A model and a pattern for data collection on collaborative actions in CSCL systems

Alejandra Martínez[1], Luis A. Guerrero[2], César A. Collazos[3]

[1] Dept. of Computer Science, Universidad de Valladolid, Valladolid, Spain
amartine@infor.uva.es
[2] Dept. of Computer Science, Universidad de Chile, Santiago, Chile
luguerre@dcc.uchile.cl
[3] Dept. of Systems, Universidad del Cauca, Popayán, Colombia
ccollazo@unicauca.edu.co

Abstract

There is an increasing interest in the CSCL field towards the definition of frameworks that support analysis of interactions in order to understand or to regulate collaboration. These analysis processes draw on the *automatic collection* of data about the collaborative processes for its further analysis by manual or automatic means. Despite this interest, current proposals solve this automatic collection using ad-hoc solutions, and thus they do not consider how this problem can be solved in a modular and reusable manner, so that it could be applied to different collaborative situations and analytical approaches. This paper shows how CSCL can benefit from the field of software engineering by the adaptation of the *command* design software pattern to the problem of CSCL data collection. In order to perform this adaptation, we draw on a model of interaction that defines the concept of *collaborative action* as the basis of any interaction, which is also described in this paper. These two aspects: the concept of collaborative action and the adaptation of the command pattern to the problem of CSCL data collection will allow us to address conceptual as well as implementation issues related to the modelling of interactions in CSCL.

1 Introduction

Analysis of interactions has become a main research topic in the last years in CSCL. However, research has mainly focused on conceptual issues [8] or on the development of experimental prototypes focused on testing a particular collaboration support model [7]. While these approaches have shown the interest of this CSCL work line, the area has so far neglected the problem of the design of these systems from a software-engineering point of view. We claim that this is an important topic, as we need to provide the developers with conceptual and practical tools that facilitate the desired integration of analysis functions in CSCL systems. This integration has to meet software quality criteria, such as modularity and reusability. We propose in this paper to borrow from the software design patterns point of view [5] in order to identify and propose design solutions that meet these objectives.

The *collaboration management cycle* proposed in [7] defines the phases that these collaboration support tools have to go through in order to perform their tasks. The first of these phases consists on the *data automatic collection* about the collaborative processes. We are going to focus on this phase, as it is the one where the interface between the functions oriented to mediate collaboration and to perform the analysis is established. Moreover, being the first phase, is the one on which the rest of the analysis processes rely.

In order to apply a software pattern to the data collection in CSCL, some steps are needed in advance. We need to provide an interaction *concept* that is both appropriate for the domain as well as operationally representable by a computer. We propose the concept of *collaborative action* to fulfill this objective. Once this concept is defined and described, we propose how a well-known software design pattern - the *command* pattern [5],- can be used to implement the data collection in systems that follow the collaboration management cycle.

This work is part of ongoing research performed by the authors that aims at the definition of a component based framework for the definition of reusable, modular and configurable solutions for the implementation of

collaboration support systems [1]. It also draws on previous authors' experience in the proposal of a patterns system for CSCW applications [6].

The rest of this paper is structured as follows: section 2 presents the concept of collaborative action, and section 3 presents its computational representation using an UML diagram. Then, section 4 introduces the *command* design pattern and how a logging tool in a CSCL context can use it. Finally, section 5 presents the main conclusions and further work.

2 Collaborative action concept

The concept of collaborative action is not easy to define. Although it has been extensively used in the literature, either its meaning has been taken for granted or it has been defined specifically within the context of each approach. The Merriam-Webster's Collegiate Dictionary defines interaction as *"mutual or reciprocal action or influence"*. Therefore, interactions can be conceptualised as *actions* that comply with a specific characteristic: *reciprocity*. Then, we are to answer what means reciprocity in our CSCL context and how it can be detected. In dialogue-based analysis, reciprocity seems to be an easy issue, if we assume that any utterance is an interaction. However, several authors have shown that this is a rather simplistic view. [3] states that the *"degree of interactivity between pairs is not defined by the frequency of interactions, but for the degree in which they have an influence in the cognitive processes of participants*", which gives an idea of the complexity of the concept. However, this statement can be challenged for two reasons. On the one hand, it is difficult to apply operationally. On the other hand, it focuses on the cognitive view of interactions, leaving apart the participatory aspects, which we must consider when we want to study interactions in all their dimensions. Another well-known challenge to the approaches that rely on explicit interactions is the presence of *silence* in many situations of real collaboration. Indeed, silence can be almost as significant, if not more, than explicit utterances [8], and therefore we should not rely exclusively on explicit discourse, but include actions when performing analysis.

In conclusion, we need a definition of interaction able to deal with actions and discourse, covering cognitive and participatory aspects of interaction, simple to process and able to deal with silence and inactivity. Taking this into account, we propose the following definition for interaction as *"an action that affects or can affect the collaborative process. The main requirement for an action to be considered a possible interaction is that the action itself or its effect can be perceived by at least a member of the group distinct of the one that performed the action*". It provides a generic view of interaction, without restricting it to a particular source of data or analytical perspective, and gives an operational criterion to select appropriate input for interaction analysis. It is also able to deal with the aforementioned problem of silence.

Before we continue with the rest of the proposal, we want to point out that the study of human action from a situated standpoint is rather different from the study of behavior in the conductivist paradigm. In order to be fully understood, human action needs to consider the *context* in which it is taking place [10]. Thus, representation of collaborative action from a situated learning perspective needs to consider context, which is in fact an open question in the modelling and analysis of human action. Next section will elaborate on this issue.

3 Computational representation of collaborative action

Once we have presented the concept of collaborative action, we will present our proposal for its computational representation in an open and standard format. As mentioned above, there are several issues one must consider when trying to model interactions. First, we have to face the problems related to the modelling of *context* in collaborative situations. Next, we have to provide a classification of collaborative action that fits the definition we have presented in the previous section. Section 3.1 will ellaborate on the representation of context, while section 3.2 will present the three main types of action we have identified.

3.1 Context representation

Although research in CSCL currently agrees on the need of considering *context* when interpreting human action, it is an open issue how this is to be done, and what elements should be considered when modelling it, especially from a computational perspective. Normally, this issue is solved depending on each particular situation, and there are few proposals of a generic representation of context.

A possible exception to this rule is the use of Activity Theory as a framework for the activity representation in its social context [4]. Although we agree that this approach is being successful at broadening the analytical

perspective of researchers, we claim that its concepts are rather generic, and need to be complemented with information related to the pedagogical context in which the learners are interacting. As an alternative, we propose to use the concepts of the DELFOS framework for the modelling of context in collaborative learning.

DELFOS was developed specifically for the definition of CSCL situations, taking into account social, pedagogical, and technological issues. It presents a model of collaborative situations, and has been validated by its use in the design of several applications [9]. It proposes the concept of *situation* to model the general features of a learning environment, including learning objectives, number of expected participants, metaphors, etc. DELFOS provides for the definition of users, roles, objects and groups that intervene in the situations. All these elements are represented in the model, as shown in figure 1.

3.2 Collaborative action representation

The second aspect we face in our proposal is to provide a generic and operational taxonomy for the representation of collaborative action. As mentioned beforehand, existing approaches focus on the representation of a single feature of the interaction, which hinders the desired integration of different sources of data. We aim at integrating dialog and action, as well as automatic and manually collected data in a common structure, by means of a new classification that focuses on the actors that take part in interactions. The main advantage of this approach is that it easily accommodates to the collected data in each system for each type of interaction.

The proposal distinguishes between *direct*, *indirect* and *participation-oriented* interactions (see figure 1):

Direct interactions. They represent the typical interaction idea, with a *source* and one or more *receivers*. It can be mediated by a *channel*, and may specify its *content*, which will normally, but not necessarily, be a dialogue representation.

Indirect interactions: This interaction is characterised by being mediated by an *object*, and therefore, it is the more common in shared workspace environments.

Participation: As an action that has no object neither receiver. Represents a *generic intervention* in a collaborative environments.

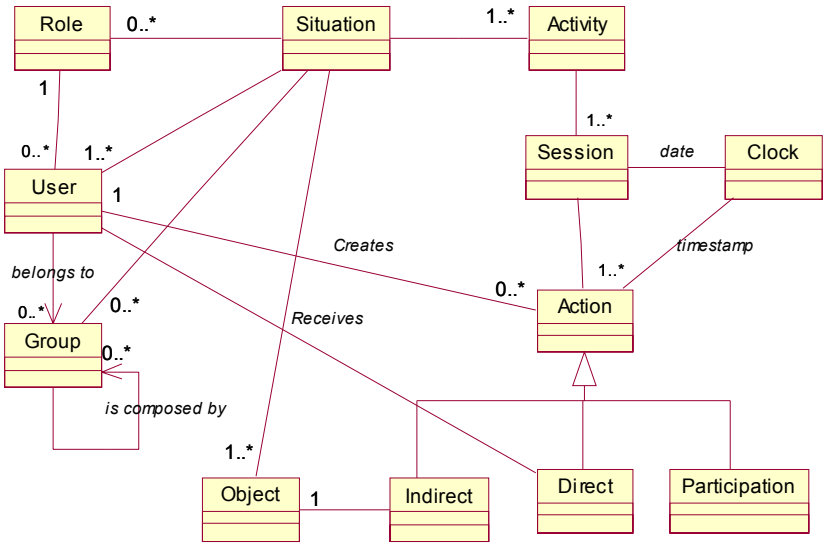

Fig. 1. Model of collaborative action. It shows the classes related with the context as well as the main types of collaborative action we have identified

Although this model is valid for *face-to-face* representing actions as well as those that are performed *through* the computer, we will restrict our scope to the latter. These automatic actions will always correspond to a command (or set of commands) executed by the CSCL application in response to an event (or to a set of them). Therefore, we have to think on how the commands are to be implemented in the application in order to facilitate the collection of actions. This solution has to be generic, so that it can be adapted to different types of actions and to different CSCL situations. It also has to provide independence between the code of the CSCL application

and the code of the collection of data, as they are different functionalities and it is desiderable that one of them can be modified without affecting the other. Next section presents a particular software pattern called the *command pattern* that meets these requirements, and how it can be applied to solve the logging of actions.

4 Adaptation of the *command* pattern for the collection of actions

The *command* design pattern is a recommendation for the implementation of all the commands of an application in order to promote independence between the *sender* of a request and its *receiver*. The pattern proposes to implement all the commands as objects with a method *Execute(),* which is the one that fulfills the command functionality. The set of these *command objects* constitutes the 100% of the functionality of the application, i.e., everything that the application has to do, including any action and the data generated by it.

As shown in figure 2, the key feature of this pattern is an abstract *Command* class, which declares the interface for executing operations, which in its simplest form includes an *Execute()* operation. The concrete *Command (ConcreteCommand)* classes are declared as subclasses of the abstract *Command* and are in charge of implementing its interface. Each *ConcreteCommand* class specifies the *Receiver* of the command by means of an instance variable that stores it. The *Receiver* can be any class that has the knowledge to fulfill the request. Finally, the *Invoker* is responsible of calling the *Execute()* operation in the *Command* interface.

The interaction between the objects in this pattern works as follows: The *Client* creates a *Concrete Command* object and sets its *Receiver*. The *Invoker* stores the *Concrete Command* object that has been instantiated, and issues a request by calling *Execute()* on the *Command* object. Then, the *ConcreteCommand* object invokes operations on its *Receiver* to carry out the request.

This pattern can be easily adapted to support the collection of data by adding a logical *is_loggeable* instance variable to the abstract *Command* class. This acts as a switch: if it set to true, the action to which this command corresponds will be logged; if it is false, the action will not be logged for its further processing. The logging processes are performed by the *Logger* class, that receives the command data and represents it in the desired format using the *WriteAction()* method. Normally, the output is a text file, but it can take any other structure that is convenient for the subsequent analysis processes. Therefore, as it can be seen in the figure, a *Logger* class can be considered as a special type of *Receiver* that is invoked conditionally if the *is_loggeable* variable is set to true in a particular command.

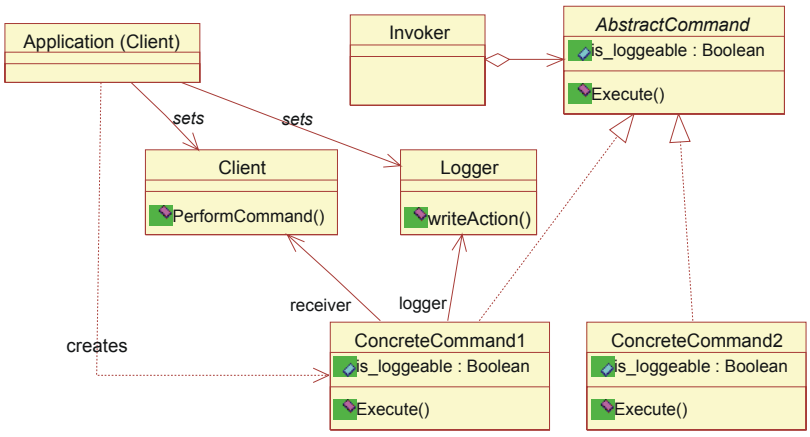

Fig. 2. The *command pattern* adapted to the collection of collaborative action. The figure on the left shows the structure of the pattern and the figure on the right shows the interaction diagram

Once we have explained the pattern and its use for implementing the collection of data in a CSCL system, let us now discuss how to integrate it with the concept of collaborative action that has been proposed beforehand. As it has been mentioned, there is a link between the commands and actions. The semantic level of a collaborative action is that of the usual commands one can expect to find in a CSCL application (create an object, issue request, answer to a question, etc.). Sometimes these commands result from the aggregation of a set of lower level or more fine-grained ones. Therefore, it is possible to define the relationship between the *concepts* of command and collaborative action, either as a one-to-one relationship, or as an aggregation of commands that constitute an action. The *command* pattern is also appropriate to log an action represented as an aggregation of

commands, as it is possible to assemble the low-level commands in a *composite command* class that represents a particular type of *collaborative action* to be logged.

In conclusion, we see that the *command* pattern applies quite straightforwardly to the logging needs of a CSCL application. A positive consequence of this is that we can take advantage from the benefits reported from the use of this pattern. If we recall that the pattern was meant to decouple the object that invokes a command from the one that performs it, CSCL applications that use it will not need to know how to log any of their actions. This will be a responsibility of the concrete action classes. The pattern is flexible, as it allows for the definition of new types of action (through a new specialisation of the abstract class Command). It facilitates also the modification of the logging mechanisms by changing the *Logger*() class, that will not affect the code of the CSCL application being logged.

Apart from these benefits, the pattern is the base of new functionalities that become very easy to implement with its use. For example, a tool that shows the user (teacher, evaluator, and student) a menu with all the possible actions performed in a particular CSCL tool. The user can choose which of these actions s/he wants to be logged, which will be represented by a *true* value in the *is_loggeable* attribute of the class that represents that action. The actions that are not meaningful for a particular analysis and therefore have not been chosen in the menu, are omitted from the log by setting to *false* this attribute. This eventually results in a more accurate and easy to analyse log than other does that simply collected all the data on the computer. This is only an example, that illustrates the interest of taking care of software design quality issues in the elaboration of CSCL applications in general, and of analysis functions in particular.

5 Conclusions and future work

CSCL needs to take software design issues seriously in order to enhance the quality of the applications in the field. This is not a pure technical problem, as the modelling of actions which is at the core of the proposal needs to take into account issues related to the context, types of interactions, etc. which draw on the needs of the domain. This fact has been reflected in the definition of collaborative action we have presented in the paper.

We have shown with a simple example how conceptual issues have been applied to the computational representation of action. Also how the use of a software pattern can facilitate and enhance the design of a core functionality in CSCL such as the collection of data about collaboration for its further analysis.

The ideas presented in this paper are related with ongoing research of the authors. On the one hand, the model of collaborative action has been used as the base for the definition of a DTD for the computational representation of collaborative action logs in XML. The genericity of the model and the standard character of this language we think that it could be used by other CSCL developers in order to produce interactions action in a way that could be easily shareable between CSCL systems and evaluation tools. On the other hand, the pattern presented in this paper is part of a more general research objective that aims at proposing a framework for the design of CSCL systems that considers the specificity of the domain. In short term, an immediate objective is to validate the ideas presented here by implementing a system that follows the *command* pattern as defined in this paper.

Acknowledgements

This work has been partially funded by European Commission Project EAC/61/03/GR009 and Spanish Ministry of Science and Technology Project TIC2002-04258-C03-02 and by the research mobility program from the University of Valladolid.

References

[1] Asensio, J. I., Dimitriadis, Y. A., Heredia, M., Martínez, A., Álvarez, F. J., Blasco, M. T., and Osuna, C. A., "From Collaborative Learning Patterns to Component-based CSCL Applications", European Conference on Computer Supported Collaborative Work (ECSCW03), Workshop:From Good Practices to Patterns, Helsinki, Finland, Sept. 2003.

[2] Dimitracopoulou A. & Petrou A. (2003) "Advanced Collaborative Learning Systems for young students: Design issues and current trends on new cognitive and metacognitive tools". In THEMES in Education International Journal.

[3] Dillenbourg, P. (1999). Introduction; What do you mean by “Collaborative Learning”? In P. Dillenbourg (Ed.), *Collaborative learning. Cognitive and computational approaches* (p. 1-19). Oxford: Elsevier Science.

[4] Fjuk, A., & Ludvigsen, S. (2001). The complexity of distributed collaborative Learning: Unit of analysis. In P. Dillenbourg, A. Eurelings, & K. Hakkarainen (Eds.), Proceedings of European Conference of Computer Support Collaborative Learning (EuroCSCL´2001); Maastricht. (p. 237-243). Maastricht: Maastricht MacLuhan Institute.

[5] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[6] Guerrero, L.A. and Fuller D. *A Pattern System for the Development of Collaborative Applications.* Information and Software Technology, Vol.43, No.7, May, 2001, pp. 457-467

[7] Jermann, P., Soller, A., & Muehlenbrock, M.(2001). “From mirroring to guiding: a review of the state of the art technology or supporting collaborative learning”. Proceedings of European Conference of Computer Support Collaborative Learning (EuroCSCL´2001); Maastricht: Maastricht MacLuhan Institute.

[8] Littleton, K., & Light, P. (Eds.). (1999). *Learning with computers: Analysing productive interaction*. London: Routeledge.

[9] Osuna, C., Dimitriadis, Y., & Martínez, A. (2001). Using a theoretical framework for the development of educational collaborative applications based on social constructivism. In P. Dillenbourg, A. Eurelings, & K. Hakkarainen (Eds.), *Proceedings of EuroCSCL* (pp. 577 - 584). Maastricht: Maastricht MacLuhan Institute.…

[10] Wilson, B., & Myers, K. (2000). Situated Cognition in Theoretical and Practical Context. In D. Jonassen & S. Land (Eds.), *Theoretical foundations of learning environments* (pp. 57-88). Mahwah, N.J.: Lawrence Erlbaum Associates.