance.

ACKNOWLEDGEMENTS
This work has been partly supported by project ESPRIT BR 7040 Amodeus II and by France Telecom CNET.

REFERENCES

1. A Metamodel for the Runtime Architecture of an Interactive System, The UIMS Tool Developers Workshop, *SIGCHI Bull*., ACM, 24, 1, 1992, 32-37.
2. Baecker, R.M., Nastos, D., Posner, L.R. and Mawlby, M.K. The user-centered iterative design of collaborative writing software, in *Proceedings of theWorkshop on Real Time Group Drawing and Writing Tools*, CSCW' 92 (Toronto, 1992).
3. Bier, E.A. and Freeman, S. MMM: A User Interface Architecture for Shared Editors on a Single Screen, in *Proc. UIST' 91*, ACM Symposium on User Interface Software and Technology, 1991, 79-86.
4. Boehm, B. Architectures as a Key Reasoning About Software System Attributes, *Workshop on Software Architectures, Dagstuhl Seminar Report*; 106 20.02.-24.02.95 (9508), Garlan, D., Paulisch, F., Tichy, W. eds., 1995, 12-13.
5. Coutaz, J. PAC, an Object Oriented Model for Dialog Design, in *Proceedings Interact'87* (North Holland, 1987), 431-436.
6. Coutaz, J., Bérard, F. and Crowley, J. Coordination of Perceptual Processes for Computer Mediated Communication. in *Proc. Second International Conference on Automatic Face and Gesture Recognition*, IEEE Computer Society Press Publ. Oct. 1996, to appear.
7. Denning, P.J. and Dargan, P.A. A Discipline of SoftwareArchitecture, *Interactions*, (1) 1, ACM Publ., 1994, 55-65.
8. Dewan, P. Multiuser Architectures, in *Proceedings EHCI'95*, Working Conference on Engineering Computer Human Interaction.
9. Dewan, P. A tour of the Suite User Interface Software, in *Proceedings UIST'90*, ACM, 1990, 57-65
10. Duke, D. and Harrison, M. Folding Human Factors into Rigourous Development, in *Proceedings of Eurographics Workshop "Design, Specification, Verification of Interactive Systems"*, Paterno', F. ed., 1994, 335-352.
11. Ellis, C. and Wainer, J. A Conceptual Model of Groupware, in *Proceedings CSCW'94*, ACM Conference on Computer Supported Cooperative Work, Furuta, R., Neuwirth, C. eds., 1994, 79-88.
12. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA. 1995.
13. Garlan, D. and Shaw, M., *An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering*, Ambriola, V. and Tortora, G. eds., Vol. 1, World Scientific Publ., 1993, 1-39.
14. Goldberg, A., *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley Publ., 1984.
15. Hill, R. The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications, in *proceedings CHI'92* (NewYork, 1992), ACM, 335-342.

16. Karsenty, A. and Beaudoin-Lafon, M. SLICE: a Logical Model for Shared Editors, in *Real Time Group Drawing and Writing Tools*, Greenberg, S., Haynes, S., Rada, R. Eds, McGraw-Hill.
17. Nigay, L. and Coutaz, J. Building User Interfaces: Organizing Software Agents. In *Proceedings ESPRIT'91 Conference*, 1991, 707-719.
18. Nigay, L. *Conception et modélisation logicielles des systèmes interactifs: application aux interfaces multimodales*, Thèse de doctorat de l’UJF, 1994.
19. Paterno', F., Leonardi, A. and Pangoli, S. A Tool Supported Approach to the Refinement of Interactive Systems, in the *Proceedings of Eurographics Workshop "Design, Specification, Verification of Interactive Systems"*, Paterno', F. ed., 1994, 85-96.
20. Patterson, J.F. A taxonomy of Architectures for Synchronous Groupware Applications, *Workshop on Software Architectures for Cooperative Systems, CSCW’94*, ACM Conference on Computer Supported Cooperative Work.
21. Pfaff G.E. et al., *User Interface Management Systems*, Pfaff, G.E. ed., Eurographics Seminars, Springer Verlag, 1985.
22. Roseman, M. and Greenberg, S. GROUPKIT: A groupware Toolkit for Building Real-Time Conferencing Applications, in *Proc. CSCW'92* (Toronto, Canada, 1992), ACM Conference on CSCW, 43-50
23. Salber, D. *De l'interaction individuelle aux systèmes multi-utilisateurs. L'exemple de la Communication Homme-Homme-Médiatisée*, Thèse de doctorat de l'Université Joseph Fourier, September, 1995.
24. Shaw, M. and Garlan, G. *Software Architecture, Perspectives on an Emerging Discipline*, Prentice Hall, 1995.