Published in DSV-IS'97, 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, Springer Verlag Publ., pp. 15-32

PAC-ing the Architecture of Your User Interface

Joëlle Coutaz

Clips-IMAG, BP 53
38041 Grenoble Cedex 9, France
Joelle.Coutaz@imag.fr, http://outlet.imag.fr/coutaz

Abstract A number of architectural models, such as PAC, are available for the software design of interactive systems. These design abstractions, however, are not always clearly articulated nor do they explicitly exploit the foundational concepts developed recently in main-stream software architecture engineering. Similarly, technical solutions from main-stream software engineering may improve portability and reusability at the code level while hindering the quality of the resulting user interfaces. This article is an attempt to undertake an explicit bridging effort between software engineering and the specific domain of user interface software design using PAC as the running example. We present a brief evolution of the architectural models for single-user systems that motivated PAC. We then unfold PAC into PAC* for designing the conceptual architecture of multi-user systems.

Keywords : Software architecture modelling, multi-agent modelling, PAC, PAC-Amodeus, PAC*.

1 Introduction

Software architectural design is progressing from craft skill to engineering discipline. We observe an increasing interest in applying scientific knowledge to resolve conflicting constraints and requirements in a form suitable for software practitioners [28]. While architecture modelling as a recognized research field is a recent phenomenon in software engineering, it emerged explicitly in the early eighties in the specific domain of interactive systems. In this particular area, the iterative nature of the development process stimulated the design of conceptual tools that would minimize the effects of future changes: necessity triggered invention.

Today, a number of architectural models, such as PAC, are available for the software design of interactive systems. These design abstractions, however, are not always clearly articulated nor do they explicitly exploit the foundational concepts developed recently in main-stream software architecture engineering. The time is ripe to undertake an explicit bridging effort.

In this article, we analyze the PAC model in the light of the concepts devised in main-stream software engineering. These contributions are summarized in the following section. In Section 3, we present a brief evolution of architectural models such as

PAC for single-user systems. Section 4 unfolds PAC into PAC* and demonstrates the extension of PAC for designing the conceptual architecture of multi-user systems.

2 Foundational concepts and factors

Foundational concepts for software architecture modelling define an initial common ground for understanding the nature of an architecture. Factors make explicit requirements that a particular design solution should satisfy.

2.1 Concepts

Foundational concepts include the notions of component, connector, style, pattern, as well as the acknowledgment of the multiplicity of perspectives on a software architecture.

One perspective is the static structural decomposition of a system at the conceptual level. Another one is the dynamic behavioral description of the structure. A static structural decomposition is expressed in terms of primary computational and storage entities, called *components*, mediated by *connectors* such as procedure calls, client-server protocols, and event-based message passing. The structure may emerge from implicit hand-crafted knowledge or may be inspired from a style.

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, 28]. Within a style, patterns may emerge.

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]. Patterns solutions can be viewed as micro-architectures within a broad architectural picture.

Architectural static descriptions are primarily expressed using boxes to represent components, and arrows to denote connectors. Although box-and-arrow notations along with natural language are pervasive in the software community, they rely on common intuitions and past experience to make sense. Formal models and architecture definition languages [28] are intended to provide precise unambiguous descriptions as well as analytical techniques for sound design decisions. Although the benefits from such tools are unquestionable, their use, in practice, is still limited.

2.2 Factors

McCall has defined the foundational factors for software engineering [19]. Modifiability is one pervasive factor for the software design of user interfaces. In this section, we discuss two other prevalent factors: portability and reusability. We analyze how they are addressed by current tools and techniques and how they relate to user interface requirements.

Portability at the code level is claimed to be supported with general purpose abstract machines such as Java [2]. Actually, portability of user interfaces is more than platform-independent code execution. Typically, screen definition and processing power vary across workstations. As a result, the rendering and responsiveness of a Java applet may be satisfactory on the developer's workstation, while not usable for a remote Internet user. The developer must have access to some model of the characteristics of the unknown target site. Such information should not be hidden by layers of abstract machines. It should be clearly advertised to programmers.

Reusability is supported in various and complementary ways. It includes the subclassing mechanism as advocated by object-oriented languages, and the compositional approach. Reusability supported by object-oriented application frameworks and toolkits requires a thorough understanding of the environment in order to elicit what can be reused. If successful at discovering the right pieces, the next stage is to decide how to perform specializations and extensions while preserving the philosophy of the underlying conceptual architecture. Because this architecture is not self-explanatory, because it is not exoskeletal [18], reusability by specialization requires a time-consuming reverse engineering process.

The compositional approach to software is intended to make possible the creation of functionally powerful applications from existing systems. In general, system integration is hard to achieve due to conflicting architectural assumptions. As reported in [14], mismatches may occur both at the component and the connector levels:

- the control models used in the components may be incompatible (e.g., several of them may expect to own the main event loop),
- conflicts may arise between data models (e.g., one component is expecting a hierarchical representation while another one relies on a stream-like model),
- protocols implemented in the connectors may be inconsistent (e.g., connectors use different communication primitives or different kinds of data).

Techniques, such as Ole, for alleviating architectural mismatch are emerging. However, their conceptual applicability or their technical availability is of limited scope. For example, one general solution to the "event-loop problem" is to run the conflicting components as distinct processes and use event gateways to accommodate different control regimes. If, on the other hand, some design decision constrains the components to operate in the same process, then brute force modifications of the source code must be performed. CGI gateways are attractive solutions to the composition problem but they apply to Internet components only [15].

Actually, there is more to the compositional approach than technical conflicts: user interface inconsistencies may arise from components that implement their own user interface. Then, either conflicts occur at the lexical level and can be repaired by editing resource files, or the source of inconsistencies is deeper in the interaction model, and, again, brute force modification is unavoidable. Performing these modifications in a safe way requires reverse engineering the source code. This may be a tremendous task

when the conceptual architecture of the user interface such as those presented next, is missing.

3 Evolution of software architecture models for interactive systems

Seminal Seeheim [25] and its revisited version, the Arch model [1], set the foundations for functional partitioning. Modifiability, motivated by the iterative design of user interfaces, was the driving principle. At that time, the slogan was "separate the functional core from the user interface". With the proliferation of toolkits, portability of the user interface at the widget level became a new concern: the Arch "Presentation Component" made explicit the notion of logical interaction object. By doing so, it defined a natural integration for virtual toolboxes such as XVT [29].

Direct manipulation introduced new functional requirements on software architectures including the notion of multi-threaded interaction and the concept of immediate semantic feedback. The PAC model (Presentation, Abstraction, Control), based on the notion of agent, was explicitly designed to support these new features while preserving the Seeheim principle [5]:

- an agent is intended to convey a thread of interaction,
- the three-fold functional partitioning of an agent makes provision for hosting domain-specific semantic in the Abstraction facet of the agent while immediate feedback is supported through its Presentation facet; the third facet, the Control, centralizes the expression of dependencies between the Abstraction and the Presentation as well as inter-agent communication.

Multimodal interaction, which supports multiple forms of interaction techniques such as speech and gesture, added yet another set of requirements. In particular, the CARE properties (Complementarity, Assignation, Redundancy, Equivalence) that characterize relationships between multiple modalities [6], made prevalent temporal constraints and style heterogeneity:

- the interpretation of deictic expressions such as "put that there", uses the notion of temporal window to combine partial information into meaningful expressions: gesture and speech expressions produced in the same temporal window can be considered for fusion. Alternatively, if they are not part of the same temporal window, then redundancy or parallel input of multiple independent commands may be planned;
- style heterogeneity comes from the integration of multiple sources of reusable code that each implements a modality interpreter. Typically, a speech recognition system that may adhere to a blackboard style, must be combined with an object-event-based style graphical toolkit such as Motif.

PAC-Amodeus has been devised to support the new requirements imposed by multimodality [20, 21]. It uses the Arch model as the foundation for the functional partitioning and populates the Dialogue Component of the arch with PAC agents. The

two adapters of the Arch, the "Presentation Component" and the "Domain Adapter", are used to accommodate style mismatch. The PAC agents, which support concurrency, are augmented with processing capabilities, such as a fusion engine [22], to support the temporal aspects of the CARE properties.

So far, we have limited our historical analysis to single user systems. With the advent of groupware and Internet-based multimedia systems, complex functional requirements are emerging. Portability, reusability, interoperability, and efficiency are becoming crucial. In Section 2, we have discussed the limits of the current technical solutions. We now present PAC*, our conceptual architecture for groupware.

4 The PAC* conceptual model

PAC* is a combination of existing architectural models: 1) it instantiates Dewan's zipper model [9] with the five level functional decomposition of Arch, 2) like PAC-Amodeus, it populates Dialogue Control components with agents [20, 21], and 3) PAC agents are refined along the three functional dimensions of the Clover model [4].

This combination of Dewan's, Arch, and PAC models is motivated in the following way: the Arch functional decomposition is an appropriate framework for accommodating style heterogeneity and for anticipating changes and portability of the user interface portion of the system. Dewan's model provides the basis for reasoning about replication, coupling, and parallelism at the layers level. PAC-Amodeus explicitly models parallelism at a finer grain within Dialogue Control components of the architecture. It also brings in patterns and operational heuristics to devise agents in conformance to the external specifications of the user interface. The Clover model helps reasoning about the functional coverage of individual agents and provides the basis for additional refinements. The description of these models are briefly summarized in the following sections.

4.1 PAC* and Dewan's model

The "generic multi-user architecture" model proposed by Dewan [9] 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 1a). At some point, the base gives rise to branches which are replicated for every user (see layers L to 0 in Figure 1a). Information flow between layers occurs vertically between adjacent layers along the input and output axis as well as horizontally for synchronizing states between peer and non peer replicated layers. Interestingly, Dewan makes a distinction between single-user events which are private to a personal workstation, and multi-user events which result in propagations to remote workstations.

The functional role and the number of layers in Dewan's model depend on 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[17]. Figure 1b) shows an instantiation of Dewan's model using the functional layers of Arch. ALV [16] is a three layer instantiation where the

semantic level is mapped to the shared abstraction A and where branches are comprised of links L and views V.

Dewan's 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.

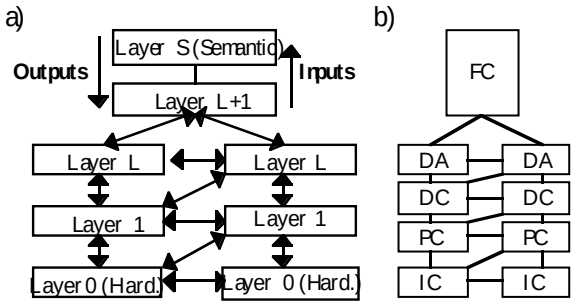

Fig. 1. a) Dewan's generic architecture for multi-user systems. Arrows denote information flow. b) An instantiation of Dewan's model using the functional layers of Arch. Here, the functional Core FC is shared while each branch is composed of a Domain Adapter DA, a Dialogue control Component DC, a logical Presentation Component PC, and a physical Interaction Component IC.

PAC* is a zipper-based architecture where the layers are instantiated using the five functional partitioning advocated by Arch: the Functional Core, the Domain Adapter, the Dialogue Component, the Presentation and Interaction Components. Although specified informally, the roles of these components are now well-understood by practitioners to cope with architectural mismatches, to reason about portability and to integrate multiple interaction techniques:

- Typically, the Domain Adapter is used as a wrapper for solving data models mismatch between the Functional Core and its user interface. For example, requests formulated from a Java user interface applet are transformed by the Domain Adapter into the SQL format expected by information retrieval engines [30].
- For interactionally rich user interfaces, such as MATIS [22] and CoMedi [7], the Presentation and Interaction Components correspond to the set of modality interpreters used in the user interface: speech recognition, computer vision-based gesture interpretation, and graphical abstract machines such as AWT [2] and Tk [23]. As shown in Figure 2, these machines sit side by side without any intercommunication and cover two levels of abstraction: the Interaction layer which provides the Presentation layer with device independence, and the Presentation layer which provides the Dialogue Component with interaction language independence. In addition, all of the interpreters of the Presentation layer

can be encapsulated in a uniform way. For example, in CoMedi, two speech recognition systems, computer-vision trackers, and the continuous media provider are all encapsulated in Tcl for use from the Dialogue Component.

- The Dialogue Component is in charge of task level sequencing. In PAC*, we reuse the design rationale developed for PAC-Amodeus: whereas the Functional Core, the Presentation and the Interaction Components have their own architecture driven by domain (or subdomain) specific criteria, we recommend to structure the Dialogue Component using the PAC style in order to support multi-threading, modularity, and task conformance.

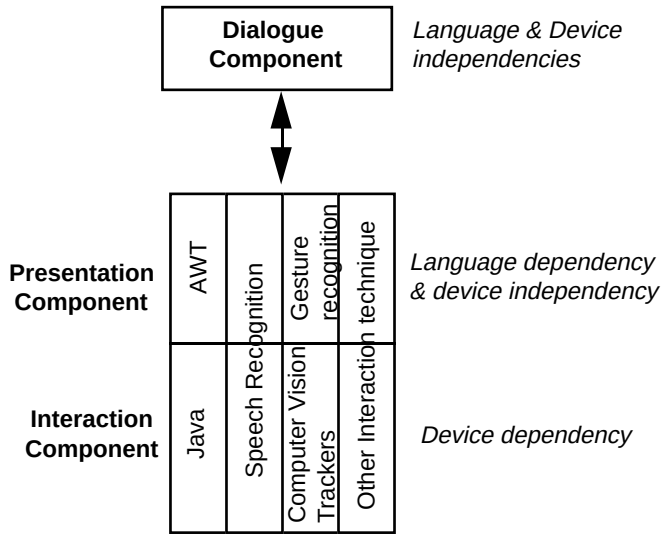

Fig. 2. Interaction Interpreters in the Presentation and Interaction Components.

4.2 PAC* and PAC

In the PAC style, the components are three facet agents interacting through event-based connectors. The facets (Presentation, Abstraction, Control) are used to express different but complementary and strongly coupled computational perspectives of the same functionality. As shown in Figure 3, no agent Abstraction is authorized to communicate directly with its corresponding Presentation and vice versa. Dependencies of any sort are conveyed via Controls. Controls serve as the glue mechanism to express coordination as well as data model transformations between the abstract and concrete perspectives. In addition, the flow of information between agents transits through Controls in a hierarchical way.

In PAC*, we reuse the heuristic rules developed for PAC-Amodeus to populate the Dialogue Component with PAC agents. Each of these rules proposes a configuration of agents, or pattern, that fits a particular situation. The complete set of PAC-Amodeus rules can be found in [20, 21]. Basically, these rules are driven by the external specifications of the user interface. Every interaction space such as a window,

a drawing area, a menubar, etc., is modelled as an agent. Views opened in cascade are related by a hierarchical link. Consistency between multiple views of the same concept is maintained through a common parent agent. Distributed user's actions over multiple agents, such as a palette and a drawing area, are synthesized into a meaningful command using a cement agent.

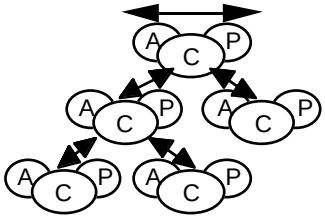

Fig. 3. In the PAC style, PAC components are configured hierarchically using event-based connectors. Arrows illustrate the information flow (vertical flow between agents, horizontal flow between the facets of an agent).

Agents, such as menus and buttons, that are implemented by the Presentation or Interaction levels are pruned from the hierarchical systematic decomposition: from first class agents, they become Presentation-level reusable code referenced in the Presentation facet of the parent agent from where they are pruned.

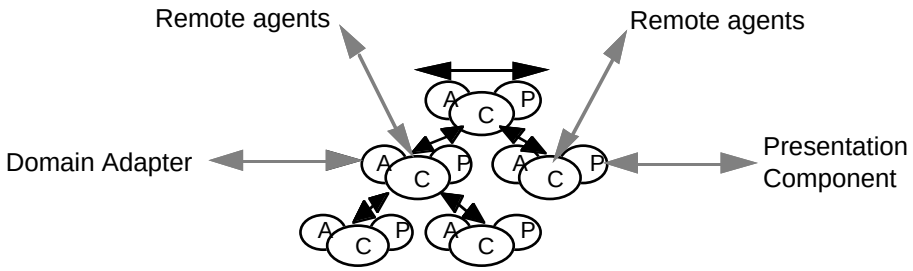

Fig. 4. A Dialogue Component in PAC* and its connectors. Dark arrows denote connectors within the Dialogue Component. Dimmed arrows represent connectors with other components.

In PAC*, just like for PAC-Amodeus, a Presentation facet maintains a data model for rendering the information maintained in its related Abstraction facet. This data model is used to feed into and receive information from the local Presentation Component. An Abstraction facet maintains a data model of domain-specific information exchanged with the Functional Core through the Domain Adapter.

Single-user events whose effects are private to a particular user are processed locally within the Interaction-Presentation-Dialogue components. Within the Dialogue Component, they are processed by local agents that may not have corresponding peers on remote workstations. Multi-user events, such as a user entering or leaving a conference, must be notified to remote sites. Dewans' model defines different forms of coupling between the layers. Within the Dialogue Component layer, coupling in PAC* is performed at the agent level. Figure 4 summarizes the possible connections

within the Dialogue Component and with outside the Dialogue Component. CoMedi is used next to illustrate the architectural possibilities.

As shown in Figures 5 and 6, CoMedi uses a fisheye porthole to support communication and awareness in a mediaspace. A slot in the porthole corresponds either to a remote user or to a group of users. An individual slot displays personal information about the corresponding remote user (e.g., level of availability, absence/presence, video image published through a filter [8]). When opening a collective slot, the current porthole is replaced with the porthole of the corresponding group. In a dedicated area of the control panel (see bottom left of Figures 5 and 6), users can check the image they export about themselves (cf. the reflexivity principle [27]). When an audio/video connection is established with a remote user, a dedicated video image is opened dynamically.

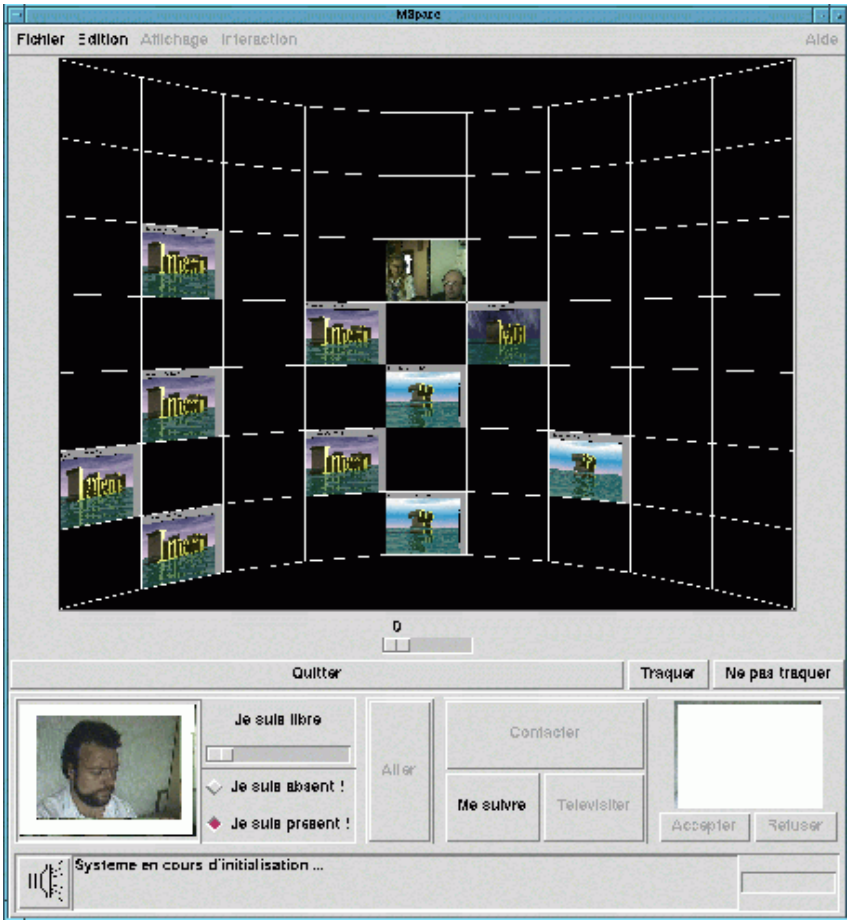

Fig. 5. Screen dump of CoMedi in neutral position: all of the slots of the porthole are of equal size.

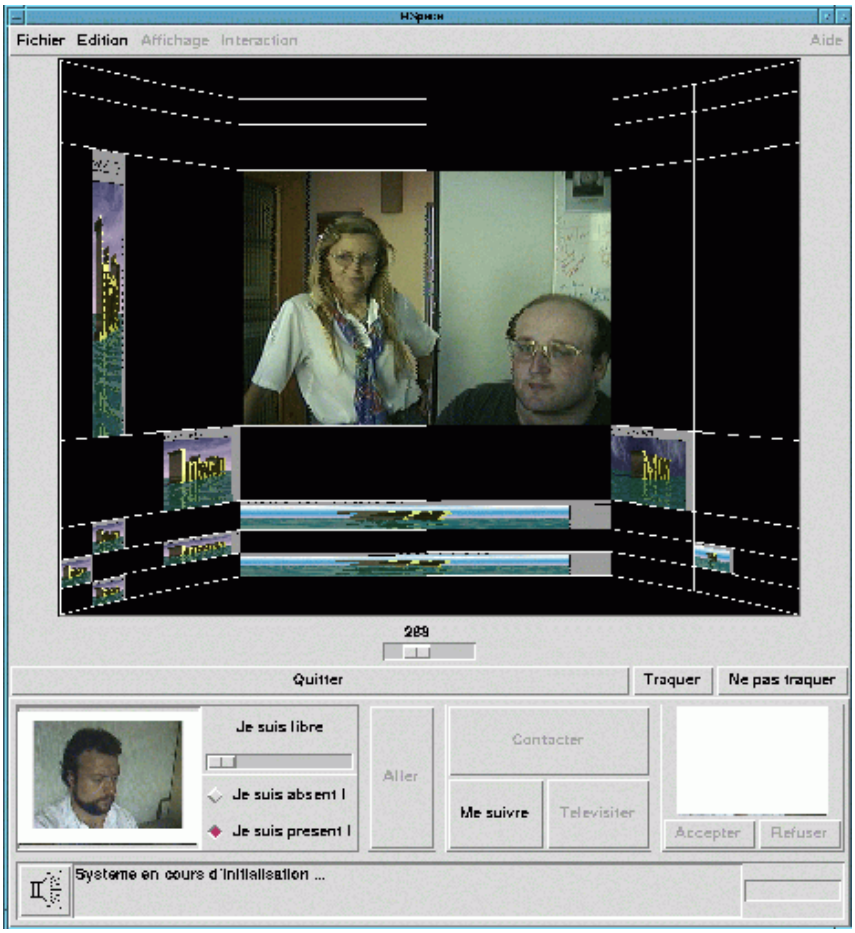

Fig. 6. Screen dump of CoMedi when focus is located on a particular slot

Figure 7 shows the refinement of the Dialogue Component of CoMedi. The Top-Level agent is in charge of the overall control of the dialogue. In particular, its Abstraction facet is in relation with the data base of users. (This data base is an active data structure implemented as a GroupKit environment [26].) Its Presentation corresponds to the overall menubar of the mediaspace.

The Porthole agent handles the local interaction with the porthole. Its Abstraction maintains the identity of the group in the current focus of attention of the local user. Its Presentation is a geometry manager that performs fisheye deformations of the porthole and maps the bitmap images provided by its siblings onto quadrilateral slots of varying sizes (the fisheye follows the mouse location as the user moves the mouse in the porthole).

Every remote user (or group of users) who belongs to the current porthole is represented by a Slot agent. Its Abstraction models the personal data of the remote

user/group to be rendered in the slot while its Presentation produces a bitmap based on the knowledge of the surface available for rendering.

The Vphone agent models the dedicated open audio/video window for Vphone connections.

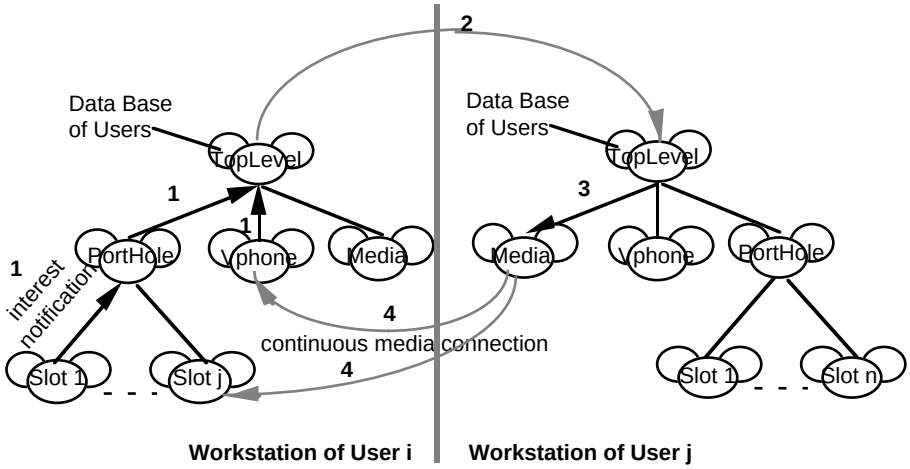

Fig. 7. The Dialogue Component of CoMedi.

The Media agent is in charge of acquiring and filtering the audio and video image of the local user. This filtered image is rendered locally in the Presentation facet of the Media agent; it is sent to every remote Slot agent that represents the local user currently rendered on remote sites; and finally, it is transmitted to the distant Vphone agent when the local user is Vphoning that distant user. For doing so, the remote slot agents and the remote Vphone agent express their interest to their local hierarchy in opening/closing an audio/video stream with that particular user (1). The remote TopLevel sends the request to the appropriate distant peer TopLevel agent (2) which, in turn, triggers the Media agent of the particular user (3). Direct audio/video connections are then established/suppressed between the Media agent and the remote subscribers (4).

4.3 PAC* and the Clover model

The Clover model, based on Ellis'model [11], provides a generic overview of the functional coverage of groupware. As shown in Figure 8, a groupware system covers three domain specific functions: production, coordination and communication [4]:

- 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 primarily 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.

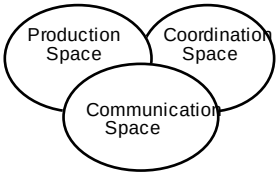

Fig. 8. The functional decomposition of groupware: the clover model.

PAC* uses the Clover model as a conceptual tool for refining the functional coverage of agents. In PAC*, agents can be functionally decomposed along two orthogonal axis: on one hand, the P, A, C functional breakdown and the clover decomposition of groupware on the other hand. In other words, the services that an agent supports for Production, Coordination, and Communication have their own Presentation, Abstraction and Control. This refinement of an agent functionality is shown in Figure 9 as a "Neapolitan PAC".

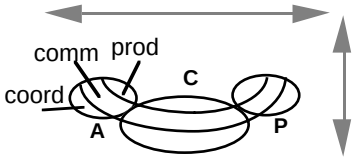

Fig. 9. A PAC* agent as a bundled "Neapolitan PAC". Arrows show horizontal communication between the A, P, C as well as vertical communication between the Production, Coordination, and Communication views.

In turn, a Neapolitan PAC* agent can be decomposed as a cluster of three PAC agents, each agent being dedicated to one class of the functional clover. One can refer to [4] for a detailed description of this refinement illustrated with an extended version of the SASSE multi-user editor [3].

.5 Conclusion

In this article we have demonstrated the motivation for PAC-ing the user interface of an interactive system ranging from single to multi-user systems. The PAC family conceptual models provide a sound basis for devising the right components of an interactive system with specific focus on Dialogue Components.

Although connectors within Dialogue Components are event-based, the nature of the connectors to be used between other components is underspecified. Experience shows that designing the connectors right is more difficult than getting the right components. We plan future work in this direction.

Another weakness of the PAC family models is the absence of formalization such as [10, 24]. Another missing element is a cookbook that would help practitioners to translate a PAC-based conceptual architecture into a motivated implemented architecture. This reflection is on our research agenda as well.

6 Acknowledgment

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

7 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. Arnold, K and Gosling, J.: The Java Programming Language, Addison Wesley, 1996.

3. 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).

4. Calvary, G., Coutaz, J., and Nigay, L: From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW, Proceedings of CHI 97, ACM publ., pp. 242-249.

5. Coutaz, J.: PAC, an Object Oriented Model for Dialog Design, in Proceedings Interact'87 (North Holland, 1987), 431-436.

6. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. and Young, R.: Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties, Proceedings of the INTERACT'95 conference, S. A. Arnesen & D. Gilmore Eds., Chapman&Hall Publ., Lillehammer, Norway, June 1995, pp. 115-120.

7. 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, pp. 106-111.

8. Coutaz, J., Crowley, J. and Bérard, F. Eigen space Coding as a Means to Support Privacy in Computer Mediated Communication. Proceedings of INTERACT'97, Sydney, to appear.

9. Dewan, P.: Multiuser Architectures, in Proceedings EHCI'95, Working Conference on Engineering Computer Human Interaction.

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. Garlan, D., Allen, R., and Ockerbloom, J.: Architectural Mismatch or Why it's hard to build systems out of existing parts, ICSE'95, ACM, 1995, pp. 179-185.

15. Gundavaram, S.: CGI Programming on the World Wide Web, O'Reilly & Associates, Inc., 1996.

16. Hill, R.: The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications, in proceedings CHI'92 (NewYork, 1992), ACM, 335-342.

17. 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.

18. Kramer, J.: Exoskeletal Software-Making Structure Explicit. Software Architectures, Schloss Dagstuhl Seminar, Garlan, D. Paulish, F. and Tichy, W. eds., Dagstuhl Seminar Report 106, 1995, pp.23-23.

19. McCall, J.: Factors in Software Quality, General Electric Ed., 1977.

20. Nigay, L. and Coutaz, J.: Building User Interfaces: Organizing Software Agents. In Proceedings ESPRIT'91 Conference, 1991, 707-719.

21. Nigay, L.: Conception et modélisation logicielles des systèmes interactifs: application aux interfaces multimodales, Thèse de doctorat de l'UJF, 1994.

22. Nigay, L. and Coutaz, J.: A Generic Platform for Addressing the Multimodal Challenge, CHI'95, ACM New York, Denver, May 1995, pp. 98-105.

23. Ousterhout, J.,K.: Tcl and the Tk Toolkit. Addison Wesley Professional Computing Series, 1994.

24. 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.

25. Pfaff G.E. et al.: User Interface Management Systems, Pfaff, G.E. ed., Eurographics Seminars, Springer Verlag, 1985.

26. 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

27. 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.

28. Shaw, M. and Garlan, G.: Software Architecture, Perspectives on an Emerging Discipline, Prentice Hall, 1996.

29. Valdez, J.: XVT, a Virtual Toolkit, Byte ,14(3), 1989.

30. VITESSE, Visualization and Interaction Techniques to Enhance Superscalar Search Engines. http://iihm.imag.fr/vernier/Vitesse.html.