Frameworks and patterns for synchronous groupware : AMF-C approach

F. Tarpin-Bernard, B.T. David, P. Primet
GRACIMP - ICTT, Ecole Centrale de Lyon, France
{Franck.Tarpin-Bernard, Bertrand.David, Pascale.Primet}@ec-lyon.fr

Abstract: Frameworks and design patterns are emerging technologies in software engineering. They increase software quality in terms of reusability, modularity and extensibility. Synchronous groupware can benefit of these new technologies. This article describes AMF-C, a multiagent model which structures each agent with a various number of facets, and two associated frameworks. Indeed, a cooperative application can use either a fragmented framework (facets are dispatched into the network) or a replicated one (each agent is totally replicated). Design patterns are identified for the definition and the interconnection of facets. In this last case, an expressive graphical formalism is used to wire control components. The design and implementation tasks are largely reduced and mainly rely on a good choice and combination of patterns. Finally, we introduce the associated tools and methodology that holds great promise in addressing the design issues.

Key words: Framework, design pattern, synchronous groupware, CSCW, AMF-C

1. INTRODUCTION

"Object-oriented application frameworks are a promising technology for reifying proven software designs and implementations in order to reduce the cost and improve the quality of software" (Fayad, 1997). Indeed, they enhance modularity by encapsulating volatile implementation details behind stable interfaces which reduce the effort required to understand and maintain existing software. Related to the framework technology, the design patterns have recently emerged in software engineering (Gamma, 1995). These patterns are supposed to describe recurring

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35349-4_22 solutions that have stood the test of time. A single framework usually contains many patterns, so these patterns are smaller than frameworks. Therefore, they are also more abstract. They are the micro-architectural elements of frameworks (Johnson, 1997). But for many authors, since some frameworks have been implemented several times, they represent a kind of pattern, too. For instance, Model/View/Controller is a user-interface framework that is described as a pattern in Bushmann & al. (1996), whereas Johnson considers that it can be decomposed into three major design patterns and several less important ones, referring to Gamma's work. Actually, both notions are complementary and their importance in software engineering is increasing regularly.

Computer supported cooperative work is also a recent field of investigation. Many models, tools and interaction patterns have been developed for experimental groupware. Few of them are becoming specialised frameworks. For instance, the National Center for Supercomputing Applications built *Habanero* (NCSA, 1996), a framework for sharing Java objects with colleagues distributed around the Internet. *TCL-TK DP* (see Smith, 1996) can also be considered as a cooperative framework. It is used by Roseman's team to implement the new version of *Groupkit* (Roseman, 1993).

As the number of CSCW experiments and observational studies is increasing, new sociological and psychological consequences of this new way of working are identified. One of the most important conclusions about these systems is that, most of the time, they are well adapted for one kind of cooperative work but can not be applied in all situations (meetings, collaborative design, teletraining, etc.). For all these reasons, we consider that groupware, and especially synchronous groupware, should provide a very wide range of patterns of interaction. But more importantly, they should provide services which allow the user (or the leader of a group) to switch at run-time from one pattern to another.

Considering these different issues, our purpose is to help design and development of flexible groupware, building cooperative frameworks and associated design patterns. This paper describes AMF-C, an architectural model which defines two cooperative frameworks, and some of the main patterns which have been identified. It concludes with the related design methodology and tools.

2. FRAMEWORKS FOR GROUPWARE

Referencing to well known works such as (Rodden, 1991) or (Ellis, 91), we have identified different forms of control (Tarpin-Bernard, 1998), among these, the most relevant are:

- **Interaction control** : management of the relations between user actions and internal data changes.
- **Notification control** : management of the notifications of actions from or to the group.
- **Access control** on data and processings : management of rights and duties of users in relation to their roles in the group.

- **Concurrency control** : coordination activity of concurrent access to shared resources in multi-user systems.

As groupware engineering can not be done from scratch, it is necessary to identify different levels of development. These four controls can be dispatched into three functional layers (see figure 1), corresponding to the three main actors involved in groupware : users, groups and computers.

User :	Collaborative application level
Group :	Groupware infrastructure
System :	Distributed system level

Figure 1. The three functional layers of a groupware environment

The first layer corresponds to the **collaborative applications** level. It contains all the cooperative software used by the users. This level is definitely user-centred, which means that it manages interaction control and proposes interfaces for notification and access controls. It uses multiusers services provided by a second layer called the **groupware infrastructure**. This layer contains all the common elements of group activities and acts as an operating system dedicated to groups. It supports collaborative work managing sessions, users and groups ; provides generic cooperative tools (e.g.: telepointer) and is responsible for concurrency control. It also implements notification protocols and provides access control mechanisms. In many groupware, these tasks are assumed by toolkits. It is a generic layer between applications and **distributed system** which constitutes the third level of our model. This last layer is essentially in charge of message multicast and consistency control. Usually, it is a computer-centred layer that provides transparent mechanisms for communication and synchronisation of distributed components which misfit with CSCW aims but which are very useful.

In the next sections, we will only develop the collaborative application level and especially our AMF-C model (Tarpin-Bernard, 1997b).

2.1 AMF : a framework for single-user software

Architectural models for groupware have to combine the knowledge of models developed for single-user applications and the constraints introduced by cooperative work. For many years, HCI community has been very interested in designing models for interactive software. One of the most important class of such models is the multiagent one. These models organise an interactive system as a set of agents that collaborate to support the dialogue between men and computers. Most of these agents are based on three components (facets) mapped on the HCI paradigm (presentation to the user, functional kernel, and interaction control). But, these models present two main disadvantages :

1. They define very large facets which mix different thematic functions.
2. They do not provide powerful mechanism to express interaction control.

To bring some solutions to these shortages, we chose to develop a multiagent model called AMF (Ouadou, 1994).

Indeed, to solve the first problem, AMF organises each agent in an appropriated number of facets. These facets can be similar to the classical components of PAC model (Coutaz, 1990) or MVC (Krasner, 1988). They can also either come from a finer split of *control* components, or from the identification of new characteristics of agents (e.g.: management of the user model), or from the duplication of classical facets (several *presentation* facets corresponding to different views). For instance, we can identify the following facets: *presentation* (I/O relations with the user), *abstraction* (logical data - functional kernel), *evaluation* (capture of the user's actions), *help* (contextual and on-line helps linked to a user model) or *user model* (information for adaptive interface). In the multiuser version of AMF we will present other facets related to the cooperative work requirements.

Finally, to solve the second problem, AMF expresses interaction control with two kinds of components:

1. Each facet presents several *communication ports* (allowing input, output or both). These ports avoid to having a permanent binding between an abstraction (a port) and its implementation (a function). Moreover, it is possible to implement the body of the functions in various languages.
2. The *Control* "facet" is an abstract facet mainly defined by *control administrators.* A control administrator has three roles:
   - To *connect,* managing logical relations between the communication ports (sources and targets) that are connected to it;
   - To *translate,* transforming the messages which come from the source ports in understandable messages for target ports;
   - To express *behaviour,* and so control strategies, using different rules of activation between a source port (A) and a target port (B). We have identified several administrators, such as : simple (if A then B), sequence (if A1, next A2, next ... An then B), conjunctive (if A1 and A2 and ... An then B)...

An example of interaction control in a single-user application.

Using the AMF concepts, it is possible to model an interaction control in a single-user application. In the simplest case, when only one agent is implicated, two simple administrators (A_1 & A_2) generally manage the relations between an action starting from the *Presentation* facet and the associated command defined in the *Abstraction* facet (see figure 2).

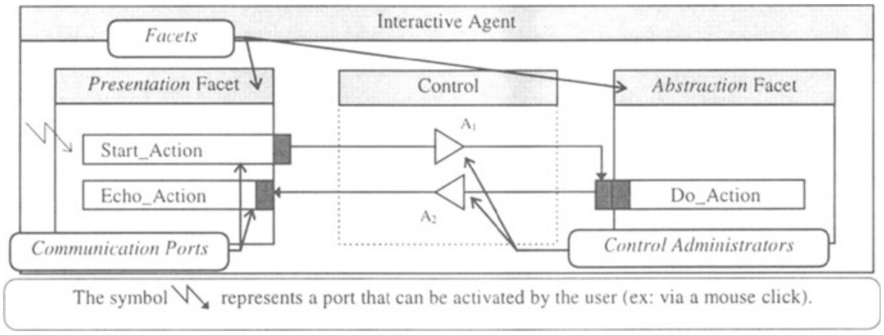


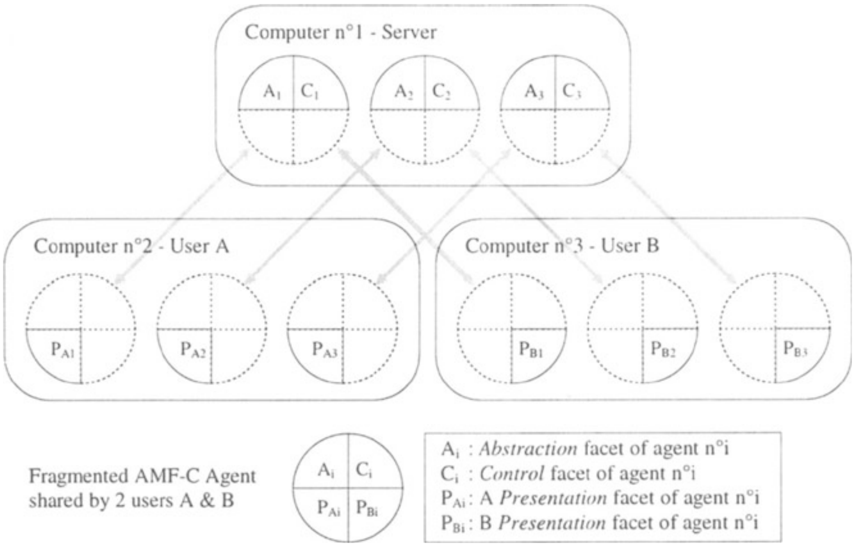
Figure 2. An interaction on a single-user agent modelled with AMF

In a multi-user context, an application must be able to notify each action of one user to the other members of his group, and each agent must be able to reproduce the actions of remote users. To solve this problem, we created AMF-C a cooperative extension of AMF (Tarpin-Bernard, 1997a). This model can adopt a fragmented form when shared agents are split and their facets distributed, and a replicated form when each agent has a representative on each workstation.

2.2 A fragmented framework for groupware

Analysing distributed systems studies, we have found the original concept of fragmented object (Gourhant, 1994). The methods and data of a fragmented object are distributed on the network and "transparent" mechanisms let it look like a classical object in a single computer. Applying fragmentation to AMF model offers an interesting approach for modelling CSCW applications. Indeed, their facets define a natural boundary for fragmentation. Thus, we can study the distribution of the facets into the network. According to the desired architecture, we can distribute *presentation*, *control* or *abstraction* facets.

The figure 3 presents a centralised architecture with three shared agents manipulated by two users. Each agent is defined by four facets : the *abstraction* and *control* facets, and two *presentation* facets corresponding to specific views of each user. In this context, each *presentation* facet can be adapted to the role of each user ($P_{Ai} \neq P_{Bi}$). It is the *control* facet which is in charge of the propagation of input/output events from or to the different facets, and especially between multiple *presentation* facets.



Computer n°1 - Server
A_1 C_1
A_2 C_2
A_3 C_3
Computer n°2 - User A
P_{A1}
P_{A2}
P_{A3}
Computer n°3 - User B
P_{B1}
P_{B2}
P_{B3}
Fragmented AMF-C Agent shared by 2 users A & B
A_i C_i P_{Ai} P_{Bi}
A_i : *Abstraction* facet of agent n°i
C_i : *Control* facet of agent n°i
P_{Ai} : A *Presentation* facet of agent n°i
P_{Bi} : B *Presentation* facet of agent n°i

Figure 3. An AMF-C fragmentation - centralised version

If we try to model an elementary interaction (e.g.: a button triggers an action on an agent), we can consider a situation in which a first user is responsible of the agent, whereas a second user can just interact with its presentation. In this case, we can imagine that the agent is mainly located on the first user's workstation (Figure 4). To assume concurrency control and maintain the consistency of the shared agent, it is necessary to define new types of administrators. In the example given on the figure 4, we have built a lock administrator which filters the access to the agent.

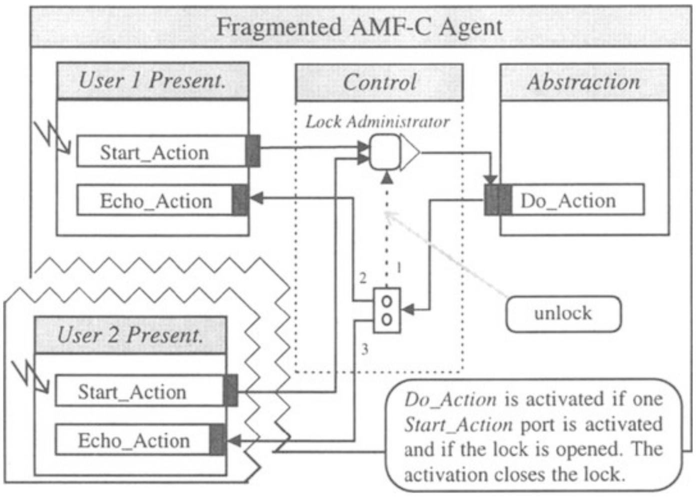


Figure 4. An example of elementary interaction on a fragmented AMF-C agent

The dynamicity property of AMF-C agents allows to formalise the adaptation of each agent to the current user's role. Indeed, the number and the form of facets is not static, any change of role can lead to substitute a facet, and especially a *presentation* one.

The fragmented AMF-C framework is well adapted to represent hybrid architecture in which some facets are centralised whereas others are replicated. Moreover, the use of a distributed object-oriented language can really ease the implementation of such a model. Indeed, in our first implementation of AMF, with C++, each facet is an object. The activation of a communication port leads to the invocation of a method of these objects. In a distributed context, this corresponds to a remote method invocation as defined in CORBA (Siegel, 1996) or Java-RMI.

However, we can notice that, to introduce more flexibility in notification control and so propose WYSIWIS relaxations, we need to multiply the number of *control* and *abstraction* facets. Indeed, if we want to process remote actions differently than local ones, we need some new *control* facets. On the other hand, several *abstraction* facets are necessary if we want to authorise users to work on their own data and let do some versioning. If we insist in this way of facet personalisation, agents fragmentation becomes an unadapted paradigm. As a consequence, looking for a maximum flexibility implies to replicate each agent and so to choose the replicated AMF-C framework.

2.3 A replicated framework for groupware

Replication is based on both notions of **reference agent** and **local agent**. When a reference agent is shared by *n* participants, *n* local representatives are distributed on the local workstations. The local agents of a same reference agent are called **brother agents**. These local agents support the manipulations of the users. The form and the content of each local agent depend on its owner's characteristics expressed in terms of roles and viewpoints. To define work contexts, which means memorise agents states, the reference agent notion is particularly interesting. Actually, reference agents can be real or virtual (figure 5).

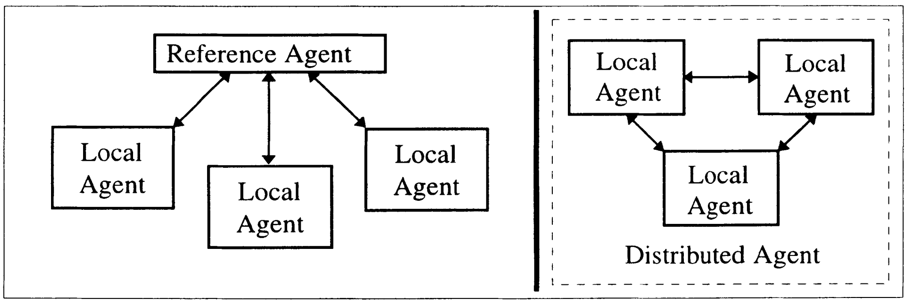


Figure 5. Two visions of reference agents : centralised or distributed

In the first case, reference agents can be localised on a server, with all the advantages (simplicity, regularity, etc.) and disadvantages (rigidity, bottleneck, etc.)

of such a situation. In the second case, each reference agent is virtual, which means that it is defined by the whole set of local agents. This approach presents other advantages (more interactivity, best fault tolerance) and disadvantages (complexity, etc.). As an intermediate solution, the reference agent can be one of the local agent and its localisation can be static or dynamic. In the static case, it is always in the same place, whereas in the dynamic case, it can be situated on the workstation of the group leader or on the workstation of its creator. In all these cases, each action (creation, modification and deletion) performed on the local agent should be notified to the other brother agents.

2.3.1 An example of replicated AMF-C framework

In our laboratory, we have experimented this framework using the ECooP groupware infrastructure (Primet, 1996a). In order to introduce a maximum flexibility in the four control presented in the beginning of section 2, we consider that four steps are relevant in a "group interaction": selection (OS : Object Selection), validation (AV : Action Validation), execution (AE : Action Ending) and unselection (OF : Object Freeing). Several sequences of actions can happen between the selection - unselection phase (figure 6a).

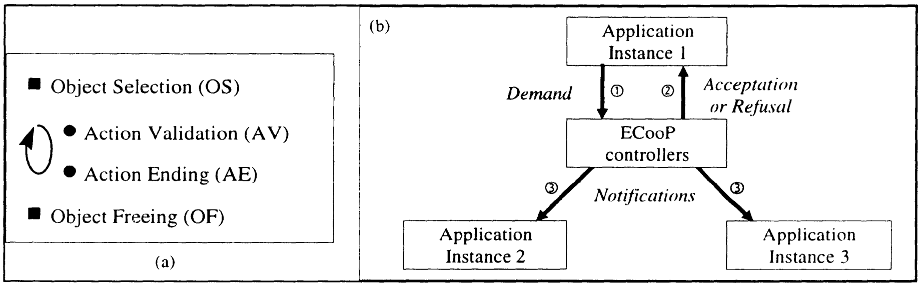


Figure 6. The four steps of the control dialogue (a) and the NCP protocol (b)

A specific **control and notification protocol** and a set of associated generic functions have been defined in order to ease the dialogue between a cooperative application and the groupware infrastructure (Primet, 1996b) and to provide flexible concurrency control mechanisms. It relies on four types of messages - D: Demand, A: Acceptation, R: Refusal and N: Notification - (figure 6a).

2.3.2 Concurrency control flexibility

At each phase of a "group interaction", a demand is systematically sent to the ECooP local controller so that it is always informed of the application state. Depending on the chosen concurrency policy and the initial control parameters, the controller answers immediately without any control or submits the request to the ECooP decision component before answering to the application. For instance, in a pessimistic policy with an earliest control (since object selection), the control is performed with the reception of an "Object Selection Demand". The decision component accepts or refuses and the local controller transmits this "collective" decision to the application. Then, the other messages are systematically accepted and notified as the lock assumes that there will be no problems with these operations. On the other hand, in an optimistic policy with a latest control (at the end of processing), the "Object Selection" and "Action Validation" demands are always accepted. The real control is only done at the end of the action execution. If a conflict occurs, its solving is related to a collective decision and the application must undo the action. A notification of this undo action is also sent to the remote applications.

In this context, whatever the decision component is (the community of local controllers or a central controller), the application interface does not have to change. Only behaviour of controller agents depends on the policy.

The replicated version of the AMF-C model fits very well with ECooP. Indeed, to implement flexible concurrency control, we first need to define specific administrators able to dialogue with local controller using functions of the ECooP API and second to build a new facet, called *Distant*, which receives the notifications of remote actions. Figure 7 presents the schematic representation the four administrators which realise the four phases of the dialogue (a) and two additional administrators (b) which can be used to implement direct manipulations (Object Selection and Action Validation can be simultaneous).

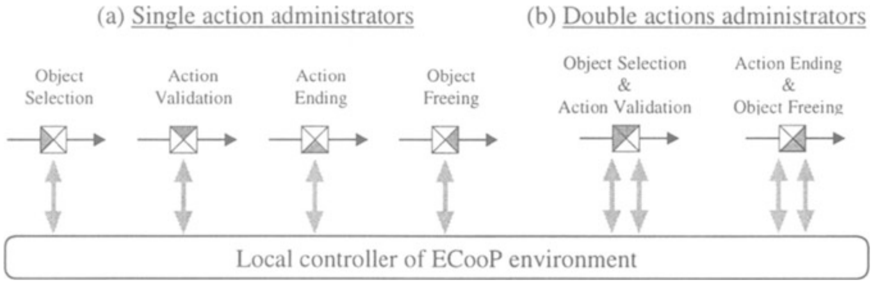

(a) Single action administrators
(b) Double actions administrators
Object Selection
Action Validation
Action Ending
Object Freeing
Object Selection & Action Validation
Action Ending & Object Freeing
Local controller of ECooP environment

Figure 7. The cooperative administrators of AMF-C

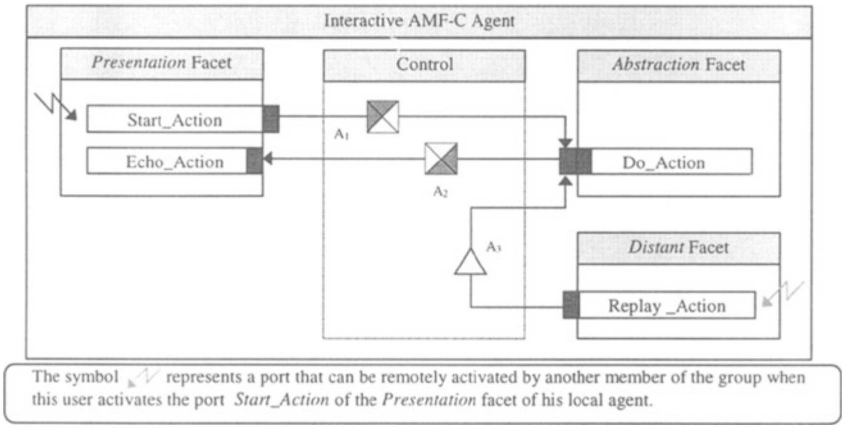
3. DESIGN PATTERNS

Considering both AMF-C frameworks, we can imagine various design patterns related to the choice of thematic facets or to the choice of control mechanisms. In this section, we develop the patterns associated to the replicated framework.

The identification of specialised CSCW facets can lead to various solutions. For instance the PAC* model (Calgary, 1997) maps the three common functional spaces of groupware (production, communication and coordination) on the structure of PAC agents. As a consequence, the authors propose several patterns dealing with various combinations of dispatching. For instance, each component (Presentation, Abstraction and Control) can be sliced into three parts corresponding to the three spaces. Another pattern dedicates PAC agents to treat the production, communication and coordination functions independently. A first adaptation can leads us to define communication and coordination facets whereas production facets can be assimilated to abstraction facets. A large presentation facet or three smaller ones then will also be defined. In practice, we met some difficulties to split agents this way because considering fine grain agents we found that they are often dedicated to one main space so that the model lost its interest.

So, in addition to the *Distant* facet which receives all the remote notifications, we introduce a second one, called *Access,* which is in charge of the adaptation of presentations according to the users' roles. It activates and deactivates the interactive control objects of the user interface according to his rights. At least we define a *Private* facet which deals with the choice of group retroaction and notification control (see next section). The AMF-C model is definitely dynamic and allows, at each instant, to modify, create or delete some agents, facets or administrators. The *Private* facet lets users change structurally the agent via an adapted interface.

Using the six administrators that we have presented in §2.3 and referring to the standard interaction pattern presented figure 1, we can define a first pattern of cooperative interaction (Figure 8). When the message sent by *Start_Action* crosses the A_1 administrator, all the remote agents receive from ECooP a message which activates the *Replay*_Action port of the *Distant* facet, so that the action is replayed on each replica of the agent.



The symbol ↙╱ represents a port that can be remotely activated by another member of the group when this user activates the port *Start_Action* of the *Presentation* facet of his local agent.

Figure 8. A first interaction pattern on a shared agent modelled with AMF-C

It is also possible to define a second pattern of interaction in which selection and unselection phases are clearly distinct from the action phases (see figure 9). This pattern allows users to see the objects which are locked (locally or remotely).

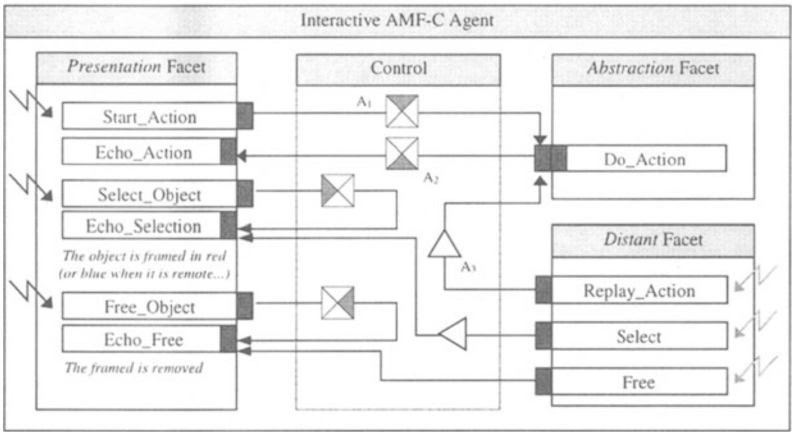


Figure 9. A second interaction pattern on a shared agent modelled with AMF-C

Considering all the specific facets presented before, we can propose a generic interactive and cooperative AMF-C agent (figure 10). This pattern only shows one direct interaction (once it is finished, the object is freed). Of course, for each action, a real agent contains one of the previous patterns.

This last figure also details the structure of the *Private* facet. As we introduced it in the previous section, relaxation of WYSIWIS can be done with AMF-C according to **several strategies**.

The first one consists in modifying the administrators linked to the *Distant* facet in order to change its connection with the *Presentation* facet or even with the *Abstraction* facet. In the second one, we can modify the implementation of the input port *Echo_Action* of the *Presentation* facet for it to have a different behaviour depending on the source of the activation message. Finally, in some cases, we can completely disconnect some communication ports (e.g.: do not propagate some scrolling actions of other participants). For instance, to disconnect a user from the others, one solution is to not notify his actions of the other members. To make such a change, it is necessary to modify the administrators which are linked to *Presentation* facets. These three strategies are represented by the three generic ports of the *Private* facet which are presented on the figure : *Change_Propagation*, *Change_Echo* and *Change_Updating*. As a consequence, the *Private* facet has a structural knowledge of the agent.

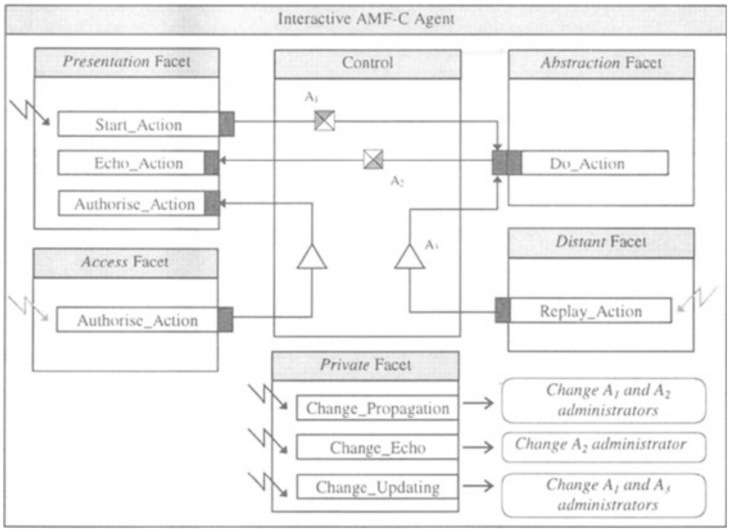


Figure 10. General structure of an interactive AMF-C agent

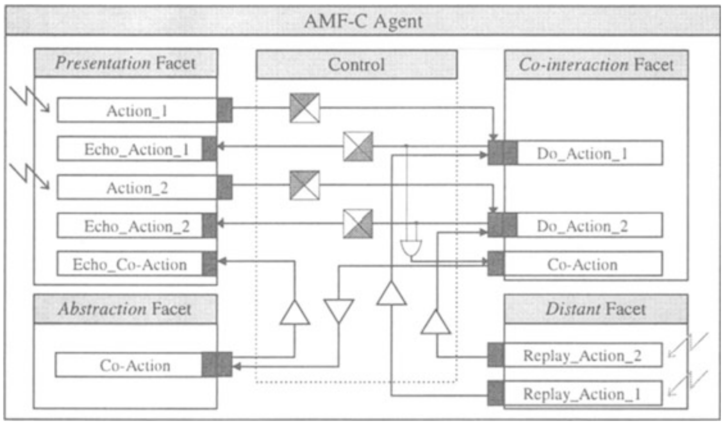


Figure 11. A design pattern of co-interaction on an AMF-C agent

Combining the definition of new facets and the use of new control mechanisms, designers can imagine new kinds of interaction. As a short example, we propose in figure 11, a pattern which models a co-interaction. We call co-interaction, a complex interaction between two or more users where each of them execute a part of the action (e.g. : two users should turn 2 different keys to open a secure door). To implement a co-interaction, we define a new facet which monitors the elementary interactions and triggers the co-interaction when all requirements are achieved (completion, timing, etc.).

4. METHODOLOGY AND TOOLS

In this paper, we have focused our attention on the presentation of AMF-C frameworks for groupware, and some of their associated design patterns. However, we did not forget, that such a work is really useful only if a methodology is also proposed and if design tools are provided.

Currently, thanks to our first design and development experiments we have started to define some rules that should be used by a comprehensive methodology. The first one deals with the choice of the AMF-C framework. According to us, the main criteria of selection is the degree of autonomy and personalisation that is looked for. Indeed, if the wanted application respects simple rules of coordination (e.g. only one kind of WYSIWIS is needed) and if the groups are homogeneous, the fragmented framework which is the simplest one is a good choice. In other cases, the replicated framework is the only one that lets you introduce a maximum of flexibility in the group processes.

Once the designer have chosen his framework, his task is now divided into two main phases :

1. Identify the agents and their services
2. Choose design patterns associated to each service

Indeed, as in other multiagent models it is often difficult to identify the agents of the application. On this particular point, AMF-C does not introduce specifities, so that all the common methods can be used. However, the situation changes concerning the choice of structural patterns for agents. It is necessary to choose what facets should be defined and how to dispatch the services into these facets. At least, designers have to choose a pattern of interaction for each service. Each time that it is meaningful to define new facet in terms of reusability and modularity we advise to do it.

Of course, because of our flexibility principle, the choice of these patterns is not static. Initially, the designer chooses what pattern is the most relevant according to his point of view. But, whether the end-user wants to change some controls, the dynamicity of AMF-C lets him choose his own pattern.

Finally, as we mentioned it in the beginning of this section, the design pattern approach fits very well with the definition of associated tools. The builder tool which is under construction will propose a catalogue of design patterns to ease groupware design. The design and implementation tasks are largely reduced by the direct building of AMF-C diagrams. In the future, this tool will also be used for the definition of new patterns and for the dynamic evolution of cooperative applications.

5. CONCLUSION

Frameworks and design patterns are both notions that have recently emerged in object-oriented software engineering. They have been proposed to ease design of large and complex software thanks to their good properties in terms of reusability, modularity, extensibility are more generally quality improvement. Groupware are among the most complex software that should be designed. Because they involve groups of users in work processes, they must combine all the advanced technologies studied in single-user situations but also take in account dynamics of work sessions and sociological rules.

Our proposal tries to cover a wide range of problems from groupware design to dynamic adaptation of cooperative applications. Our key model is the multifaceted multiagent AMF model. Its cooperative extension AMF-C leads us to propose two frameworks for groupware based on both paradigms : fragmentation and replication. The graphical formalism of AMF-C eases the building of various design patterns corresponding to several kind of problems.

Designers' tasks then consist of choosing these patterns in respect with these frameworks. The methodology and its associated tools are the keys elements for the success of this approach. Currently, we focus our efforts on the development of both elements continuing to implement and evaluate our own synchronous groupware. Today, we have already developed a kinematic diagram editor and a 3D scene builder that validates the concepts presented in this paper.

Finally, the AMF-C graphical notation mainly describes the organisation of multiagent applications in terms of structures and relationships. Referring to Kruchten's work (1995), we consider that it can be used to express logical views on a system. However, this notation only shows static views. It needs to be completed by scenarios in order to fully use the dynamic property of AMF-C.

6. REFERENCES

Coutaz J., Nigay L. (1997), From Single-User Architectural Design to PAC*: A Generic Software Architecture Model for CSCW, CHI'97, Atlanta, ACM Publ., 242-249.

Coutaz J. (1990) Architecture Models for interactive software: Failures and Trends, in G. Cockton (eds.): *Engineering for Human-Computer Interaction*, Elsevier Sc. Publ., 137-153.

Ellis C.A., Gibbs S.J. & Rein G.L., (1991), Groupware : some issues and experiences, Communication of ACM, Vol 34. n°1, 39-58.

Fayad M.E., Schmidt D.C. (1997), Object-Oriented Application Frameworks, *Communications of the ACM*, Oct., Vol 40. n°10, 32-38.

Gamma E., Helm R., Johnson R., Vlissides J. (1995), Design Patterns : Elements of Reusable Object-Oriented Software, Addison Wesley, Reading, MA.

Gourhant Y., Makpangou M., Le Narzul JP., Shapiro M. (1994), Fragmented objects for Distributed Abstractions, in Readings in Distributed Computing Systems, Eds Casavant & Singhal, IEEE Computer Society Press, 170-186.

Johnson R.E. (1997), Frameworks = Components + Patterns, *Communications of the ACM*, Oct., Vol 40. n°10, 39-42.

Krasner G.E., Pope S.T., (1988), A cookbook for using the model-view controller user interface paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, 1(3), 26-49.

Kruchten P. (1995), The 4+1 View Model of Architecture, *IEEE Software*, November, 12 (6), 42-50.

NCSA (1996), Habanero Home Page : http://www.ncsa.uiuc.edu/SDG/Software/Habanero/

Ouadou K. (1994), AMF : Un modèle d'architecture multi-agents multi-facettes pour Interfaces Homme-Machine et les outils associés, Ph D, Ecole Centrale de Lyon, France.

Primet P. (1996a), ECooP a flexible CSCW Environment, Technical Report, Ecole Centrale de Lyon, France

Primet P. (1996b), Contrôle de concurrence dans les collecticiels: mise en oeuvre de la flexibilité, Proceedings of CRAC'96, Paris, France

Rodden T. (1991), CSCW and Distributed Systems: the problem of Control, *Proceedings of the ECSCW '91*, Amsterdam, Kluwer Academic Press.

Roseman M. (1993), Tcl/Tk as a Basis for Groupware, *Proceedings of Tcl 93 Workshop*, University of Calgary, Alberta Canada,

Siegel J. (1996), CORBA - Fundamentals and Programming, John Wiley.

Smith B., Rowe L. A. (1996), An Introduction to Tcl-DP, Cornell University, http://www.cs.cornell.edu/Info/Projects/zeno/Tcl-DP/Tutorial/tutorial.html

Tarpin-Bernard F. (1997a), "Travail Coopératif Synchrone Assisté par Ordinateur : Approche AMF-C", Ph D, Ecole Centrale de Lyon, France.

Tarpin-Bernard F., David B.T. (1997b), AMF a new design pattern for complex interactive software ?, *Proceedings of International HCI'97*, San Francisco, in Design of Computing Systems, 21 B, Eds Elsevier, 351-354.

Tarpin-Bernard F., Primet P. (1998), *Flexibility in synchronous groupware*, paper submitted to the Journal of Computer Supported Collaborative Work, Kluwer Academic Publishers

7. BIOGRAPHY

Franck Tarpin-Bernard is an associate professor graduated in 1997 (PhD). He is also engineer of the Ecole Centrale de Lyon and has been working on CSCW and software engineering for four years. Bertrand David is professor and co-director of the GRACIMP laboratory. He works on HCI, CSCW, Concurrent Engineering and cooperative learning. Pascale Primet is an associate professor in computer science. She mainly studies groupware, distributed computing and high speed networks.

Discussion

Len Bass: How do you handle time constraints between distributed events?

Franck Tarpin-Bernard: Co-Action will manage the time constraints between two distributed events that must be combined to obtain a complete command.

Prasun Dewan: What is the difference between an administrator and a facet?

Franck Tarpin-Bernard: A facet is a set of communication ports. Administrators are sub-components. They can be considered as a special kind of facet. The control facet includes all the administrators.

Prasun Dewan: You showed us how the collaborative behaviour can be changed by changing the patterns. At what time is the pattern bound , at application creation time or at runtime? If it is at runtime, then users could change dynamically from, say synchronous to asynchronous coupling.

Franck Tarpin-Bernard: Currently the set of patterns must be defined at application creation time by the designer but users can change dynamically from one pattern to another at run-time.

Prasun Dewan: But users do not think in terms of facets.

Franck Tarpin-Bernard: Yes, but the Presentation facets are in charge of presenting the interaction patterns to the user and triggering the structural adaptations of agents. So the application could adapt its structure automatically to the user's needs.

Helmut Stiegler: Your focus is on synchronous co-operation. Groupware in general has a broad scope, including workflow systems, which are asynchronous. Do you know already how to include asynchronous collaboration?

Franck Tarpin-Bernard: We thought about it. Maybe we only need a new facet. For each specific problem, a new kind of facet is created. We need new facets and maybe new administrators to handle this problem.

Nick Graham: How many controller styles (i.e., concurrency, control policies) have you implemented?

Franck Tarpin-Bernard: We developed various pessimistic approaches. About optimistic approaches, we only tried one: Undo conflicting actions.

Nick Graham: The dialogue protocol seems to restrict us to relatively simple object-function dialogues. Is this correct?

Franck Tarpin-Bernard: Not at all. If the designer needs to manage complex objects such as sets of objects he will consider them as a new class of agent. Moreover, like in Object Oriented languages, we have defined the notion of inheritance for agents. So you can build a large hierarchy of agents.

Prasun Dewan: Do you provide a general undo function?

Franck Tarpin-Bernard: We tried to create a new facet Undo similar to the coordinator. We are not sure that we can reuse undoing functions in different CSCW systems.

Joelle Coutaz: Can the model cope with a mixture of fragmented and replicated agents?

Franck Tarpin-Bernard: Conceptually it is quite feasible but in practice the implementation tools and the infrastructure you use may lead to only one of the policies.

Joelle Coutaz: How does AMF-C cope with heterogeneity of styles?

Franck Tarpin-Bernard: Just like in PAC-Amodeus. The Dialogue Controller is expressed as AMF-C agents whose facets call external functions.