



Repositorio Institucional de la Universidad Autónoma de Madrid

<https://repositorio.uam.es>

Esta es la **versión de autor** del artículo publicado en:

This is an **author produced version** of a paper published in:

International Journal of Continuing Engineering Education and Life Long
Learning 13.3-4 (2003): 268 – 279

DOI: <http://dx.doi.org/10.1504/IJCEELL.2003.003275>

Copyright: © 2003 Inderscience Enterprises Ltd.

El acceso a la versión del editor puede requerir la suscripción del recurso

Access to the published version may require subscription

Developing applications with a framework for the analysis of the learning process and collaborative tutoring

Miguel Ángel Mora*, Roberto Moriyón and Francisco Saiz

School of Computer Science, Universidad Autónoma de Madrid,
Carretera de Colmenar Viejo, Km 15, 28049 Madrid, Spain
E-mail: Miguel.Mora@uam.es E-mail: Roberto.Moriyon@uam.es
E-mail: Franciso.Saiz@uam.es
*Corresponding author

Abstract: In this paper we describe FACT, a Framework for the Analysis of the learning process and Collaborative Tutoring, developed at Universidad Autónoma de Madrid, Spain. First we describe guided collaborative tutoring; the type of tutoring that FACT makes possible. After that, we show the architecture of FACT, and different applications developed with it, as well as how it can be used to build new teaching tools from simple training applications. FACT uses learning histories and related annotations to enhance the guidance of the teaching process. It allows students and tutors to review the work of the students and the history of their work sessions, using the original application, while allowing reviewers to extend that history. Optionally, reviewers can attach commentaries to some points of the extended and revised histories. The generated applications can be used collaboratively, either synchronously or asynchronously. The generated histories can be stored in order to create a knowledge database with solved problems and commentaries from tutors and students.

Keywords: Framework; guided collaborative tutoring; computer supported collaborative learning.

Reference to this paper should be made as follows: Mora, M.A., Moriyón, R. and Saiz, F. (2003) 'Developing applications with a framework for the analysis of the learning process and collaborative tutoring', *Int. J. Continuing Engineering Education and Lifelong Learning*, Vol. 13, Nos. 3/4, pp.268–279.

Biographical notes: Miguel Ángel Mora is a PhD student at the School of Computer Science at Universidad Autónoma de Madrid (UAM), Spain. He graduated in computer engineering in 1997 at UAM, and has been working in computer supported collaborative learning and computer supported collaborative work research since then.

Roberto Moriyón is a Professor at the School of Computer Science at Universidad Autónoma de Madrid, Spain. He graduated in mathematics and received a PhD degree in mathematics from Princeton University in 1979. His research areas are, among others: computer assisted teaching, user interface development and computer supported collaborative learning.

Francisco Saiz is an Associate Professor at the School of Computer Science at Universidad Autónoma de Madrid (UAM), Spain. He graduated in mathematics and physics, and received his PhD in computer science from UAM in 1994. He has carried out research on automatic problem solving in mathematics, model-based user interfaces, automatic data presentation on the web and computer assisted mathematics education.

1 Introduction

Nowadays there are many courses offered through the internet, which allow a number of students to access educational resources in a flexible way. Distance learning technologies simplify the creation and use of these courses while reducing their development costs.

Specific problems arise when students need complex feedback from the teacher or when they have to work together to try to solve complex tasks. In these situations, since it is usually impossible for a teacher to help all the students simultaneously, it is not possible to communicate synchronously with all of them. Moreover, a deep asynchronous analysis of the problems the students are facing can only be achieved by reproducing the original context in which the problems arise. One possible solution to this kind of problem is to use asynchronous collaborative tools, which help tutors to review the work of the students in the same context in which the problems arise and to communicate their remarks asynchronously in a flexible way.

In recent years a large number of experiments, with collaborative applications that enhance the learning process, have been developed. Most of them are based on generic collaborative tools, like shared workspace systems such as BSCW [1], collaborative editors or conference systems like NetMeeting [2]. For example, with NetMeeting we can share applications synchronously, so the tutor can teach a student to use some tools interactively. Another solution is to record a demo of the session so students can review it later [3]. The problem is that this method is very limited because these videos generate too much network traffic, need a long time to be downloaded, and do not permit the material to be easily re-edited by the students. Related solutions based on the analysis of interaction histories have been proposed [4,5]. One solution to the previous problem that is closely related to these proposals is to use a guided collaborative tutoring system, which permits tutors and students to review their work and collaborate between them using the same collaborative applications. This technique can minimise the network traffic while allowing incremental problem solving [6].

In this paper we describe FACT, a Framework for the Analysis of the learning process and Collaborative Tutoring. FACT uses learning histories and related annotations to enhance the guidance of the teaching process. It allows students and tutors to review the work of the students, the history of their work sessions and alternative proposals by the tutor or by other students, using the original application, while allowing reviewers to extend that history with new proposals. Optionally, reviewers can attach commentaries to some points of the extended and revised histories. The generated applications can be used collaboratively, either synchronously or asynchronously. The generated histories can be stored in order to create a knowledge database with solved problems and commentaries from tutors and students.

The rest of the paper is organised as follows: first we discuss issues related to guided collaborative tutoring, the type of tutoring that FACT makes possible, we describe the requirements that guided collaborative tutoring applications must satisfy. Then, we describe the architecture of FACT, and we show different applications developed with it, and how it can be used to build new teaching tools from simple training applications.

2 Guided collaborative tutoring

A fundamental aspect in the learning process is the resolution of a student's doubts that arise during his or her work. In this paper we are concerned with the development of technologies that simplify the transformation of simple tutoring applications into ones that allow a tutor to help students to solve the doubts they have while working with the application. We shall see that as a by-product of our work, in some cases it will be possible to use the same mechanisms in order to give the student a more automatic kind of help without the need of a human tutor.

The problem takes place in two completely different situations. In the first case, passive learning occurs while the student is acquiring static information from some type of media (text, images, simulations, etc); this information can be part of a course. In this case usually there is no external evidence of the difficulties the student is facing. The only way a student may notice their need for feedback is by becoming aware that there is some part of the information they are supposed to apprehend but that they do not understand well. The amount of help that can be given to students in this situation is very limited. When there is information that the students do not understand or they are having other problems, the type of assistance available consists mainly of providing them with the means to communicate easily with the tutor. The communication channels are the standard ones, and the problem becomes one of sending static information to the tutor. Some kind of automatic help can also be given by means of hypermedia when the difficulties can be anticipated by the designer of the learning materials, and semi-automatic approaches are also sometimes possible, by means of intelligent search tools.

On the other hand, when a student is learning new skills by executing specific tasks, he or she is in a situation described as active learning. This can happen while carrying out experiments in a simulated laboratory, solving problems interactively or doing virtual work with different computer applications; for example by means of simulation of an artefact. In this case, it very often happens that students are not aware of the mistakes they make. Typically, while solving a problem, students often find out that something is wrong either because they are not able to find the solution or because a test for the solution that has been obtained shows it is not acceptable. In this case the detection of the mistake can be extremely complicated, due both to the complexity of the problem and the difficulty of finding the precise instant when an error has occurred. The specific types of application for which our approach works are those that are conceived for practical learning and where the actions the student has to accomplish consist of a discrete set of events. Thus, for many drawing applications and computer games that allow continuous motion of objects, our work cannot be used, but most interactive applications can use our mechanisms in order to teach their use in a guided collaborative way.

By guided collaborative tutoring we mean a style of tutoring that allows a single tutor to review the work of a student or a group of students by reproducing the steps they have

Developing applications with a framework

accomplished, proposing alternative solutions, and adding comments to the steps taken by the students. Comments can also be added to the previous recommendations of the tutor.

In this paper we show a framework that can increment the functionality of a suitable computer application for practical training by converting it in a relatively simple way into a guided collaborative tutoring application. Moreover, the framework allows different levels of synchronisation between the students and the tutor, as we shall describe below. It can also be used by a group of people in order to follow a synchronous explanation of a work session accomplished by someone else.

In the remaining part of this section we describe the requirements that guided collaborative tutoring applications must satisfy. Our framework must give support for all these requirements.

In order to allow tutors to synchronously guide students by reviewing their previous work with them, guided collaborative teaching applications must give support for the use and management of histories of collaboration. This also allows the teacher to analyse the work of the students at any given moment, by adding comments or alternatives, which can be played as animations by the students. This is the main feature, which must be provided by applications of this kind.

Besides this, an analysis of the temporal relations between the different steps in the process of the resolution of students' doubts, by a teacher, will give us more synchronicity requirements. These steps are:

- doubt discovery
- its communication
- search for a solution
- its communication
- its analysis.

They must be accomplished in this order, in a synchronous or asynchronous way.

Another fundamental requirement is that reviewers of working sessions, usually teachers, must be able to formulate alternative paths of action by just following them from a given situation while they are reviewing the original session.

Other requirements for guided collaborative tutoring are the following ones: students must be able to work by themselves and to collaborate between themselves. Comments from the students must be attached to the context in which they are made, so that an asynchronous observer of their actions can see them as appropriate.

Tutors have privileges to synchronously guide their students by enforcing them to know their proposals and comments, and to accept their decisions with respect to the next decision to be taken.

Both the students and the tutor can also make asynchronous analysis of the working session, either in isolation; in which case they can later have a reviewing session of their analysis, with the other students, in order to guide them in future work, or synchronously with the remaining participants in the session; in this case, the tutor can also lead the process by means of specific privileges.

Both the students and the tutor can switch seamlessly between being synchronised with the working session or working on the analysis of its history.

Tutors can receive reports from specialised agents that analyse the different working sessions, and call their attention to special situations that might need to be addressed directly between them and the students. This allows the integration between intelligent tutoring systems and a human tutor.

The main functional requirement that training applications must satisfy in order to allow the management of the corresponding collaboration histories is the support of undo and redo. This allows the teacher to show better solutions to the students once the application has been enhanced with guided collaborative tutoring capabilities, going backwards in the history, if necessary, and producing a new branch in it.

3 FACT

In this section we describe the FACT framework. FACT is a framework implemented in Java that achieves the previous requirements. We shall first give a general view of the mechanisms that it includes, and then we shall give a description of its architecture.

In order to cope with the requirements that have been listed in the previous section, FACT uses two main concepts: histories and sessions. Linear histories are sequences of events that have been performed consecutively by a user on the underlying application. Each linear history can be performed when started at some specific state of the application, which can be either its initial state (initial history) or the state obtained after part of another history is performed. Review histories are trees made with linear histories, one of which is an initial one, while the remaining histories are attached to elements of others in such a way that each history can be performed once those that precede it are performed up to its point of attachment. For example, when using FACT with a chess training application, a simple review history for a chess game can consist of two linear histories; the initial one corresponding to the game as it was played by two students, and a secondary one that starts at a position where one of the students overlooked the possibility to give check mate to his opponent, and includes the moves that are needed to win the game from that position. In some cases this analysis of a winning position will require the presence of more branches in the corresponding history.

On the other hand, a session refers to the work done synchronously during an interval of time by a group of people whose components can vary from one instant to another. Sessions are associated to a history, and at every moment they are associated to a specific node of their corresponding history. We shall show the main features of sessions by means of the example of a chess game given below. In this situation the initial game would be played by the students through one or more sessions, where they might be working synchronously or asynchronously. During each of these sessions the students can make new moves or alternatively review the previous moves; actually, both players can be playing simultaneously and asynchronously in different sessions. Independently of this, the tutor and other authorised students might be looking at the game in different ways. In the first case they can just follow the development of the game: this is a special case of a session that is synchronised to the last state of a linear history. In the second case they can follow the steps (both playing and reviewing) of one of the players by incorporating themselves to the corresponding session. Finally, they can review the game at their own pace by creating their own session; in this case they can also add alternatives to the game.

Developing applications with a framework

This degree of flexibility is achieved by FACT by means of two mechanisms: on the one hand, based upon the availability of undo and redo functionality in the underlying application, FACT allows the enhanced guided tutoring application to have collaborative undo and redo. This is needed in order to be able to go back and forth through the history. On the other hand, all that is needed is the creation of mechanisms for the creation of new sessions, either by separation of a group of users from another one or by simply attaching it to a specific state in the history. Finally, each user in a session can act in three different ways:

- going ahead in the resolution of the problem by adding new actions
- creating alternatives
- exploring the working session and its alternatives.

All other users, in the same session, follow these actions synchronously.

An essential aspect to collaborative guided tutoring is the management of authorisations to modify histories and to make users follow actions synchronously. FACT uses a very simple mechanism in order to handle this issue. First let us mention that every session corresponds to a specific history and every history corresponds to a tutoring system. We shall refer to sessions, histories and tutoring systems as tutoring activities, and we will say that histories are more extensive than sessions and tutoring systems are more extensive than histories. All tutoring activities have a corresponding set of users who can play three possible roles. These roles, ordered by their level, are application user, reviewer and spectator. Independently of this, users can be administrators of any tutoring activities. The capabilities that correspond to each role are as follows:

- Administrators can decide who is allowed to play any of the roles up to his own level for the same tutoring activity. This level cannot be higher for any particular user than the one the user holds for more extensive tutoring activities. The tutor of a tutoring system is an application user and an administrator for it. In general, the creators of histories and sessions are administrators of them.
- Application users of tutoring systems can create histories, and they become application users of them. Similarly, both application users and reviewers of histories can create sessions in which they can play the same role.
- Any person who plays any role for a given tutoring activity can take the same role for any other corresponding activity, of smaller amplitude, that already exists.
- Application users of sessions can participate actively in them by interacting with the application. They are also reviewers of the session.
- Reviewers of sessions can navigate through the corresponding history within the session.
- Spectators of sessions just follow synchronously the actions executed by other users.

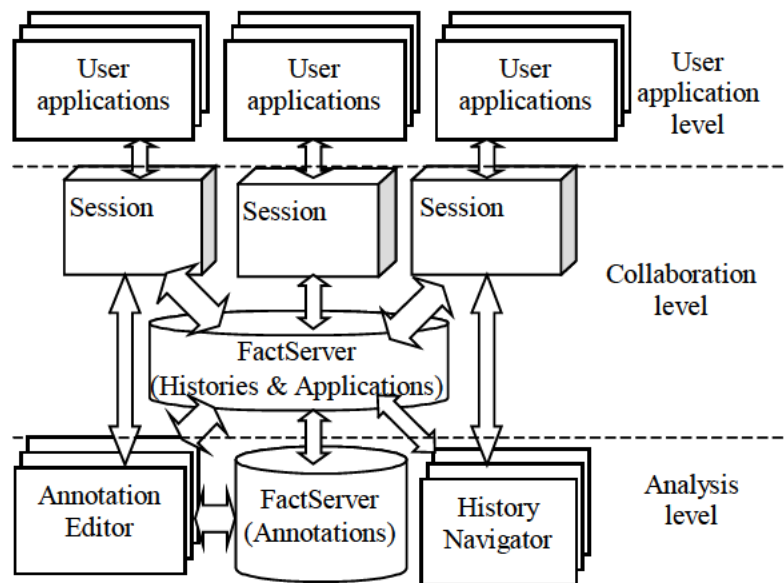
For example, in a calculus teaching session the tutor will be an administrator, and an application user of the tutoring system, and in principle the students will be just application users of the tutoring system. As a consequence of this, students can create histories, and assign any role in their history to other students. These students can later incorporate themselves to an existing session up to the level of that role. In particular, a

student can authorise other students to actively join the resolution of a problem, as well as to review it and to add alternative resolution paths and related annotations; a tutor is always able to do these things, without authorisation.

A more sophisticated authorisation system would also take into account the ownership of secondary histories in order to allow actions on them, as well as deletions. FACT allows deletions of histories only when the role in the history of the user who takes that action is at least as high as that of the user who created the starting point of the history to be deleted. So, for example, when a tutor adds an alternative to a problem, no one can delete it except himself, but when a student who is not the one trying to solve the problem adds an alternative, both the tutor and the student working on the problem can erase it.

All the above facts have implications for the architecture of FACT, that involve a server, generic clients, and an Application Programming Interface (API) that allows the integration of the user application with the rest of the framework; this architecture can be divided into three different functional levels, according to Figure 1, in which the FACT Server appears to be split between the last two levels.

Figure 1 FACT: distributed architecture



- *User application level:* in this level the framework encapsulates the user applications with a collaborative undo/redo abstraction of the application's behaviour.
- *Collaboration level:* this level provides the basic collaboration operations, and includes the functionalities of the FACT server that correspond to histories and applications, as will be explained below. These functionalities include the supervision of users' access and the maintenance of synchronisation among users of the same session. A logger, that is part of the FACT server, stores all the events produced by the different user applications through each session, creating a tree of events.

Developing applications with a framework

- *Analysis level*: this level provides the analysis capability of the history of collaboration produced in the previous level, adding more information, such as annotations, reviewing or creating new branches in such a history. With this kind of applications we can elaborate a problem knowledge database, which can be reused and modified by other students or enriched with new contributions. This functionality is also achieved through the FACT server.

As we have said already, FACT is written in Java, and FACT applications communicate with the server in a transparent way by means of Remote Method Invocation (RMI) [7]. The framework can be used in the development of applications as well as applets, which can be started in a browser through the web.

Each user application, including the annotation editor and the history navigator, communicates with the FACT server, forming groups of users that work synchronously. The FACT server allows the asynchronous work between the groups that correspond to different sessions.

The annotation editor allows the use of annotations, which can be associated with different nodes of the history stored by the server. These nodes can be viewed with a history navigator, which also allows a user to navigate through the history.

In the rest of this section we describe the last two functional levels of FACT in relation to its components.

3.1 Collaboration level

This level provides the collaboration aspects required by the analysis level. FACT is not a generic collaborative framework, but a framework that provides analysis capabilities on top of a specialised collaboration layer, which can be used as a framework in order to provide guided collaborative tutoring. This collaboration layer has some special characteristics, like the use of a logger, that store the events in a tree, rather than in a linear structure. In addition, each node of the tree can be referenced by a label, which allows the connection between the logger and the annotation module.

The logger is the central element in the FACT system, and it is responsible, together with the annotation server, for the asynchronous capabilities of the framework. It stores all the events produced by the different user applications through each session. Special care must be taken in order to ensure that these events accumulate the necessary information to perform do or undo operations in any instance of the user application.

Sessions are also parts in this level. Through each Session, the FACT server supervises users' access and maintains its group of users in synchronisation. This process of synchronisation consists of the redo of the events produced in an instance of the user applications in the rest of the instances connected to the same session server.

A session points to a specific node in the tree-like history. This pointer is moved by the users when a new event is produced or when a user wants to review the history, producing undo or redo events. When a new event is produced, the logger creates a new node in the history, descending from the actual node in the tree, enlarging the actual branch or creating new branches if necessary.

A user can choose between working synchronously with other users, or working alone by choosing the appropriate session or by launching a new one. This is done by a session manager. Thanks to this session manager, dynamic variations of the group, that

constitute each session, are possible. The session manager is also in charge of the definition of access control rules. An example of such a rule is restricting the access to the rest of the users, except for the tutor.

3.2 Analysis level

The analysis level is the fundamental part of FACT from the point of view of the functionality that is available to the user. This level allows the analysis of user interaction with the applications, using generic tools, such as the history navigator or the annotation browser, which can be used as stand-alone applications or included in the user interface of the main application, as components. This level also provides a generic API for application dependent agents to notify users when a special situation arises.

The history navigator borrows events from the logger, shows these events to the user, who can see a brief description of each event, and the annotations associated with it in the annotation browser. In the current version of the framework, users can choose between two versions of the history navigator; one in which the user can see the whole tree that forms the history, and another one in which only a limited number of previous events can be seen, together with their alternatives, if they exist.

The navigator, like the rest of the user applications, is associated to a session server and allows a group of users to navigate synchronously through the history.

The navigation process is very simple: when the user points to a specific node in the learning history the navigator sends to the session server undo and redo events, extracted from the logger, in order to switch to a different state of the application. Users can see the work done by other users because all share the same logger and the same initial state of the application.

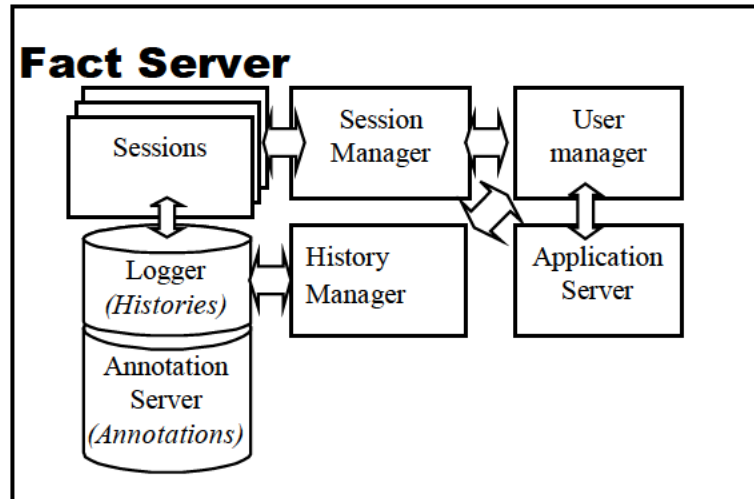
Users can also make annotations, which can be seen by the rest of the users in an asynchronous way. These annotations can be associated to a concrete point of the history in such a way that a user can see the associated annotations while reviewing the history.

The annotation server, which is also part of the FACT server, is the central repository of asynchronous communications between the collaborators. It stores multimedia objects, e.g. html pages, audio, or video. The annotation browsers/editors are synchronised, and the server notifies the clients when a new annotation has been added or deleted, or when the annotation, that a user is viewing, changes.

The annotation editor is like a web browser, which can also edit html pages/annotations. The html text can include audio, video, images, or links to different moments in the history.

Figure 2 shows the different components that form the FACT server.

Figure 2 Architecture of the FACT server

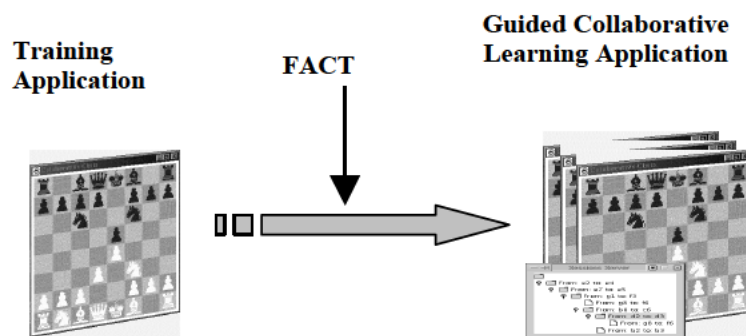


4 Using FACT

We have developed different kinds of applications in order to prove the usability of this framework: firstly, a collaborative puzzle, which demonstrates how to transform a simple application into a collaborative one, supporting guided collaborative tutoring; and secondly, two more complex applications: a chess tutor, ChessEdu [8] and a prototype of a WYSIWYG editor of courses of mathematics (ConsMath) which provides guided collaborative tutoring.

In this section we will describe how to use FACT from the user and developer perspectives. Initially, we will illustrate the developer perspective by describing the application requirements and showing the necessary steps to integrate an existing application into the framework (see Figure 3).

Figure 3 Integrating a training application with FACT



Actually the framework only supports Java applications, so the first requirement is to represent the application using a Java class.

The second requirement is to provide a do/undo abstraction of the application behaviour. This must be accomplished using the `javax.swing.Undo.UndoManager` Java class, which centralises all the user events.

These events must contain enough information to be stored and reproduced after restarting the application, which is the third requirement. This is necessary in order to transmit the events to all instances of the final application, or to store these events in a history.

Special care must be taken with the use of shared resources, such as files or other devices, printers, etc, to avoid conflicts during the simultaneous use of these resources from different instances.

In order to transform the original application into one that is guided collaborative tutoring, just two steps must be taken:

The first one is to substitute the original `UndoManager` with one that is specialised, provided by the framework, which communicates all the user events to the server and distributes these events to all instances in the same session

The final step is to insert the application into the framework, which consists of adding the main class of our Java application to the list of available applications.

From the user perspective, the first step needed to launch the guided collaborative tutoring applications is to start the `FactServer`. After that a user must start a `Fact` client authenticating with the server. The `Fact` client allows them to create new histories or sessions, to join existing ones, or to launch new collaborative applications. At the same time, the `Fact` client provides the user with collaborative analysis tools, such as the annotation editor or the history browser and navigator.

5 Conclusions

We have described the requirements that generic guided tutoring collaborative applications must satisfy, at both the user and functional levels. We have also described `FACT`, a framework for the development of such applications. `FACT` is based on the use of learning histories. From the didactic point of view this allows a tutor to point out to the students their mistakes, their consequences, and related simple situations they should look at in order to correct their mistakes, while retaining the benefits traditionally obtained by group learning. Moreover, by using this kind of application, tutors are able to pay attention to a larger number of students with essentially the same degree of effectiveness.

New Java applications can be developed with this framework, making few modifications to the original ones, consisting basically of creating or replacing the undo/redo mechanism of the application with a specialised one that connects to the `FactServer` and communicates with the user through the interface of the `FactClient`.

References

- 1 Appelt, W. (1999) 'WWW based collaboration with the BSCW system', *Springer Lecture Notes in Computer Science 1725*, Milovy (Czech Republic), pp.66-78.
- 2 NetMeeting Home (n.d.) Available from: <http://www.microsoft.com/windows/netmeeting> [Accessed 1 April 2002],
- 3 Cockburn, A. and Dale, T. (1997) 'CEVA: a tool for collaborative video analysis', *Group'97 Proceedings*, ACM press, pp.47-55.
- 4 Plaisant, Rose, Rubloff, Salter, and Shneiderman (1999) 'The design of history mechanism and their use in collaborative educational simulations', *CSCl'99 Proceedings*.
- 5 Graham, T.C.N. and Grundy, J.C. (1999) 'External requirements of groupware development tools', *Engineering for Human-Computer Interaction*, Dordrecht: Kluwer Pub., pp.363-376.
- 6 Mora, M.A. and Moriyón, R. (2001a) 'Collaborative analysis and tutoring: the fact framework', *Advanced Learning Technology: Issues, Achievements and Challenges Technologies*, Los Alamitos(CA): IEEE Computer Society Press, pp.82-85.
- 7 Daconta, M., Saganich, A. and Monk E. (1999) *Java 2 and JavaScript for C and C++ Programmers*, John Wiley & Son.
- 8 Mora, M.A. and Moriyón R. (2001b) 'Guided collaborative chess tutoring through game history analysis', *Computers and Education. Towards an Interconnected Society*, Dordrecht: Kluwer Academic Publishers, pp.243-250.