Patterns in Collaborative System Design, Development and Use

B. David, O. Delotte, R. Chalon, F. Tarpin-Bernard, K. Saikali

Laboratoire ICTT, Ecole Centrale de Lyon – INSA de Lyon
36, avenue Guy de Collongue, 69134 ECULLY, France
Bertrand.David@ec-lyon.fr

Abstract: Design, Development and Use of collaborative systems is a very interesting and complex case of cross-pollination between usability and software engineering. User Centered Design (UCD) is a necessity for this kind of systems allowing to different actor to work together in a cooperative environment. Classical HCI is augmented by the HHI (human- human interaction) and by the need of awareness (shared conscience of other actors and of their actions on the common work). The complexity of CSCW tools is growing from asynchronous to synchronous, from individual applications to cooperative systems and from classical workstation based static environment to Capillary CSCW (David et al., 2003a), i.e. mobile environment with handheld devices. We proposed an environment, architecture, models, tools and overall methodology for design, development and use of collaborative systems (David et al., 2003b). In this position paper we propose to examine this environment from UCD and pattern point of view. This is a special patterns-oriented walkthrough of our work..

Key-words: Patterns, CSCW, model-based design

1 Introduction

Our approach of patterns is the following. In relation with Alexander (1977), Gamma (1995), Borchers (2001) and Seffah (2003), we adopt more comprehensive definition which is relatively simple: **A pattern is a collection of elements and their relationships which can be repetitively reached or used in analysis, design, development and use (of cooperative systems).** This definition is generic and can be specialized to correspond to Alexander's, Gamma's, Borchers's, Seffah's, etc. views.

The concepts of element and relationship are generic with appropriate specialization for each context. In this way during our pattern oriented walkthrough, we can consider:

- Scenarios as patterns
- Algorithms as patterns
- Steps of methodology as patterns
- Frameworks and patterns
- Design rules as patterns
- Interaction configurations as patterns
- Etc.

Of course, individual scenarios, algorithms, steps of methodology, etc. are not patterns, but either their general structure or their reusable specialization can become patterns. In pattern based approach the actors are permanently oscillating between top-down and bottom-up views, i.e. from patterns to concrete situations, from concrete situations to reusable patterns.

As we will see in more detail thus after, we can reuse or imagine:

- Analysis patterns: for scenarios discover
- Transformation patterns: for transformation a set of scenarios into a model (CAB)
- Projection patterns: for projection of CAB to software architecture
- Interaction patterns: for collaborative application interactions
- Etc.

2 Pattern oriented walkthrough

In this walkthrough we study the patterns in a well-organized process based on a collection of models which are used in analysis, design development and use of cooperative systems.

CSCW (computer supported cooperative work) (Ellis, 94, Andriessen, 03) research proposes a new type of software, called groupware, which is an interactive multi-participant application allowing participants to carry out a "joint" task working from their own workstations. It is now a question of managing not only the man-machine interface but also the man-man interface mediated by the machine. The relationship between the participants can be considered from various points of view. Ellis et al. (Ellis, 94) proposed a matrix which classifies the nature of cooperation in regard to time - synchronous or asynchronous, and to distance -

local or remote aspects of cooperation. This classification was extended later, introducing awareness of cooperation, foreseeability or unpredictability of collaboration and location. The possibility of bringing together geographically distant people is an important contribution of groupware. The first aim of groupware is thus to propose a support for the abolition of space and time distances. Moreover, knowledge and management of the interventions of the multiple participants appear necessary. In fact, the participants constitute a work group that has to be organized with respect to working conditions, time and location. The organization can lead to the definition of different roles, sub-groups and phases of project work. The success of cooperative work can be measured by the way in which the groupware is able to create and support good group dynamics, which contributes to the disappearance of the virtuality of participants' presence. The project must be able to proceed as naturally as in collocation and without IT support. It must even take advantage from an organization of more effective work based on the new possibilities offered by information technologies (IT). The technological devices used should not interfere with the work or the group dynamics needed for project accomplishment. When designing cooperative systems, it is thus necessary to be aware that the usability aspect, the aim of which is to validate the environment suggested, is at least as significant as the engineering aspect. The evolution of users' practices during the project life-cycle must be taken into account in order to provide an effective and adaptable environment.

In-depth analysis of cooperation reveals several dimensions which must be examined, as initially proposed by Ellis (Ellis-94) with the Clover model, i.e. a support of production, conversation / communication and coordination between participants.

2.1 Scenario-based Design, CAB Model and patterns

Carroll's view of Scenarios-Based Design (Carroll, 00), "Scenarios are stories" which can be expressed more or less freely or formally, is an interesting starting point for collaborative system design and evolution. However, it seems important to go further. We propose to apply this scenario-based approach in a more organized way. In relation with the Clover model, the scenario discovering process can be driven by this model to organize discovery scenario process in relation with production, coordination or conversation space or in their intersections. **To facilitate this analysis, scenario patterns are proposed for each of Clover model space, i.e. production, conversation / communication and coordination (Figure 1 and Figure 2, 1).**

At the second step of our design process, we propose to synthesize these scenarios in a model integrating collaborative application behaviors. We call this model a CAB model (Collaborative Application Behavior Model). We consider that it is important to integrate the scenarios as soon as possible in CAB perspective i.e. to ask the scenario writers to express explicitly the position of the scenario in relation with the CAB Model. The main goal of the CAB model is to describe explicitly the structure of actors, artifacts, contexts and tasks that characterize the behavior of the cooperative application in three Clover spaces (co-production, coordination, conversation). Each scenario expressing a task might indicate its position in relation with these actors, processes, artifacts and contexts. In this way it is possible to elaborate progressively the CAB model for a given application. The CAB model for a specific collaborative application contains concrete actors, artifacts, tasks and contexts which the cooperative application will take into account. **To facilitate this transformation from scenarios to CAB, transformation patterns are proposed. Their objective is to express transformation principles, i.e. projection of each scenario to the CAB components (Figure 1, 2).**

The CAB model itself is based on patterns (Figure 1, 3). For instance, actors' organization can take different forms, as hierarchical, flat or nested which can be suggested by different patterns. In the same way, tasks expressed in different scenarios are studied in order to organize them. The goal is to eliminate redundancies and to elaborate a task tree and a task process. The task tree can be expressed in ConcurTaskTree formalism proposed by Paterno (Paterno, 00). Of course, patterns are useful to express typical task sub-trees. The process view is a workflow view with temporal and logical dependencies between tasks. Here also patterns can be used to express main intermediate workflow configurations (Saikali et al., 2001). The context view is an expression of different contexts (logical or physical) related to environment and devices constraints, if any. Their structuring by patterns seems very useful. The validation of CAB model from completeness, correctness and coherence are also based on patterns of validation. The CAB model will be used in the elaboration stage (development or prototyping).

1. register 2. identify and input in the session 3. leave a session 4. joint a session 5. observe 6. contact one or several actors 7. observe solicitations 8. answer solicitations 9. establish a synchronize contact 10. ask for advice 11. actualize	12. decline identity 13. document 14. write a memo 15. make diagnosis 16. validate diagnosis 17. organize actions 18. diagnostic together 19. repair together 20. propose help 21. describe actual situation 22. constitute an aparté

Figure 1: Open-ended list of scenario patterns which become HCI patterns

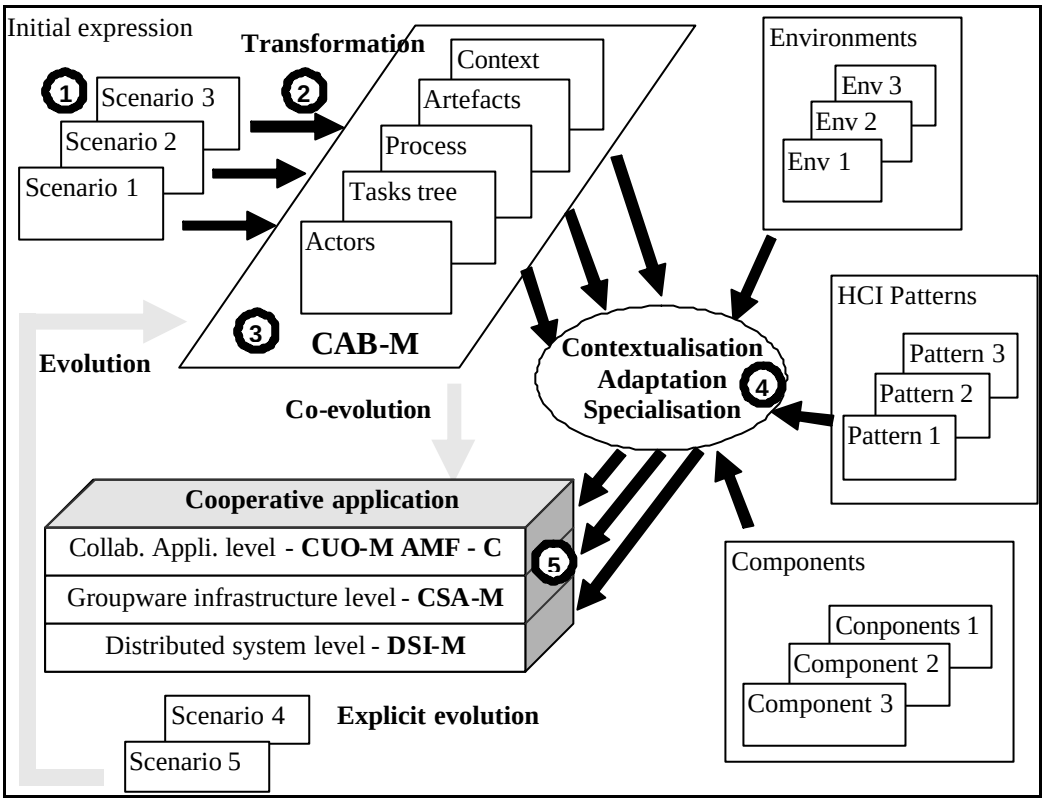

Figure 2: Model-based process for design and evolution of cooperative systems

2.2 Software infrastructure and patterns

With respect with software engineering considerations, the cooperative application cannot be carried out from scratch. It is necessary to identify different levels of development which are more or less dependent on the application. In framework based approach, three functional layers are recognized.

The top layer corresponds to the collaborative application level. It contains all the cooperative software employed by the users. This level is definitely user-oriented, which means that it manages interaction control and proposes interfaces for notification and access controls. This model is called CUO-M (Collaboration User-Oriented Model). It uses multi-user services provided by a second layer called the groupware infrastructure, called CSA-M (Collaborative System Architecture Model). It is a generic layer between applications and the distributed system. This layer contains the common elements of group activities and acts as an operating system dedicated to groups. It supports collaborative work by managing sessions, users, it groups and provides generic cooperative tools (e.g. telepointer) and is responsible for concurrency control. It also implements notification protocols and provides access control mechanisms. The last layer is essentially in charge of message multicast and consistency control. We call it DSI-M (Distributed System Infrastructure Model). Usually,

it is a computer-oriented layer which provides transparent mechanisms for communication and synchronization of distributed components which misfit with CSCW aims but which are very useful.

The degree of generality (and genericity) is not the same for these three layers and models. The lowest layer (DSI-M) is for the most part independent from the collaborative applications. The middle level (CSA-M) can be dependent on a category of applications, but is stable for each category and each application during its life-cycle. The highest level (CUO-M) is, by construction, dependent on the application, because it is constructed (or specialized) with respect to the CAB-M.

This software infrastructure is framework – pattern oriented, i.e. framework gives main software architecture principles and patterns are used at each layer to express typical and reusable local behaviors (Figure 1.5). This is the case at the collaborative application level, as we will see later. It is also the case at groupware infrastructure level, with for instance pessimistic or optimistic ways to manage concurrency, and at distributed system infrastructure in relation for example, with network speed aspect.

2.3 AMF-C as CUO Model and patterns

An appropriate CUO model should fulfill three main objectives. Firstly, it organizes the software structure to improve implementation, portability and maintenance. Secondly, it helps identify the functional components, which is essential during the analysis and design process. Its third role is to facilitate the understanding of a complex system, not only for designers, but also for end-users.

AMF-C (the French acronym for Collaborative Multi-Faceted Agent) (Tarpin-Bernard et al., 97) is our proposal for the CUO model for collaborative software which fulfills all these objectives. AMF-C is a generic and flexible model that can be used with design and implementation tools. It includes a graphical formalism that expresses the structures of software, and a run-time model that allows dynamic control of interactions.

The current trend in software engineering is to identify design patterns (Gamma, 95), which help developers to share architectural knowledge, help people to reuse architectural style, and help new developers to avoid traps and pitfalls traditionally learned only as a result of costly experience. AMF proposes a multi-faceted approach, in which a configuration of facets or each facet can be a pattern. Each new identified behavior which seems to be reusable can be formalized as a new facet, i.e. a new pattern. AMF also proposes a very powerful graphical formalism which helps understand complex systems. This formalism is used as a design tool by editors and builders. It represents agents and facets with overlapped boxes, communication ports with rectangles which contain the associated services, and control administrators with symbols which express their behavior. Using the AMF, 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 (figure 3). It constitutes an elementary pattern of interaction. 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 et al., 1998).

2.4 Development, evolution and patterns

We propose a development process which is based on projection of the CAB model on the software architecture based on CUO, CSA and DSI models. This projection is a complex transformation with mainly three aspects (Figure 1,4), which are Contextualization, Adaptation and Specialization:

- The contextualization process transforms CAB model in an executable application in relation with the context describing the hardware configuration of the workstation (PC, PDA,…) to take into account Capillary CSCW (cooperative work using handheld devices) (David et al., 2003b).
- The adaptation is the process which takes into account user's characteristics and his preferences.
- The specialization is the process which takes into account the roles assigned to the user and corresponding tools in order to produce an appropriate working configurations.

These transformations are in relation with architectural choices and interface plasticity. They are driven by three sorts of patterns: environmental patterns, HCI patterns and component patterns.

2.4.1 AMF-C architectures and patterns

Two collaborative architectures have been proposed in AMF-C context (Tarpin-Bernard et al., 1998): fragmented and replicated framework.

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.

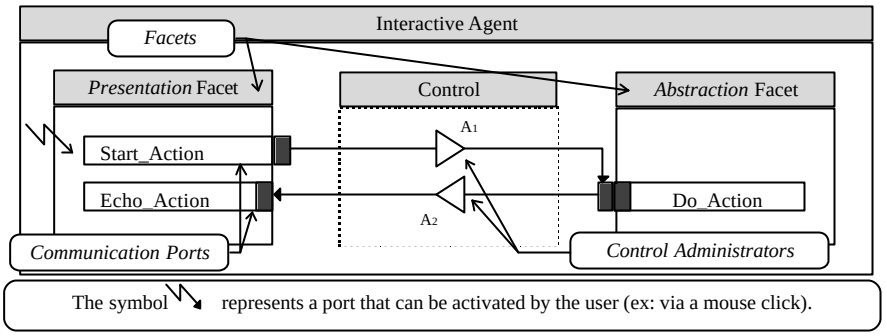

Figure 3: Elementary pattern of interaction expressed in AMF

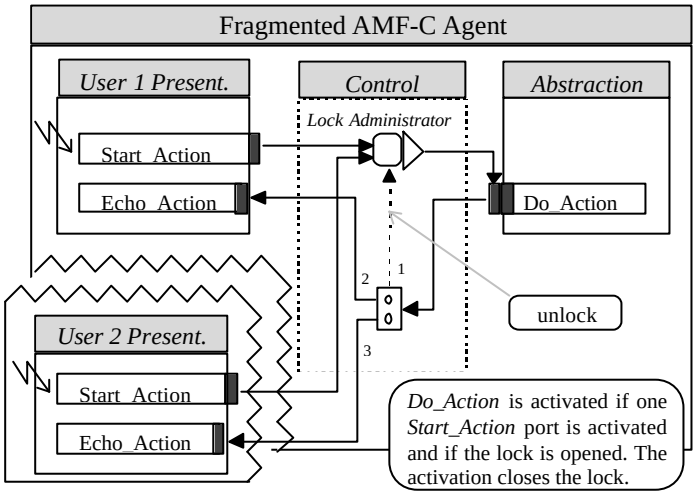

Figure 1: An example of elementary interaction on a fragmented AMF-C agent

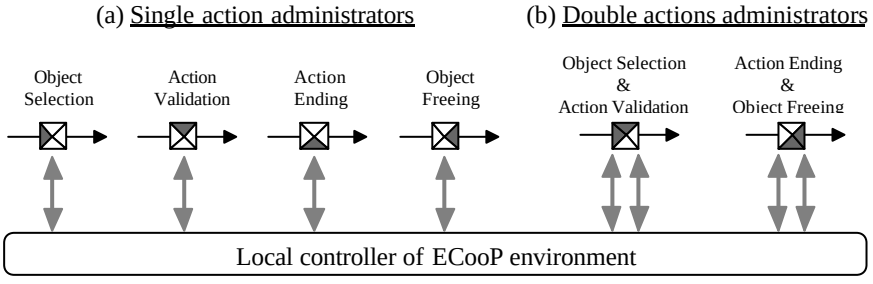

Figure 5: Cooperative administrators of AMF-C

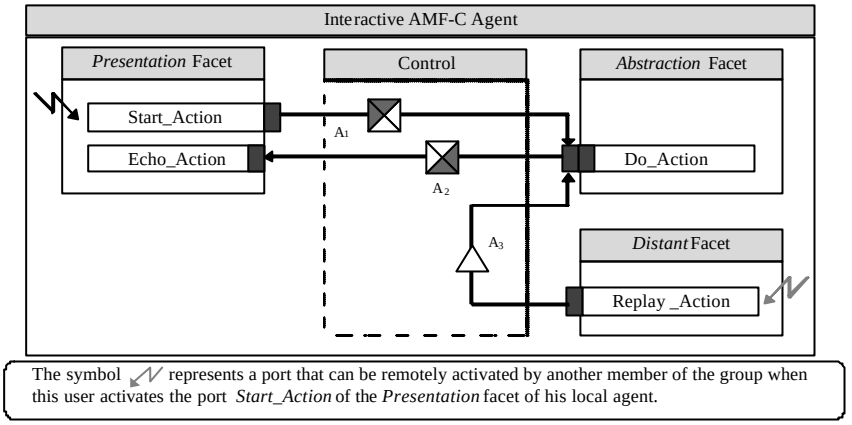

Figure 6: A first interaction pattern on a shared agent modelled with AMF-C

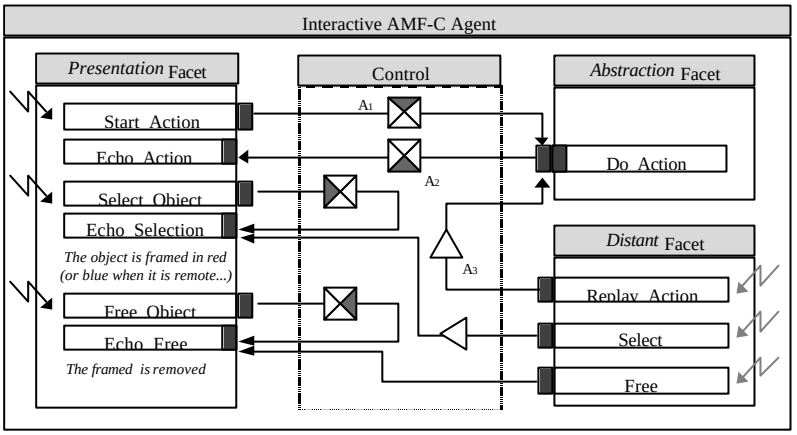

Figure 7: A second interaction pattern on a shared agent modelled with AMF-C

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.

The replicated version of the AMF-C model fits very well with ECooP, CSA model implemantation. 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 5 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).

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. We give here two examples of patterns associated to the replicated framework.

Using the six administrators that we have presented figure 5 and referring to the standard interaction pattern presented figure 3, we can define a first pattern of cooperative interaction (Figure 6). 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.

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 7). This pattern allows users to see the objects which are locked (locally or remotely).

2.4.2 Workflow, coordination and patterns

In this paragraph, we give a short example of how our framework 2FLOW (Saikali et al., 2001) for adaptive workflow can be used for the construction of a collaboration pattern. This example is about the management of a dialog among many participants.

The dialog can be assimilated to a generic process composed of generic activities that are: "give hand" (allows a participant to talk or to use a certain tool), "ask hand", and "free hand". However, the execution order of these activities is not known by advance but the events that enact them are clearly identified: "hand released", "hand requested" and "hand attributed". It is also possible to identify two generic roles that can be associated to the activities: "participant" and "coordinator". Notice that an actor can take both roles. This process can easily be translated into 2FLOW components, as it is shown in figure 8.

2FLOW adaptability mechanisms allow us to particularize this pattern into different processes that share the same principle of execution. For example, the previous pattern can be specialized into a process for managing shared resources. Moreover, the actor-role-activity approach that we use allows the involvement of automatic actors in the process, as well as human actors. For example, we can consider using a computerized agent in the role of the dialog coordinator.

Naturally, thanks to the inherited base behaviors, the pattern and its derived processes are fully functional; they can thus be directly integrated into another application, a collaborative one for instance.

2.4.3 Evolution

During application life-cycle, users progressively change their perception of the system and their use. The cooperative application could, during its use, take into account requirements concerning evolution expressed explicitly or implicitly by its users. This is particularly the case in the context of Capillary CSCW (cooperative work using handheld devices) (David et al, 2003b) in which behavior evolution is more important related to context condition (connected or disconnected work) and the device used. To take into account this evolution in an explicit way, new scenarios can be presented which upgrade or extend initial the COB model. The Cooperative Application Behavior model evolves smoothly and it is important to be able to take this evolution into account. This evolution can vary in importance and its impact can be taken into account in different ways:

- If the evolution is within the scope of the system, i.e. these adjustments have been imagined and they are at the disposal of the user in the "configuration panel" (use of different interaction modalities, modification of awareness, choice of different WYSIWIS relaxation, plasticity of user interface, etc.) as alternative patterns.
- If the evolution is more important and leads to modification of the CAB model, two different solutions are possible:
  - Change of interaction pattern with the same algorithmic behavior: this evolution is relatively easy and can be performed by the end-user (i.e. by visual programming using AMF-C graphic formalism),
  - Change is more important also with algorithmic behavior evolution: in this case a developer intervention seems inevitable.

To take into account this re-configurability, adaptability and flexibility, we need new scenarios, which can also replace or modify existing ones. Their processing leads to modification of the CAB model and has an impact on the CUO model. The evolution can either be implemented on the existing IT support or this support can evolve too. We expect that the latter evolution, which concerns the CSA and DSI models, will seem to be out of scope of the evolutions which can be taken into account dynamically. To be able to modify dynamically CUO, we need to have at our disposal a meta-model of the CUO model to create dynamically new AMF-C agents in the relation with the re-configuration of the COB model. This co-evolution can be implemented either by the end-user himself or at least expressed by him, on his directives or under his control, by the developer. In this context patterns are the good way to manage the diversity and increase the granularity of intervention.

3 Conclusions

Two main observations can constitute the conclusion of this position paper. First one is related to the project of case study to be elaborated during the workshop on the theme of e-shop. It seems me that the problematic of e-shop can be relatively close to our problematic of cooperative systems design, development and use. In this way it seems interesting to transfer, adapt and complement the patterns identified and proposed in our study.

Second observation concerns the comprehensive view of patterns as point of junction between HCI and software engineering fields. This concept is at the same time very important, as demonstrated our study, but can be consider as buzzword, if the definition is not clarified as well as the relation with the concepts of scenario, use-case, algorithm, design rule, step of methodology, etc. That should be one of the future developments during or after the workshop.

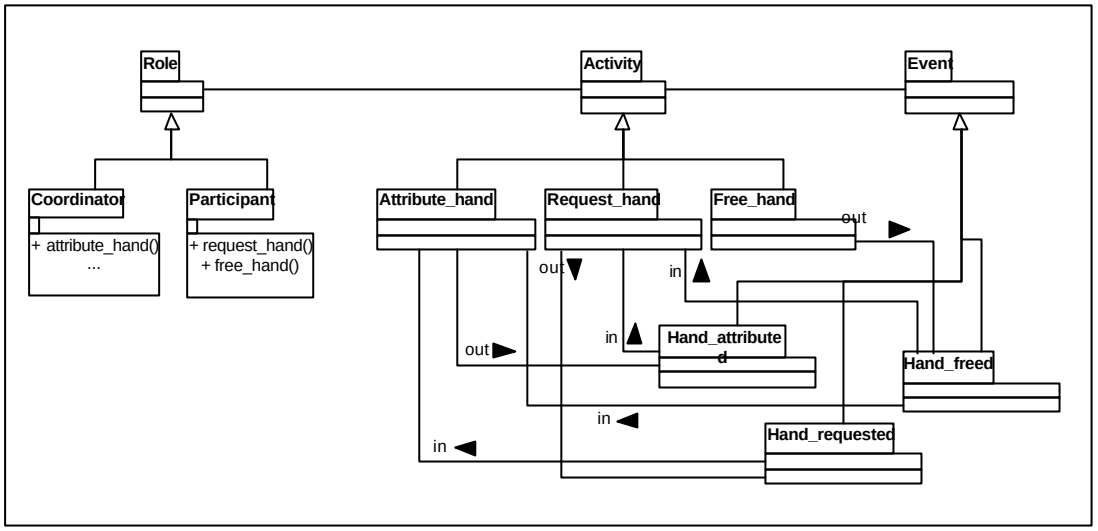

Figure 8: Collaboration pattern - managing a dialog

References

Alexander C. (1977) *"A Pattern Language: Towns, Buildings, Construction"*. Oxford University Press, New York.

Andriessen J.H.E. (2003). *Working with Groupware: Understanding and Evaluating Collaboration Technology*, Springer, CSCW Series

Borchers J. (2001). *A Pattern Approach to Interaction Design*, Willey Series in Software Design Patterns, Wiley.

Carroll J. M. (2000). *Making Use: Scenario-Based Design of Human-Computer Interactions*, The MIT Press 2000.

David B., Chalon R., Delotte O., Vaisman G (2003a). Scenarios in the model-based process for design and evolution of cooperative applications. In: *Human-Computer Interaction Theory and Practice* (Jacko J., Stephanidis C. ed.) Vol. 1, LEA, London, pp. 68-72.

David B., Chalon R., Vaisman G., Delotte O. (2003b). Capillary CSCW. In: *Human-Computer Interaction Theory and Practice* (Stephanidis C., Jacko J., ed.) Vol. 2, LEA, London, pp. 879-883.

David B., Chalon R., Delotte O., Ros J., Boutros N. (2003c). *Travail coopératif capillaire en dépannage, maintenance et interventions de crise*. In: 5th International Congress on Industrial Engineering, 26th-29th October 2003, Québec, Canada

Ellis C.A., Wainer J. (1994), *A Conceptual Model of Groupware*, ACM CSCW'94 Conference, pp. 79-88, ACM Press

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

Paterno F. (2000). *Model-Based Design and Evaluation of Interactive Applications*, Applied Computing Series, Springer.

Saikali K. David B. (2001). Using Workflow for coordination in Groupware applications. Blandford, Vardendonckt, Gray (eds), *Interaction without Frontiers, Proceedings of Human Computer Interaction 2001*, Springer Verlag, People and Computer vol.15

Seffah A. (2003). Pattern Corpus and Web-based pattern catalogue, *In progress work*, University of Concordia, Montreal

Tarpin-Bernard F., David B(1997), AMF a new design pattern for complex interactive software ?, International HCI'97, San Francisco, 24-29 August 1997, in *Design of Computing Systems*, 21 B, Eds Elsevier, ISBN 0444 82183X, pp 351-354

Tarpin-Bernard F., David B.T., Primet P. (1998). *Frameworks and patterns for synchronous groupware: AMF-C approach*. IFIP Working Conference on Engineering for HCI: EHCI'98, Greece. Kluwer Academic Publishers. pp. 225-241.