JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

Computer-Supported Collaborative Learning
DOI 10.1007/s11412-007-9018-0

Supporting collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams

Nilufar Baghaei · Antonija Mitrovic · Warwick Irwin

Received: 12 March 2007 / Accepted: 16 July 2007

Abstract We present COLLECT-*UML*, a constraint-based intelligent tutoring system (ITS) that teaches object-oriented analysis and design using Unified Modelling Language (UML). UML is easily the most popular object-oriented modelling technology in current practice. While teaching how to design UML class diagrams, COLLECT-*UML* also provides feedback on collaboration. Being one of constraint-based tutors, COLLECT-*UML* represents the domain knowledge as a set of constraints. However, it is the first system to also represent a higher-level skill such as collaboration using the same formalism. We started by developing a single-user ITS that supported students in learning UML class diagrams. The system was evaluated in a real classroom, and the results showed that students' performance increased significantly. In this paper, we present our experiences in extending the system to provide support for collaboration as well as domain-level support. We describe the architecture, interface and support for collaboration in the new, multi-user system. The effectiveness of the system has been evaluated in two studies. In addition to improved problem-solving skills, the participants both acquired declarative knowledge about effective collaboration and did collaborate more effectively. The participants have enjoyed working with the system and found it a valuable asset to their learning.

Keywords Collaboration support · Computer supported collaborative learning · Constraint-based modelling · Evaluation · Intelligent tutoring system · Problem-solving support · UML class diagrams

N. Baghaei (✉) · A. Mitrovic · W. Irwin
Department of Computer Science and Software Engineering, University of Canterbury,
Private Bag 4800, Christchurch, New Zealand
e-mail: nilufar.baghaei@gmail.com

A. Mitrovic
e-mail: tanja@cosc.canterbury.ac.nz

W. Irwin
e-mail: w.irwin@cosc.canterbury.ac.nz

Introduction

Web-based collaborative learning is becoming an increasingly popular educational paradigm as more students who are working or are geographically isolated engage in education. As such, when students do not meet face to face with their peers and teachers the support for collaboration becomes extremely important (Constantino-Gonzalez and Suthers 2002).

In the last decade, many researchers have contributed to the development of computer-supported collaborative learning (CSCL) and advantages of collaborative learning over individualized learning have been identified (Inaba and Mizoguchi 2004). Some of the particular benefits of collaborative problem-solving include: encouraging students to verbalize their thinking; encouraging students to work together, ask questions, explain and justify their opinions; increasing students' responsibility for their own learning; increasing the possibility of students solving or examining problems in a variety of ways; and encouraging them to elaborate and reflect upon their knowledge (Soller 2001; Webb et al. 1995). These benefits, however, are only achieved by active and well-functioning learning teams (Jarboe 1996). Several systems for collaborative learning have been developed, but the concept of supporting peer-to-peer interaction in CSCL systems is still in its infancy. Different strategies for computationally supporting online collaborative learning have been proposed and used, but more studies are needed to examine the utility of these techniques (Jerman et al. 2001).

This paper presents an intelligent tutoring system, the goal of which is to support the acquisition of both problem-solving skills and collaboration skills. We have developed a constraint-based problem-solving environment in which students construct UML class diagrams that satisfy a given set of requirements. It assists students during problem-solving and guides them towards the correct solution by providing feedback. The system is designed as a complement to classroom teaching and when providing assistance, it assumes that the students are already familiar with the fundamentals of UML.

We started by developing a single-user version. Next, we extended the system to support groups of students solving problems collaboratively. All constraint-based tutors developed so far support individual learning, but COLLECT-*UML* is the first to add support for collaborative learning as well. The system provides feedback on both collaboration issues (using the collaboration model, represented as a set of meta-constraints) and task-oriented issues (using the domain model, represented as a set of syntax and semantic constraints).

We start with a brief overview of related work in "Related work." "Single-user version of COLLECT-UML" presents the basic features of the single-user version of COLLECT-UML. The architecture of the collaborative version of the system is discussed in "The architecture of COLLECT-UML," while the following section presents the student interface and justifies the design decisions made. "Modeling collaboration" describes the collaborative model, which has been implemented as a set of meta-constraints. In "Evaluation," we present the results of two evaluation studies performed. "Conclusions" are given in the last section.

Related work

The CSCL systems can be categorized into three main types in the context of the collaboration support (Jerman et al. 2001). The first category includes systems that reflect actions and make the students aware of the participants' activities. Increasing awareness

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

about such actions could help students maintain a representation of other team members' activities and can considerably influence the collaboration (Plaisant et al. 1999). The systems in the second category monitor the state of interactions; some of them aggregate the interaction data into a set of high-level indicators and display them to the participants (e.g., Sharlok II [Ogata et al. 2000]), while others internally compare the current state of interaction to a model of ideal interaction, but do not expose this information to the users (e.g., EPSILON [Soller and Lesgold 2000]). Finally, the third class of systems offer feedback on collaboration. The coach in these systems plays a role similar to that of a teacher in a collaborative-learning classroom. The systems can be categorized by the nature of the information in their models, and if they provide feedback on strictly collaboration issues or both collaboration and task-oriented issues (Jerman et al. 2001). Examples of the systems focusing on the social aspects include Group Leader Tutor (McManus and Aiken 1995) and DEGREE (Barros and Verdejo 2000), while examples of systems addressing both social and task-oriented aspects of group learning are COLER (Constantino-Gonzalez et al. 2003) and LeCS (Rosatelli et al. 2000).

Although many tutorials, textbooks and other resources on UML are available, we are not aware of any attempt to develop a CSCL environment for UML modeling. However, there has been an attempt (Soller and Lesgold 2000) at developing a collaborative learning environment for OO design problems using Object Modeling Technique (OMT), a precursor of UML. EPSILON monitors group members' communication patterns and problem solving actions in order to identify situations in which students effectively share new knowledge with their peers while solving OO design problems. The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities.

Existing approaches to analyzing the collaborative learning interaction

Analyzing the collaborative learning process requires a fine-grained sequential analysis of the group interaction in the context of the learning goals. The following describes five different computational approaches available in the literature for performing such analysis (Soller and Lesgold 2000).

- *Finite state machines*: McManus and Aiken's (1995) Group Leader system compares sequences of students' conversation acts to those allowable in four-finite-state machines developed to monitor discussions about comments, requests, promises, and debates. The Group Leader analyzes sequences of conversation acts, and provides feedback on the students' trust, leadership, creative controversy, and communication skills. For instance, the system might note a student's limited use of sentence openers from the creative controversy category, and recommend the student to use them.
- *Rule learners*: Katz et al. (1999) developed two rule-learning systems, String Rule Learner and Grammar Learner that learn patterns of conversation acts from dialog segments that target specific pedagogical goals. The rule learners were challenged to find patterns in the hand-coded dialogs between expert technicians and students learning electronics troubleshooting skills. The conversations took place within the SHERLOCK 2 environment for electronics troubleshooting.
- *Decision trees and plan recognition*: COLER (Constantino-Gonzales and Suther 2000) coaches students as they collaboratively learn entity-relationship modeling. Decision trees that account for both task-based and conversational interaction are used to dynamically give feedback to the group.

- *Hidden Markov models:* EPSILON (Soller and Lesgold 2000) monitors group members' communication patterns and problem solving actions in order to identify (using machine learning techniques) situations in which students effectively share new knowledge with their peers while solving object-oriented design problems. The system first logs data describing the students' speech acts (e.g., request opinion, suggest, and apologise) and actions (e.g., Student 3 created a new class). It then collects examples of effective and ineffective knowledge sharing and constructs two hidden Markov models that describe the students' interaction in these two cases. A knowledge sharing example is considered effective if one or more students learn the newly shared knowledge (as shown by a difference in pre-/post-test performance), and ineffective otherwise. The system dynamically assesses a group's interaction in the context of the constructed models, and decides when and why the students are having trouble learning the new concepts.

We propose meta-constraints as an effective way of modeling collaboration, as described in detail later in this paper. COLLECT-*UML* is one of the rare systems to provide both domain-level feedback and feedback on collaboration. LeCS (Rosatelli et al. 2000) is another CSCL system that provides both domain-level feedback and collaboration-based feedback. It is a web-based collaborative case study system that can be applied to any domain in which the learning from case studies method is used. The system provides a solution tree, so that the students can visualize the building up of their solution. However, there are several limitations in LeCS: the sentence openers are only intended to facilitate discussion and are not analyzed by the system; the individual work is not assessed, it is only used to generate the solution tree; evaluation of the case study solutions is the task of the case instructor; the domain knowledge concerning the case study is very simple; the information obtained from the chat and text area are not examined and the feedback on collaboration only captures participation and timing.

Single-user version of COLLECT-*UML*

Constraint-based tutors are Intelligent Tutoring Systems (ITS) that use constraint-based modelling (CBM) (Ohlsson 1994) to generate domain and student models. These tutors have been proven to provide significant learning gains for students in a variety of instructional domains. As is the case with other ITSs (Brusilovsky and Peylo 2003), constraint-based tutors are problem-solving environments; in order to provide individualized instruction, they diagnose students' actions and maintain student models, which are then used to provide individualized problem-solving support and generate appropriate pedagogical decisions. Constraint-based tutors have been developed in domains such as SQL (the database query language) (Mitrovic 1998, 2003; Mitrovic and Ohlsson 1999), database modeling (Suraweera and Mitrovic 2002, 2004), data normalization (Mitrovic 2002, 2005), punctuation (Mayo and Mitrovic 2001) and English vocabulary (Martin and Mitrovic 2003). All three database tutors were developed as problem-solving environments for tertiary students (Mitrovic et al. 2004), but the two language tutors are aimed at elementary school children. Students solve problems presented to them with the assistance of feedback from the system.

The domain that COLLECT-*UML* teaches is object-oriented (OO) analysis and design using the Unified Modelling Language (UML). An OO approach to software development is now commonly used (Sommerville 2004), and learning how to develop good quality OO

software is a core topic in Computer Science and Software Engineering curricula. OO systems consist of *classes* (with *structure* and *behavior)*, and *relationships* between them. Relationships have multiplicity and names can be of different types (association, aggregation, composition, inheritance or dependency). In OO analysis and design, these structures exist independently of any programming language, and consequently many notational systems have been developed for representing OO models without the need for source code. UML is the predominant notation in use today.

UML consists of many types of diagrams, but *class diagrams* are the most fundamental for OO modeling, as they describe the static structure of an OO system: its classes and relationships. For readers unfamiliar with OO or UML, class diagrams can be viewed as conceptually akin to the *entity-relationship diagrams* used for data modeling, with support for OO features such as inheritance and methods (Booch et al. 1999).

OO analysis and design can be a very complex task, as it requires sound knowledge of requirements analysis, design and UML. The text of the problem is often ambiguous and incomplete, and students need a lot of experience to be successful in analysis. UML is a complex language, and students have many problems mastering it. Furthermore, UML modeling, like other design tasks, is not a well-defined process. There is no single best solution for a problem, and often there are several alternative solutions for the same requirements. UML is also suitable for discussion due to its open-ended nature.

COLLECT-*UML* concentrates on teaching students how to construct a UML class diagram to represent the OO concepts present in informal textual descriptions of software requirements. This type of exercise has been used successfully for several years in our introductory software engineering course, with the support of human tutors. The system was designed to supplement the existing teaching programme by presenting additional problems and providing automated tutoring.

At the beginning of interaction, a student is required to enter his/her name, which is necessary in order to establish a session. The session manager requires the student modeler to retrieve the model for the student, if there is one, or to create a new model for a new student. Each action a student performs is sent to the session manager, as it has to link it to the appropriate session and store it in the student's log. Then, the action is sent to the pedagogical module. If the submitted action is a solution to the current problem, the student modeler diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module, which generates appropriate feedback.

Students interact with COLLECT-*UML* via its interface (Fig. 1) to view problems, construct UML class diagrams, and view feedback. The top pane contains buttons that allow the student to select a problem, view the history of the session, inspect his/her student model, ask for help, or print the solution. The central part is a Java applet, which shows the problem text and provides the UML modelling workspace. Feedback is presented on the right, while the bottom part allows the student to submit solutions.

The interface is not purely a communication medium: it also serves as a means of supporting problem solving. The interface provides information about the domain of study as it contains a drawing bar with UML constructs. Students can therefore remind themselves of the basic building blocks to use when drawing UML diagrams. In order to draw a UML diagram, the student selects the appropriate drawing tool from the drawing toolbar and then positions the cursor on the desired place within the drawing area.

COLLECT-*UML* contains an ideal solution for each problem, which is compared to the student's solution according to the system's domain knowledge, represented as a set of constraints (Ohlsson 1994). The system's domain model contains a set of 133 constraints defining the basic domain principles, a set of problems and their solutions (Baghaei et al.

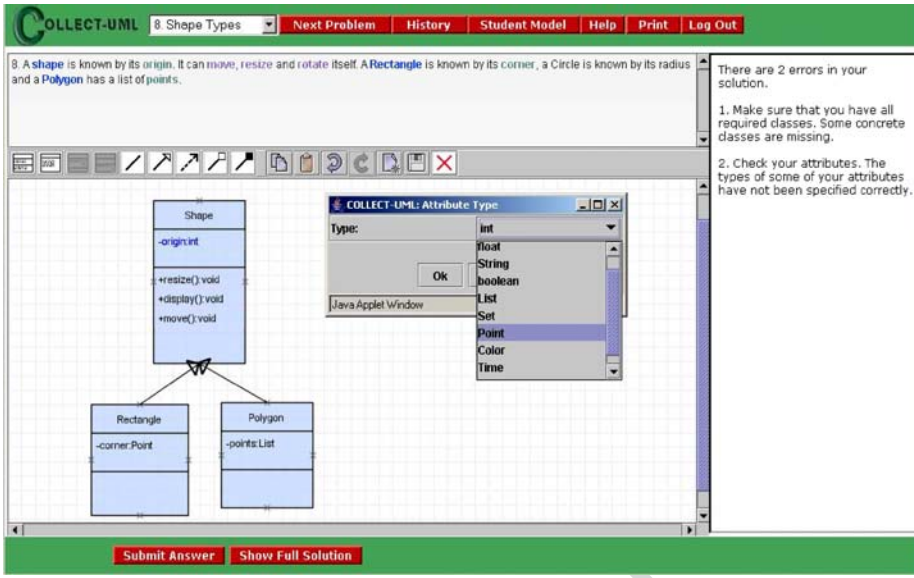


Fig. 1 Single-user version of COLLECT-*UML* interface

2006). Although there is only one solution stored for each problem, the system allows for alternative ways of solving a problem, as there are constraints that check for equivalent constructs between the student solution and the stored solution. In order to develop constraints, we studied material in textbooks (e.g., Fowler 2004) and also used our own experience in teaching UML and OO analysis and design.

Figure 2 illustrates a constraint from the UML domain. The relevance condition identifies a subclass in the ideal solution (IS), and then checks whether the student's solution (SS) contains the same class. The student's solution is correct if the satisfaction condition is met, when the matching class is a subclass of another class. The constraint also contains a message that would be given to the student if the constraint is violated. The last two elements of the constraint specify that it covers some aspects of specialization/generalization, and also identifies the class to which the constraint was applied.

The system was evaluated in a real classroom and the results show that students' performance increased significantly and they enjoyed the user-friendliness and self-learn capability of the system. For details on the architecture, functionality and the evaluation studies of the single-user version please refer to Baghaei and Mitrovic (2005) and Baghaei et al. (2005, 2006).

Fig. 2 Example of a domain constraint

```
(161
  "Check whether you have defined all required subclasses. Some
   subclasses are missing."
  (and (match IS SUBCLASSES (?* "@" ?tag ?*))
       (match SS CLASSES (?* "@" ?tag ?*)))
  (match SS SUBCLASSES (?* "@" ?tag ?*))
  "specialisation/generalisation"
  (?tag))
```

The architecture of COLLECT-*UML*

The collaborative version of the system (Baghaei and Mitrovic 2006) is designed for sessions in which students first solve problems individually and then join into small groups to create group solutions. The system provides support for both phases: during the individual phase, it provides feedback on each individual's solution, while in the group phase it comments on the group solution, comparing it to the solutions of all members of the group and at the same time providing feedback on collaboration.

The collaborative teaching strategy used in COLLECT-UML is based on the socio-cognitive conflict theory (Doise and Mugny 1984). According to this theory, social interaction is constructive only if it creates a confrontation between students' divergent solutions. The system, therefore, tries to create the conditions necessary for effective conflict by identifying the differences between the group solution and individual solutions, making the students aware of the differences and asking them to resolve the conflicts in their solutions, and request and give explanations. There are other CSCL environments in the literature based on socio-cognitive conflict theory, e.g., COLER (Constantino-Gonzalez et al. 2003).

The system's architecture is illustrated in Fig. 3. COLLECT-*UML* is a Web-enabled system and its interface is delivered via a Web browser. The application server consists of a session manager that manages sessions and student logs, a student modeler that creates and maintains student models for individual users, the constraint set, a pedagogical module, and a group modeler, responsible for creating and maintaining group models. The pedagogical module uses both the student model and the collaboration model in order to generate pedagogical actions. The student model records the history of usage for each constraint (both for domain constraints and the constraints from the collaboration model), while the

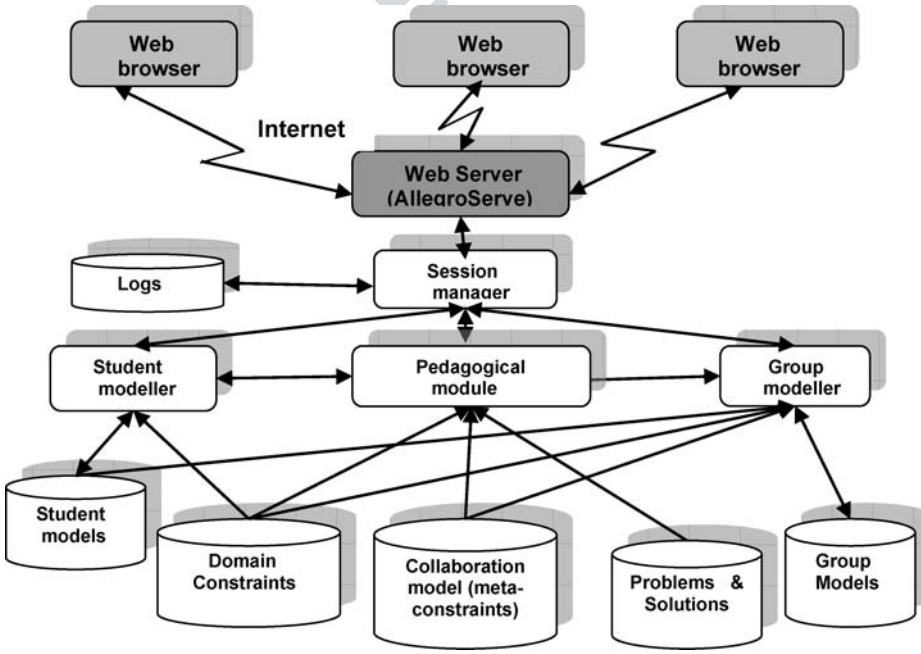


Fig. 3 The architecture of COLLECT-*UML*

group model records the history of group usage for each domain constraint. The system is implemented in WETAS (Martin and Mitrovic 2002, 2003), a constraint-based authoring shell, which provides all tutoring functions such as intelligent analysis of students' solutions, problem/feedback selection and session management. WETAS itself is implemented in Allegro Common Lisp, which provides a development environment with an integrated Web Server (AllegroServe 2006).

The student interface

The student interface is shown in Fig. 4. The problem description pane presents a design problem that needs to be modelled by a UML class diagram. Students construct their individual solutions in the private workspace (right). They use the shared workspace (left) to collaboratively construct UML diagrams while communicating via the chat window (bottom).

The private workspace enables students to try their own solutions and think about the problem before they start discussing it in the group. The group diagram is initially disabled. It is activated after a specified amount of time, and the students can start placing components of their solutions in the shared workspace. This may be done by either copying/pasting from private diagram or by making new components in the group diagram. The private and shared workspaces have been put into split-panes, which would give the

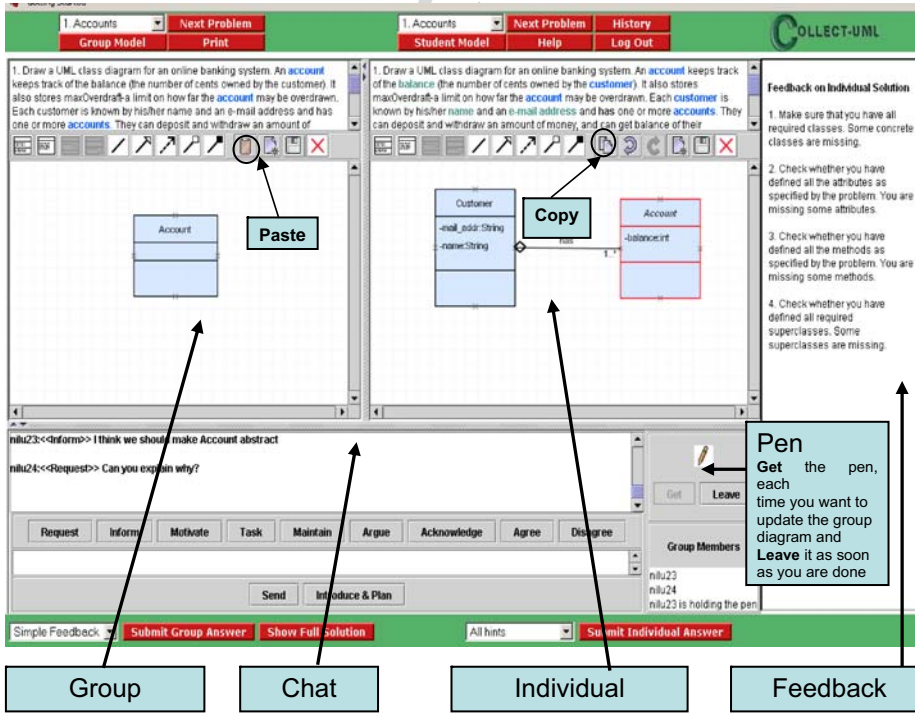


Fig. 4 COLLECT-*UML* interface

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

users the flexibility to resize the areas. The students need to select the components' names from the problem text by highlighting or double-clicking on the words.

The *Group Members* panel shows the team mates already connected. Only one student, the one who has the pen, can update the shared workspace at a given time. The control panel provides two buttons to control this workspace: *Get Pen* and *Leave Pen*. Additionally, this panel shows the name of the student who has the control of this area.

The chat area enables students to express their opinions using one of the communication categories. When a button is selected, the student has the option of annotating his/her selection with a justification. The contents of selected communication categories are displayed in the chat area along with any optional justifications. The students need to select one of the communication categories before being able to express their opinions.

While all group members can contribute to the chat area and the group solution, only one member of the group (i.e., the group moderator) can submit the group solution (by clicking on the *Submit Group Answer* button). The system provides feedback on the individual solutions, as well as on group solutions and collaboration. All feedback messages will appear in the frame located on the right-hand side of the interface.

The domain-level feedback on both individual and group solutions is offered at four levels of detail, upon submission of the solution: *Simple Feedback*, *Error flag*, *Hint* and *All Hints*. The first level of feedback simply indicates whether the submitted solution is correct or incorrect. The *Error flag* indicates the type of construct (e.g., class, relationship, method, etc.) that contains the error. *Hint* offers a feedback message generated from the first violated constraint. A list of feedback messages on all violated constraints is displayed at the *All Hints level*. In addition, the group moderator has the option of asking for the UML class diagram of the complete solution by clicking on *Show Full Solution* button.

Initially, when the student begins to work on a problem, the feedback level is set to the *Simple Feedback* level. As a result, the first time a solution is submitted, a simple message indicating whether or not the solution is correct is given. This initial level of feedback is deliberately low, as to encourage students to solve the problem by themselves. The level of feedback is increased incrementally with each submission until the feedback level reaches the *Hint* level. In other words, if the student/group moderator submits the solutions three times the feedback level would reach the *Hint* level, thus incrementally providing more detailed messages. The system was designed to behave in this manner to reduce any frustrations caused by not knowing how to develop UML diagrams. Automatically incrementing the level of feedback is terminated at the *Hint* level to encourage the student to concentrate on one error at a time rather than all the errors in the solution. The system also gives the student the freedom to manually select any level of feedback according to their needs. This provides a better feeling of control over the system, which may have a positive effect on their perception of the system. In the case when there are several violated constraints and the level of feedback is different from *All hints*, the system will generate the feedback on the first violated constraint. The constraints are ordered in the knowledge base by the human teacher, and that order determines the order in which feedback would be given.

The collaboration-based advice is given to individual students based on the initial planning of the problem, content of the chat area, the student's contributions to the shared diagram and the differences between student's individual solution and the group solution being constructed (Table 1). There are four different time intervals the meta-constraints are evaluated at, which are described later in this document.

The *Next Problem*, *Submit Group Answer*, and *Show Full Solution* buttons associated with the group diagram can be controlled by the moderator only, but the *Group Model*

Table 1 Collaboration-based feedback types t1.1

Feedback Category		Examples of Feedback Messages
Initial Planning	Encouraging Individual Thinking	You may wish to think about the problem and construct a UML diagram in your individual workspace first, before joining the group discussion.
	Encouraging Advanced Planning	Would you like to introduce yourself to your teammates and plan the session?
Use of Communication Categories		You may wish to explain to other members why you agree or disagree with a solution.
		You seem to just agree and/or disagree with other members. You may wish to challenge others ideas and ask for explanation and justification.
		Ensure adequate elaboration is provided in explanations.
Comparing Individual Diagrams with the Group Diagram and vice versa		Some classes in your individual solution are missing from the group diagram. You may wish to share your work by adding those class(es)/discuss it with other members.
		Some methods in the group diagram are missing from your individual solution. You may wish to discuss this with other members.
Contribution to the Group Diagram		You may wish to give explanation and provide justification each time you make a change to the shared diagram.

button can be accessed by all the members to inspect their group model (Fig. 5). The group model visualizes the group's knowledge of the main OO concepts being taught (i.e., classes, attributes, methods, relationships, and specialization) in terms of skill meters, showing how much of the corresponding knowledge they have covered/learned for each concept. It is calculated using the number of satisfied constraints and total number of constraints relevant to each OO concept. The students can use the *Help* button (at the top of the individual workspace) to get information about UML Modeling, *Submit Answer* to get feedback on their individual solutions and *Next problem* to move on to a new problem (regardless of the problem the group is working on at that point). The students cannot view full solutions in the individual workspaces (that option is only available under the shared workspace). Viewing the full solution by individual members of the group might stop them from thinking about the problem and/or collaborating with the rest of the group member.

In the following subsections, we justify some of the design decisions we made in designing the student interface. We discuss the use of communication categories, the importance of turn taking and the inclusion of the private workspace. These justifications are based on the findings of previous research conducted on computer-mediated collaboration.

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

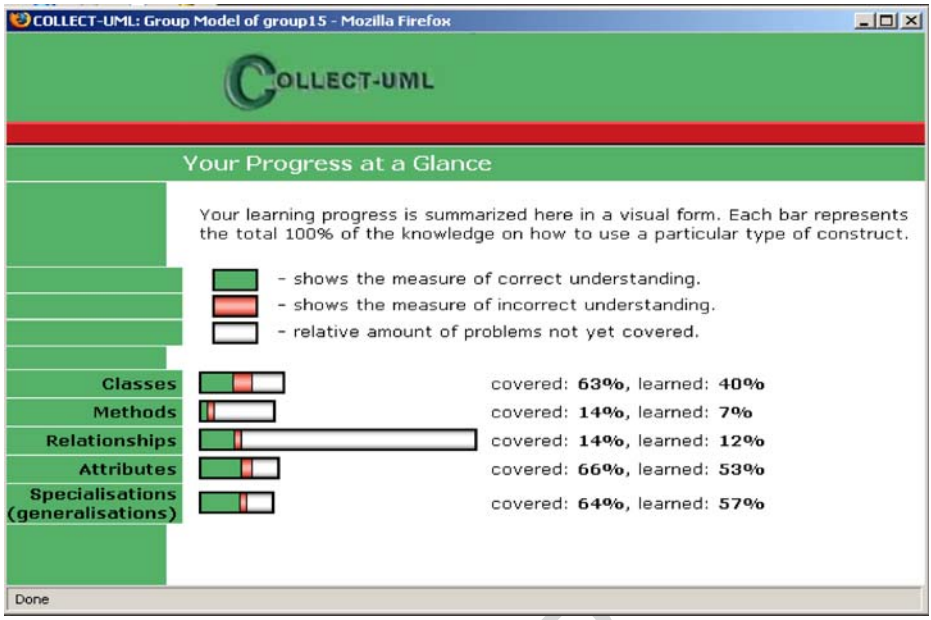


Fig. 5 Open group model

Communication categories

The use of communication categories structures the students' conversation and eliminates off-task discussions. The structured chat interface with specific sentence openers can promote more focus on reflection and the fundamental concepts at stake (Baker et al. 2001). The usage of structured dialogue requires extra effort from students in comparison to free-form input, as students have to find relevant categories for their statements. Although this kind of interaction is slower and more demanding, it structures the data and hence makes it easier to analyze interactions between the group members.

Results from various projects indicate that the use of the structured dialogue "supports and increases learners' task-oriented behavior, leads to more coherence in discussing argumentatively the subject matter, promotes reflective interaction, lightens the learners' typing load, guides the sequence and the content of the dialogue, and is characterized as an adequate pedagogical approach for virtual learning groups" (Gogoulou et al. 2005). However, requiring learners to select a communication category before typing the remainder of their contribution may tempt them to change the meaning of the contribution to fit one of the sentence openers, thus changing the nature of the collaborative interaction.

Finally, it is to be noted that, besides the gains that learners may achieve through a structured dialogue, "this dialogue is also crucial for realizing the benefits of a significant meta-analysis of collaborative students, constituting another advantage of a structured interface" (Dimitracopoulou 2005).

Turn taking

Turn taking is supported in our system by taking and leaving a pen whenever the participants want to make a contribution. A study (Rummel and Spada 2005) has integrated

empirical findings from different research approaches to define relevant characteristics of a good collaboration, and the authors consider turn-taking as one of those characteristics. According to their results, explicitly handing over a turn can be a good way of compensating for the reduced possibilities to transmit nonverbal information.

An implication of providing such a protocol is that deadlocks can be created in cases where one partner cannot proceed with problem-solving alone and at the same time refuses to pass the control over to the other partners. The advantage, however, is that turn taking maintains clear semantics of a participant's actions and roles in the shared workspace (Dimitracopoulou 2005). The lack of providing turn-taking protocol in most of computer-mediated collaboration tools is considered to be one of the limitations of such tools (Feidas et al. 2001).

Private workspace

Providing a well-balanced proportion of individual and joint work phases is considered crucial for successful collaboration in a study by Rummel and Spada (2005). The individual phase allows each group member to use his/her strengths (in terms of domain knowledge and problem-solving skills). This is later followed by a collaborative phase, which includes discussions of various opinions thus supporting information exchange.

Allowing enough time for individual work is of central importance in the case of complementary expertise of the collaborating partners. However, recent studies have provided evidence that individual work is often neglected in studies on computer-mediated collaboration (Hermann et al. 2001). The private workspace also enables students to try solutions without feeling they are being watched (Constantino-Gonzalez et al. 2003). The collaboration scripts developed in the literature (e.g., Dillenbourg 2003) also includes individual activities as well as collective ones, indicating the importance of having an individual work phase.

Modeling collaboration

Research on learning has demonstrated the usefulness of collaboration for improving student's problem-solving skills. When learning in a collaborative setting, students are encouraged to work together, share ideas and their reasoning, ask questions, explain and justify their opinions, and elaborate and reflect upon their knowledge (Webb et al. 1995; Soller 2001). All of these activities increase students' responsibility for their own learning and open up new ways of solving or examining problems. These benefits, however, are only achieved by active and well-functioning learning teams (Jarboe 1996). Simply putting students together and giving them a task does not mean that they will collaborate well. Collaboration is a skill, and, as any other skill, needs to be taught and practiced to be acquired. To work well together, all members need to be active, and need to provide encouragement to each other.

In a recent project, Rummel and Spada (2005) studied the effect of instructional approaches on improving collaborative skills in computer-mediated settings. The authors concluded that "learning by unguided collaborative problem-solving on a task is much less effective than systematic intervention and almost as bad as having no opportunity for learning at all." Students learning via CSCL technology need practice, guidance and support in learning the social interaction skills, just as students learning in the classroom need support from their instructor (Soller 2001).

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

The goal of our research is to support collaboration by modeling collaborative skills. COLLECT-*UML* is capable of diagnosing students' collaborative actions, such as contributions to the chat area and contributions to the group diagram, using an explicit model of collaboration. This collaboration model is represented using constraints, the same formalism used to represent domain knowledge. A significant contribution of our work is to show that constraints can be used not only to represent domain-level knowledge, but also higher-order skills such as collaboration.

Our model of collaboration consists of set of 25 meta-constraints representing ideal collaboration. The structure of meta-constraints is identical to that of domain-level constraints: each meta-constraint consists of a relevance condition, a satisfaction condition and a feedback message. The feedback message is presented when the constraint is violated. In order to develop meta-constraints, we studied the existing literature on characteristics of effective collaboration, such as (Constantino-Gonzalez et al. 2003; Vizcaino 2005; Soller 2001; Rummel and Spada 2005), and also used our own experience in collaborative work.

The meta-constraints are divided into four main groups: constraints that monitor students' contributions to the group diagram (making sure that students remain active, encouraging them to discuss the differences between their individual diagrams and the group diagram, etc.), constraints that monitor students' contributions to the chat area and the use of communication categories, constraints that monitor the differences between the student's individual solution and the group solution, and constraints that monitor the initial planning of tackling the problem. Table 1 shows different categories of meta-constraints with one or more examples for each category.

There are four different time intervals the meta-constraints are evaluated at, which were chosen based on our experience from the pilot study: one-off (e.g., the meta-constraint checking whether the students have introduced themselves to their team-mates and have planned the session and the meta-constraint checking that the student has constructed a diagram in his/her individual workspace before joining the group discussion), 5 min (e.g., asking students to ensure adequate elaboration is provided in their explanations), 8 min (e.g., encouraging students to explain to other members why they agree or disagree with a solution), and 10 min (e.g., encouraging students to contribute to the construction of the group diagram).

Figure 6 illustrates the four meta-constraints. The relevance condition of constraint 227 focuses on methods that are defined for certain classes in the student's individual solution (referred to as SS), when the same classes also exist in the group solution (GS). For this constraint to be satisfied, the corresponding methods should also appear in the group solution. If that is not the case, the constraint is violated, and the student will be given the feedback message attached to this constraint, which encourages the student to discuss those methods with the group, or add them to the group solution. Constraint 229 focuses on the use of communication categories in student's contribution (referred to as SC), checking whether the student has provided any explanation for the changes they have made to the group diagram. Constraint 238 is relevant if the student has made a contribution to the chat area and its satisfaction condition checks whether the student has typed a statement after using any of the available communication categories. If not, it encourages them to provide more explanation as part of their contribution. Constraint 240 is always relevant (because its relevance condition is always true); its satisfaction condition checks whether the student has made any contributions to the elements of the group solution (classes, methods, attributes or relationships), or to the chat area. If that is not the case, the feedback message suggests the student to contribute to the discussion.

```
(227
  "Some methods in your individual solution are missing from the
   group diagram. You may wish to share your work by adding those
   method(s)/discuss it with other members."
  (and (match SS METHODS (?* "@" ?tag ?name ?class_tag ?*))
       (match SS CLASSES   (?* "@" ?class_tag ?*))
       (match GS CLASSES   (?* "@" ?class_tag ?*)))
  (match GS METHODS (?* "@" ?tag ?name2 ?class_tag ?*))
  "methods"
  (?class_tag))

(229
  "You may wish to give explanation and provide justification each time
   you make a change to the shared diagram."
  (or-p (match SC CLASSES (?* "@" ?class_tag ?*))
        (match SC METHODS (?* "@" ?method_tag ?*))
        (match SC ATTRIBUTES (?* "@" ?attr_tag ?*))
        (match SC RELATIONSHIPS (?* "@" ?rel_tag ?*)))
  (and  (match SC DESC (?* "@" ?tag ?*))
        (or-p (match SC DESC (?* "@" "Request" ?*))
              (match SC DESC (?* "@" "Inform" ?*))
              (match SC DESC (?* "@" "Motivate" ?*))
              (match SC DESC (?* "@" "Task" ?*))
              (match SC DESC (?* "@" "Maintenance" ?*))
              (match SC DESC (?* "@" "Argue" ?*))))
  "descriptions"
   nil)

(238
  "Ensure adequate elaboration is provided in explanations."
  (match SC DESC (?* "@" ?tag ?text ?*))
  (not-p (test SC ("null" ?text)))
  "descriptions"
   nil)

(240
    "Would you like to contribute to the group discussion?"
     T
    (or-p (match SC CLASSES (?* "@" ?class_tag ?*))
          (match SC METHODS (?* "@" ?method_tag ?*))
          (match SC ATTRIBUTES (?* "@" ?attr_tag ?*))
          (match SC RELATIONSHIPS (?* "@" ?rel_tag ?*))
          (match SC DESC (?* "@" ?tag ?*)))
    "descriptions"
     nil)
```


Fig. 6 Examples of meta-constraints

In order to be able to evaluate meta-constraints, the system maintains a rich collection of data about all actions students perform in COLLECT-*UML*. After each change made to the group diagram, an XML event message containing the update and the identity (id) of the student who made that change is sent to the server. Each chat event consists of the student id, the type of sentence opener they have used and the content of the message.

Histories of all the contributions made to the shared diagram as well as the messages posted to the chat area are stored on the server. The internal representation consists of seven components (i.e., *Relationships*, *Attributes*, *Methods*, *Classes*, *Superclasses*, *Subclasses* and *Desc*). The *Desc* component (short for Description) includes the student's activities in the chat area during a specified amount of time. The meta-constraints are evaluated against

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

these histories, and feedback is given on contributions that involve adding/deleting/updating components in the shared diagram as well as contributions made to the chat area.

Soller (2001) proposed a collaborative learning model (CL) that identifies the characteristics exhibited by effective learning teams. The five facets of the CL model are participation, social grounding, performance analysis and group processing, application of active learning conversation skills and promotive interaction. The CL model also supports strategies that could be implemented by CSCL systems for helping groups acquire effective collaborative learning skills. COLLECT-UML supports a number of these strategies:

- *Participation* is supported by encouraging students to participate, if they remain inactive for a specified amount of time.
- *Social grounding* is supported by assigning the moderator role to one student in each team. The moderator is responsible for submitting the group solution.
- *Active learning conversation* is supported by providing feedback on collaborative skill usage, storing student and group models and encouraging students to challenge or explain others' ideas.
- *Performance analysis and group processing* is supported by providing feedback on group/individual performance and allowing students to inspect their student/group models (Fig. 5).
- *Promotive interaction* is supported by ensuing adequate elaboration is provided in students' explanations and updating student/group models when students ask for and receive help.

Evaluation

As the credibility of an ITS can only be gained by proving its effectiveness in a classroom environment, we have conducted two evaluation studies with COLLECT-*UML*, described in this section.

Pilot study

We conducted a pilot study in March 2006. The study aimed to discover users' perceptions of various aspects of the system, mainly the quality and usefulness of feedback messages (both task-based and collaboration-based) and the interface.

The participants were 16 postgraduate students enrolled in an Intelligent Tutoring Systems course at the University of Canterbury, whom we divided into eight pairs. The participants had completed a half of the course before the study and were expected to have a good understanding of ITSs. All participants except one were familiar with UML modeling.

The study was carried out in the form of a think-aloud protocol (Ericsson and Simon 1984). This technique is increasingly being used for practical evaluations of computer systems. Although think-aloud methods have traditionally been used mostly in psychological research, they are considered the single most valuable usability engineering method (Nielsen 1993). Each participant was asked to verbalize his/her thoughts while performing a UML modeling task using COLLECT-UML and collaborating with his/her team-mate. Data was collected from video footages of think-aloud sessions, informal discussions after the session and researcher's observations.

The majority of the participants felt that the interface was nicely designed and found the chat tool to be very useful for communicating their ideas. Most of them said that the problems were challenging and seemed to tackle a good range of complexity. A few participants mentioned that they found the interface a bit complicated and needed more time to learn how to use it. In order to name a new component (*class*, *attribute*, *method* or *relationship*), the students were required to highlight phrases from the problem text. Although some participants found this somewhat restrictive initially, they became more comfortable with the interface once they had a chance to experiment with it.

The definitions of concepts used in designing UML class diagrams were included in the Help document, which several participants found quite useful. The majority of the participants felt that the feedback messages helped them understand the domain concepts that they found difficult. Since they spent only about 30–40 min working with the system, they did not pay much attention to the collaboration-based feedback and hence were not able to comment on the quality of such messages.

The participants provided several suggestions, which were used to modify the system after the study. Following the comments some students made on the chat area, the color of the text was changed to make it easier to read. One participant commented that it was possible to paste elements into the group diagram without holding the pen. This error was fixed so that the participants could not make any changes to the group diagram unless they were holding the pen. There were also a few suggestions for further improvement, e.g., being able to copy a group of elements from the individual diagram and paste them into the group diagram (instead of one element at a time), being able to resize the problem text area, asking for the definitions of static elements to be included in the Help document, and asking for the group diagram to be updated more often.

Evaluation study

The evaluation study was carried out at the University of Canterbury in May 2006, after COLLECT-*UML* was enhanced in the light of the findings from the pilot study. The study involved 48 volunteers enrolled in an introductory Software Engineering course. This second-year course teaches UML modelling as outlined by Fowler (2004). The students learned UML modeling concepts during 2 weeks of lectures and had some practice during 2 weeks of tutorials prior to the study.

The study was conducted in two streams of 2-h laboratory sessions over 2 weeks. In the first week, the students filled out a pre-test and then interacted with the single-user version of the system. Doing so gave them a chance to learn the interface and provided us with an opportunity to assess their UML knowledge and decide on the pairs and moderators.

At the beginning of the sessions in the second week, we told students what characteristics we would be looking for in effective collaboration (that was considered as a short training session). The instructions describing the characteristics of good collaboration and the process we expected them to follow (Fig. 7) were also handed out. The idea of providing students with such a script and therefore supporting instructional learning came from a study conducted by Rummel and Spada (2005). The participants were also given a screenshot of the system highlighting the important features of the multi-user interface (Fig. 4).

The students were randomly divided into pairs with a pre-specified moderator. The moderator for each pair was the one who had scored better in the pre-test (filled out in the first week). The pairs worked on a big, relatively complex problem (given in Appendix) individually and joined the group discussion whenever they were ready—the group

Initial Phase

- Introduce yourself to each other
- Decide on how much time you are planning to spend on the individual diagram
- Ask questions about UML if you are not sure about anything (don't talk about the solution though)
- Read the problem text carefully and construct a UML diagram for the problem description in your individual workspace
- The group diagram will be enabled after 10 minutes. After the group diagram is enabled, you can start discussing your solution with other group members (whenever you are ready)

Main Phase

- After the shared diagram gets activated, get the pen (request it if someone else is already holding the pen) and copy and paste a component of your individual diagram to the shared workspace, when the pen is available
- Release the pen as soon as you finish with adding a component to the shared diagram. Don't hold the pen for too long and let other members contribute too
- Compare your individual solution with the group diagram being constructed in the shared workspace. Let the group members know if there is any difference between your solution and the shared solution
- Actively discuss any changes you make to the shared diagram with the other group members. After every change you make to the group diagram, make sure you give explanation and provide justification in the chat area
- After a member makes a change to the shared diagram or suggests something, make sure to express your opinion as to whether or not you agree with it and why
- Ask your team-mate to give explanation and provide justification, if you cannot follow their contribution
- Inform your team member that you read and/or appreciate their comments
- Challenge other members' contributions to the shared diagram and don't accept an idea if you do not agree with it
- Make sure you are contributing to the shared diagram and/or the chat area. Don't just sit there and watch your team-mate solving the problem

Final Phase

- Let the moderator know whether or not you agree with the final diagram before he/she submits it to the system
- Discuss the feedback from the system with each other and modify the shared diagram accordingly
- Move on to the next problem and follow the previous procedure (individual problem-solving, collaborative problem-solving and group agreement on a joint solution)

Fig. 7 Exemplary collaboration

diagram was activated after 10 min. We made sure that the pairs were physically separated, so that they could only communicate through the chat window.

At the end of the session, each participant was asked to complete a post-test, which was used to compare their performance with the pre-test from the previous session. They were also asked to fill out a questionnaire commenting on the interface, the impact of the system on their domain knowledge and their collaborative skills, and the quality of the feedback messages provided by the system on their individual and collaborative activities.

Interacting with the system

The experimental group consisted of 26 students (13 pairs) who received feedback on the domain model as well as their collaborative activities. The control group consisted of 22 students (11 pairs) who only received feedback on the domain model (no feedback on collaboration was provided in this case). There were four female participants in four different pairs (one from the control group and three from the experimental group). Both control and experimental groups received instructions on characteristics of good collaboration at the beginning of the session.

Both versions of the system provided five levels of feedback on students' solutions (*Positive/negative*, *Error Flag*, *Hint*, *All Hints*, *Full Solution*). Table 2 presents some general statistics about the second week of the study. Active pairs are those who collaborated (i.e., contributed to the chat area, the group diagram or both). Out of ten active pairs, six pairs in the control group and eight pairs in the experimental group submitted their group solutions and received feedback from the system. The logs for the other active pairs show that they constructed a group diagram and/or discussed it in the chat area, but the moderators did not submit the final solution. Four pairs in each group managed to solve the problem; two experimental pairs and one control pair got it right on their first submission.

As can be seen from Table 3, the experimental group students contributed more to the group diagram, with the difference between the average number of individual contributions for control and experimental groups being statistically significant (t=2.03, p=0.03). The meta-constraints generated collaboration-based feedback 19.4 times on average for the experimental group. The total amount of time spent interacting with the system was 1.4 h for the control group and 1.3 h for the experimental group.

Pre- and post-test performance

The pre-test and post-test each contained four multiple-choice questions, followed by a question where the students were asked to design a simple UML class diagram. The tests included questions of comparable difficulty, dealing with inheritance and association relationships. The post-test had an extra question, asking the participants to describe the aspects of effective collaborative problem-solving. The mean scores of the pre- and post-test are given in Table 4. The numbers reported for the post-test do not include the collaboration question.

The most important measure of the ITS effectiveness is the improvement in performance. The average mark on the pre-test for the students who participated in the study was 52% for control group and 49% for the experimental group (Table 4). There was no significant difference on the pre-test, meaning that the groups were comparable. The students' performance on the post-test was significantly better for both the control group (t=2.11, p=0.01) and the experimental group (t=2.06, p=0.002). The experimental group,

Table 2 Number of pairs, active pairs and pairs which submitted/ solved the problems

	Control	Experimental
Pairs	11	13
Active pairs	10	10
Pairs submitted solutions	6	8
Pairs solved the problem	4	4

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

Table 3 Number of group submissions, contributions to the group area and total interaction time

	Control		Experimental	
	Average	SD	Average	SD
Group submissions	5.66	6.02	4.62	5.09
Meta-constraints applied	–	–	19.37	9.02
Individual contributions to the group diagram	11.7	8.65	18.72	10.57
Individual contribution to the chat area	22.22	15.33	23.92	11.70
Individual submissions	19.81	20.56	16.40	18.51
Total time (hours)	1.39	0.29	1.27	0.38

who received feedback on their collaboration while working with the system, performed significantly better on the collaboration question (t=2.02, p=0.003), showing that they acquired more knowledge on effective collaboration.

The effect size for the experiment was also calculated. The common method to calculate it in the ITS community is to subtract the control group's mean score from the experimental group's mean score and divide by the standard deviation of the scores of the control group (Bloom 1984). Using this method, the effect size of the system on student's collaboration knowledge is very high:

$$\left(\text{Average collaboration score}_{\text{exp}} - \text{Average collaboration score}_{\text{control}}\right) \Big/ \text{s.d.}_{\text{control}} = 1.3.$$

Learning

We have analyzed the students' individual log files in order to identify how students learn the underlying domain concepts during their interaction with COLLECT-*UML* in the second week. Figure 8 illustrates the probability of violating a domain constraint plotted against the occasion number for which it was relevant, averaged over all domain constraints and all participants in control and experimental groups. The data points show a regular decrease, which is approximated by a power curve with a close fit of 0.78 and 0.85 for the control and experimental groups respectively, thus showing that students do learn constraints over time. The probability of 0.21 for control group violating a constraint on the first occasion of application decreased to 0.09 at its eleventh occasion, displaying a 61.9% decrease in probability. The probability of 0.23 for experimental group violating a constraint on the first occasion of application decreased to 0.12 at its eleventh occasion, displaying a 47.8% decrease in probability.

Table 4 Mean pre- and post-test scores

	Control		Experimental	
	Average (%)	SD (%)	Average (%)	SD (%)
Collaboration	22	22	52	39
Pre-test	52	20	49	19
Post-test	76	25	73	25
Gain score	17	28	21	31

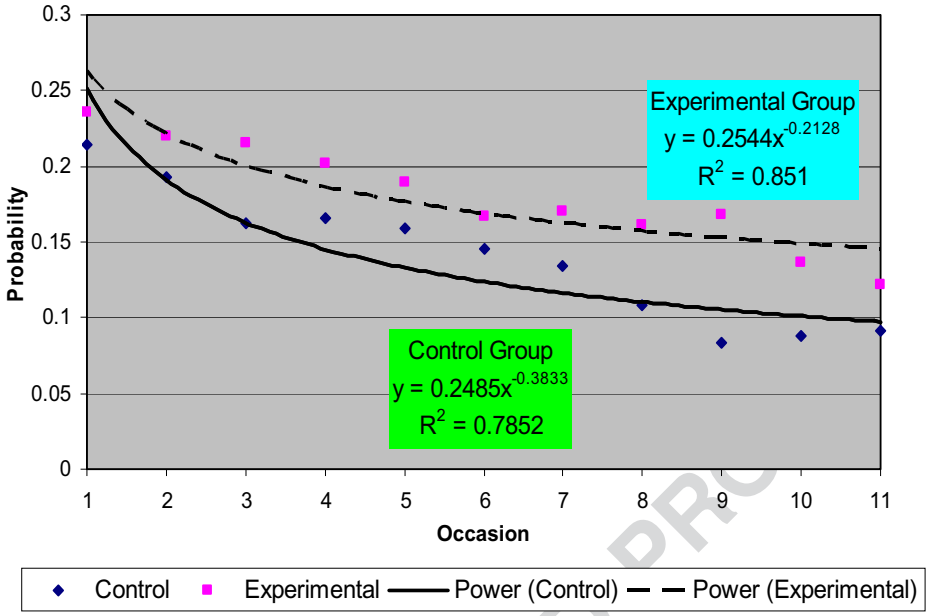


Fig. 8 Probability of domain constraint violation for individuals in control and experimental groups

Figure 9 illustrates the learning curve for meta-constraints only (for the experimental group). The data points show a decrease, which is approximated by a power curve with a R^2 fit of 0.59, initial error probability (0.32) and learning rate (−0.16), thus showing that students learn meta-constraints over time. Because the students used the system for a short time only, more data is needed to analyze learning of meta-constraints, but the trend identified in this study is encouraging. In general, the students violate more task-based constraints than meta-constraints, as there are more domain constraints than meta-constraints.

We found out that 20 domain constraints (out of 76 constraints that were relevant for the problem) were never violated by the participants, meaning that the students already knew the corresponding domain concepts. These constraints can be divided into several groups: (1) constraints that make sure the name of each class or attribute is unique; (2) constraints that check whether classes, attributes, inheritances, compositions and aggregations are represented in the student's solution using appropriate UML constructs; (3) a constraint making sure that each method parameter has a name; (4) a constraint that makes sure each class has at least one attribute or method; (5) constraints that check inheritances in students' diagrams, making sure that there are no cycles; (6) a constraint that makes sure each subclass is connected to a superclass; (7) a constraint that makes sure the right set of classes participate in the associations, and finally (8) constraints that check whether all the superclasses/subclasses are necessary.

The difficult domain constraints (which were violated most often by the participants during their interaction with the system) are the following: (1) a constraint that checks the types of attributes; (2) constraints that check for missing methods, aggregation relationships and abstract classes in the student's solution; (3) constraints that check whether the source and destination multiplicities of the associations have been specified, and finally (4) a constraint that makes sure concrete classes have not been used to represent abstract classes.

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

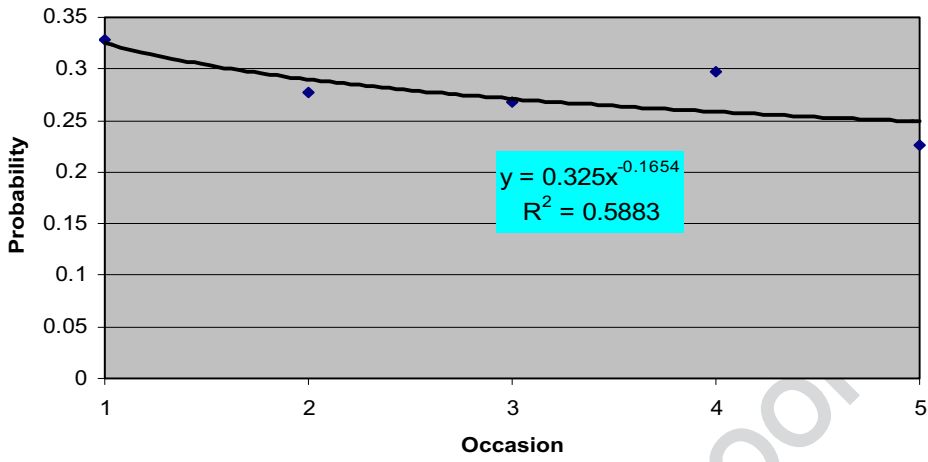


Fig. 9 Probability of meta-constraint violation for the experimental group

In all these cases, the constraints are very specific, and it is likely that the student will focus on these elements of the solution only when the solution is mostly correct.

The easy meta-constraints (violated the least by the students during their interaction with the system) included: (1) a meta-constraint that makes sure students ask for or provide explanations and justifications whenever they (dis)agree with their team mates; (2) a meta-constraint that makes sure adequate elaboration is provided in student's explanations; (3) a meta-constraint that checks whether the student has constructed a diagram in their individual workspace before joining the group diagram, and finally (4) meta-constraints that compare the individual and group workspace checking for missing methods and attributes.

The difficult meta-constraints, which were violated the most by the participants, included the ones checking that the student is contributing to the group discussion and shared diagram (applied every 10 min), and the meta-constraints letting students know that some aggregations, inheritances and classes in the group diagram are missing from their individual solutions and suggesting them to discuss this with other members.

Use of communication categories

Communication categories structure the students' conversation and eliminate the off-task discussions to a great extent. The percentage of off-topic conversations was 3.84% for the control group and 1.55% for the experimental group. The pie charts summarizing the control and experimental groups' interactions are shown in Figs. 10 and 11 respectively. The experimental group was more balanced in this respect, as the students participated more in group maintenance (by using the *Maintain* opener) and task management activity (*Task* opener), requesting information, arguing and disagreeing with other members compared with the control group. *Inform, Acknowledge* and *Introduce and Plan* contributions occurred more in the control group.

Examples of good and bad collaboration

Analysis of session logs shows that four pairs from the control group and seven pairs from the experimental group collaborated well. We chose pair A from the control and pair B

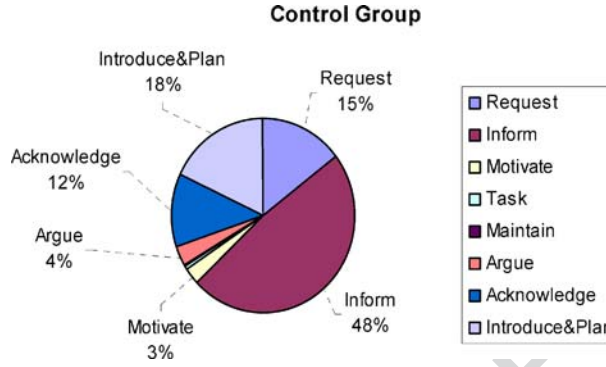


Fig. 10 Use of communication categories by the control group

from the experimental group to illustrate examples of good and bad collaboration. There was no difference between the average pre-test marks of the two pairs (60% for pair A and 58% for pair B). However, pair B did much better on the post-test (average of 85% compared to 60% scored by pair A). Figures 12 and 13 illustrate the probability of domain constraint violation for pairs A and B respectively. As it can be seen, the data points in Fig. 13 show a regular decrease, which is approximated by a power curve with a R^2 fit of 0.87, initial error probability (0. 23) and learning rate (−0.76), thus showing that students learn domain constraints over time, whereas that is not the case for pair A. The probability of 0.3 for pair A violating a constraint on the first occasion has decreased to 0.29 at its seventh occasion, which is almost the same as the initial error probability.

Figures 14 and 15 show the probability of meta-constraint violation for the two members in pair B (pair A did not receive feedback on their collaboration). The data points show a regular decrease, which is approximated by a power curve with a R^2 fit of 0.89/0.85, initial error probability (0.42/0.31) and learning rate (−0.92/−1.2) for members B1/B2 respectively, thus showing that students learn meta-constraints equally well over time.

We also looked at the use of communication categories by the two pairs. Pair B was much more balanced in using different communication categories, while pair A used the *Inform* communication categories extensively and spent very little time on planning the session in advance.

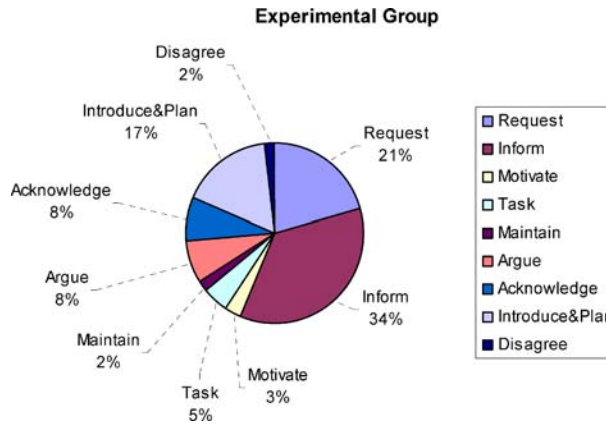


Fig. 11 Use of communication categories by the experimental group

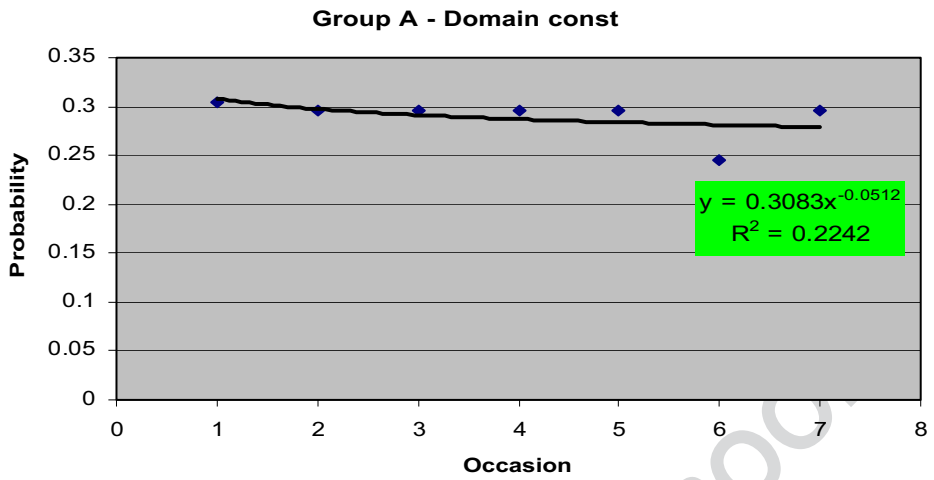


Fig. 12 Probability of domain constraint violation for Pair A

Figures 16 and 17 show timelines of actions performed by each student, where A1 and B1 are moderators. The diamonds represent the contributions to the chat area, the squares represent their contributions to the group diagram and crosses are used to show the moderators asking for feedback on the group diagram. The timelines do not show the activities of the pairs on their individual diagrams.

As it can be seen in Fig. 16, the moderator is much more active than the other student. Since they were part of the control group, they were not receiving feedback on their collaboration activities. We have indicated the parts where getting collaboration feedback would have been useful. For example, collaborative feedback would have been generated at 12:27 asking member A1 to provide an explanation or justification after making a change to the shared area, or at 12:48 asking them to elaborate on their contribution when they used an empty sentence opener. Collaborative feedback could have also encouraged member A2

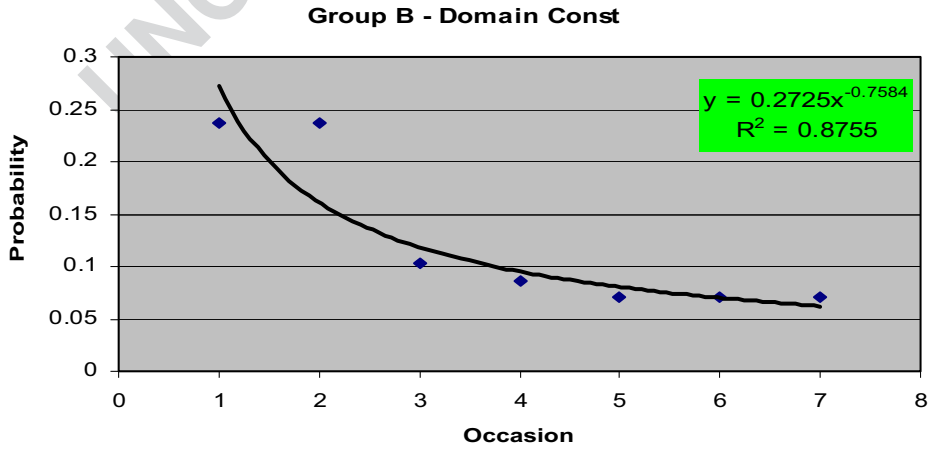


Fig. 13 Probability of domain constraint violation for Pair B

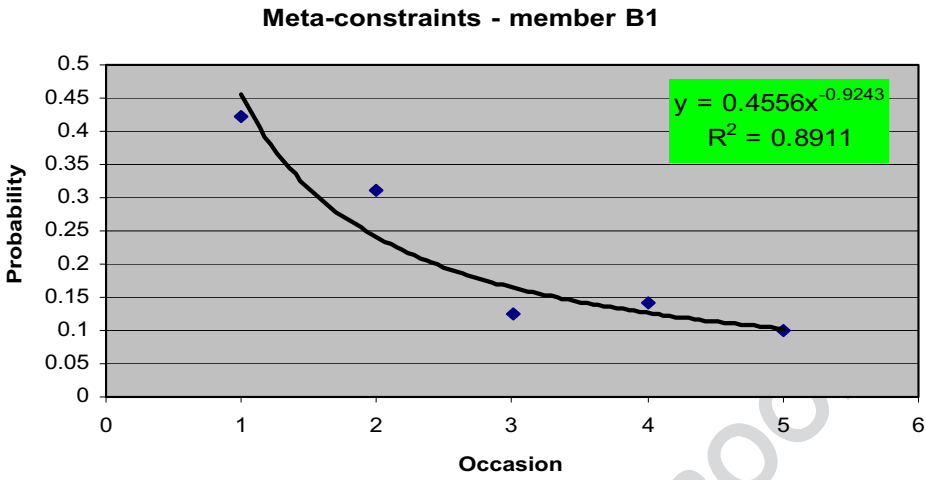


Fig. 14 Probability of meta-constraint violation for Member B1 of Pair B

to be more active and to provide justification each time they made a change to the shared diagram.

Figure 17 shows the summary of collaboration in pair B, including the parts where the students received collaborative feedback. For example, at 12.18 the collaborative feedback encourages member B1 to justify their contributions on the group diagram. At 12:44 and 12:53, the changes (in this case creating a *Transaction* class and aggregation relationships) were explained by using an *Inform* sentence opener. Also at 12:41, member B1 received meta-constraint 223 which states *Some relationship types (aggregations) in your individual solution are missing from the group diagram. You may wish to share your work by adding those aggregation(s)/discuss it with other members.* As we can see, member B1 disagrees with the association created by member B2 at 12:49 (using a *Disagree* sentence opener) and changes the relationship to aggregation instead. The change is then justified by using an *Inform* sentence opener at 12:53.

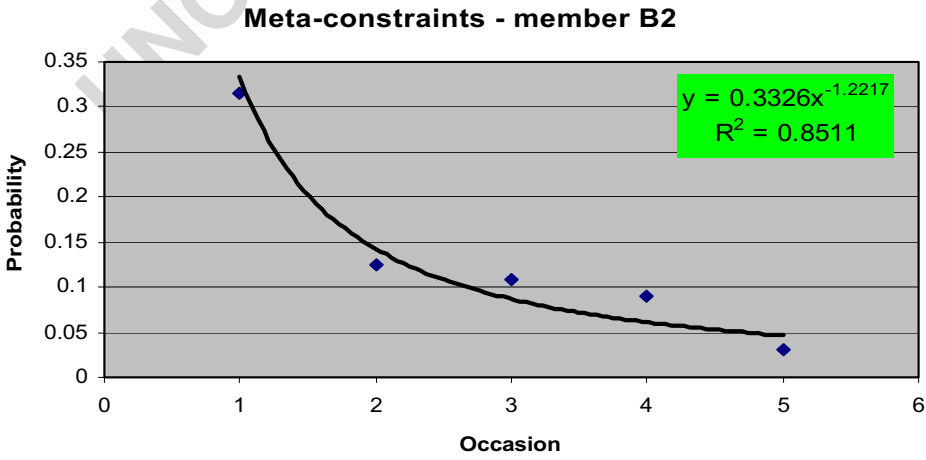
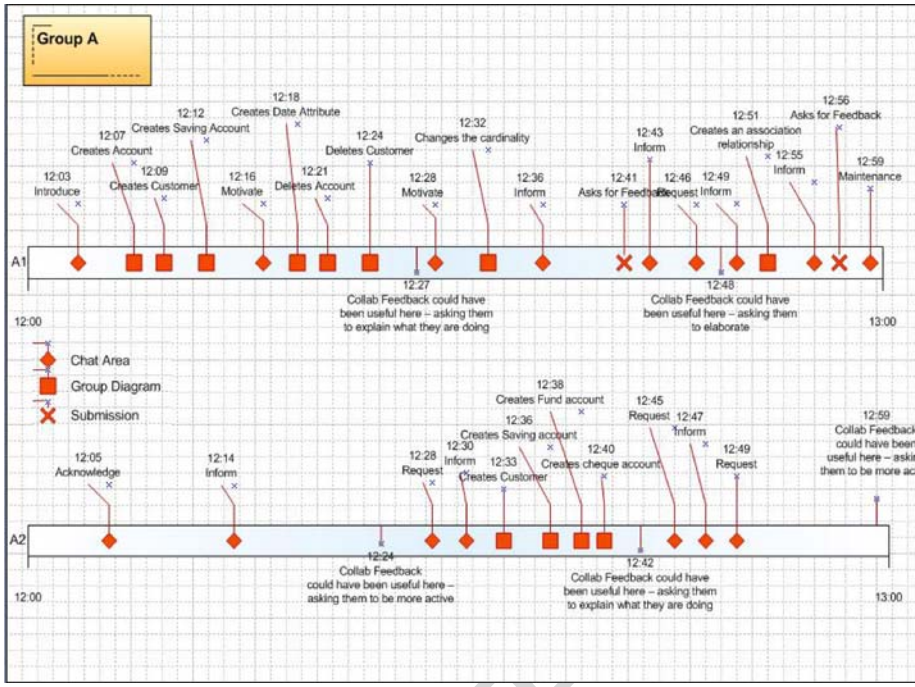


Fig. 15 Probability of meta-constraint violation for Member B2 of Pair B

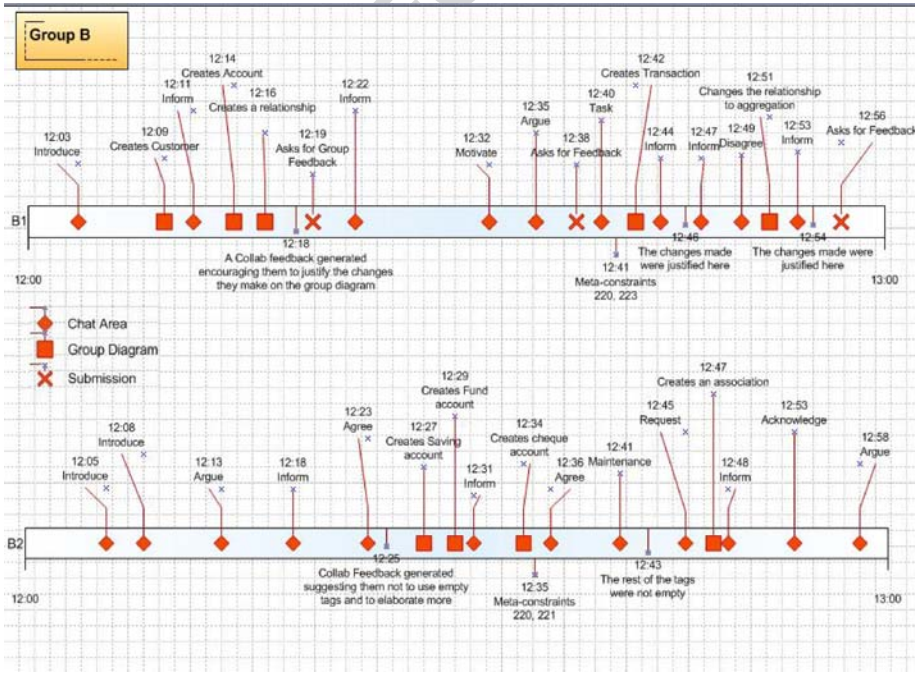
JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007



Q2

Print will be in black and white

Fig. 16 Part of the collaboration log of Pair A (control)



Q2

Print will be in black and white

Fig. 17 Part of the collaboration log of Pair B (experimental)

Examples of collaboration feedback being useful for member B2 is at 12:25 where it encourages him to make sure adequate elaboration is provided when he uses an empty *Agree* sentence opener at 12:23. The student did not use an empty sentence opener from that point on. The same student also created an association at 21:47 following the feedback message received at 12:35 saying *Some relationship types (associations) in your individual solution are missing from the group diagram. You may wish to share your work by adding those association(s)/discuss it with other members.*

We also analyzed collaboration of another pair C from the control group who collaborated effectively compared with other pairs and also did well in the UML diagram. These students had a lower initial error rate on their learning curves. Figure 18 shows the probability of domain constraint violation for pair C. The data points show a regular decrease that is approximated by a power curve with an R^2 fit of 0.91, initial error probability (0. 13) and learning rate (−0.93), thus showing that the members learn domain constraints over time.

Figure 19 shows an excerpt of the collaboration log of pair C. We have highlighted some parts where getting feedback would have made the collaboration process more effective. For instance, at 12:17 a collaboration message could have encouraged member C1 to be more active in the chat area and at 12:55 to give an explanation and provide justification after making changes to the shared diagram. As shown in Fig. 19, member C2 is more active in the chat area and is not making much contribution to the shared diagram, leaving member C1 to make most of the changes. A feedback message could have encouraged him to contribute more to the shared diagram.

Subjective analysis

The participants were given a questionnaire at the end of the session to determine their perceptions of the system. Table 5 presents a summary of the responses. Seventy-three percent of the control group and 41% of the experimental group were familiar with UML

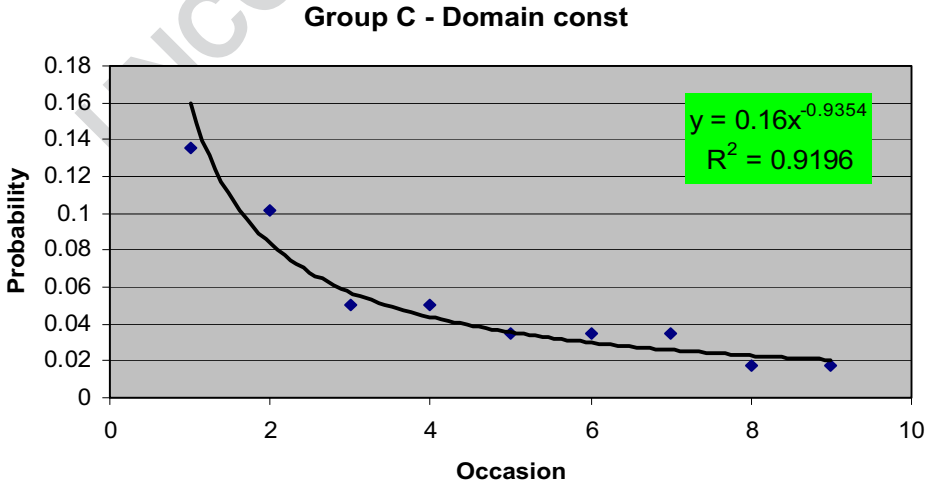
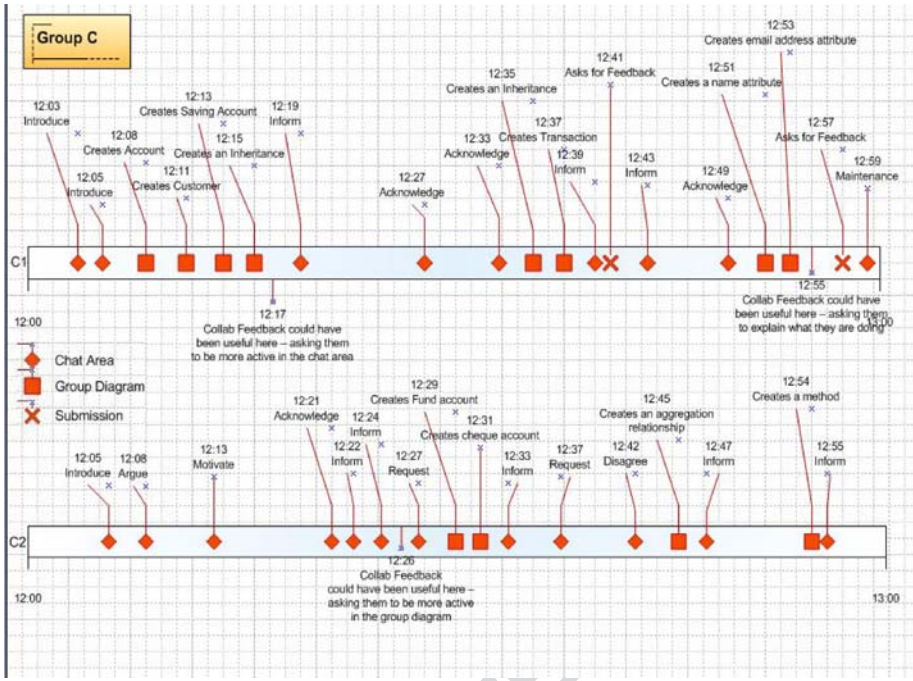


Fig. 18 Probability of domain constraint violation for Pair C

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007



Q2

Fig. 19 Part of the collaboration log of Pair C (control)

modeling from lectures and some work, and the rest had previous experience only from the lectures. Most of the participants (61% of control group and 78% of experimental group) responded they would recommend the system to other students.

The mean responses when asked to rate how much they learned by interacting with COLLECT-*UML* were 2.8 and 3.5 for control and experimental groups respectively, on the scale of 1 (nothing) to 5 (very much). The students found the interface easy to learn and use (the mean responses were 3.4 and 3.0 for control and experimental groups respectively). The majority of participants said they needed 10–30 min to learn the interface and become comfortable using it.

Table 5 Mean responses from the user questionnaire for the evaluation study

	Control		Experimental	
	Average	SD	Average	SD
Amount learnt	2.8	0.9	3.5	0.7
Enjoyment	2.8	1.1	3.3	1.0
Ease of using interface	3.4	1.0	3.0	0.9
Usefulness of partner	3.1	1.4	3.2	1.2
Effect of working in groups	3.6	1.0	3.6	0.9
Usefulness of task-based feedback	3.2	0.8	3.6	0.8
Usefulness of collaboration-based feedback	–	–	3.6	0.7

Students were offered individualized and group feedback on their solutions upon submission. The mean ratings for the usefulness of task-based feedback (given on their UML diagrams) were 3.2 and 3.6 for control and experimental groups respectively, and the mean rating for the usefulness of collaboration-based feedback was 3.6 for experimental group.

Fifty-five percent of the control participants and 59% of experimental participants had indicated that they would have liked to see more details in the feedback messages, whereas the rest of the participants mentioned that they had been provided with enough details and more details would have taken away the task of thinking/problem solving. Several participants asked for more problems to be included in the system. The comments we received on open questions show that the students liked the system and thought it improved their knowledge, and also pointed out several possible improvements.

Discussion

The results show that meta-constraints are an effective way of modeling collaboration. The students' declarative knowledge of collaboration increased after the study: the experimental group (who received feedback on their collaboration) scored significantly higher when asked to describe effective collaborative problem solving. The learning curves also prove that student's domain knowledge increases, as they learn constraints during problem solving. All participants performed significantly better on the post-test after short sessions with the system, suggesting that they acquired more knowledge in UML modeling. Subjective evaluation shows that most of the students felt working in groups helped them learn better and that they found the system to be easy to use.

The questionnaire responses suggested that most participants appreciated the feature of being able to view the complete solution and found the hints helpful. Responses showed that the participants found the problems challenging and enjoyed the user friendliness and learning support of the system. There were a few suggestions for further improvement.

There were other encouraging signs suggesting that COLLECT-*UML* was an effective teaching tool. A number of students who participated in the study inquired about the possibility of using COLLECT-*UML* after the study, for practicing UML modeling and preparing for the exam.

Conclusions

The paper discussed the design and implementation of COLLECT-*UML*, a CSCL environment developed to teach students effective collaboration and UML modeling. We presented the system's architecture, interface and functionality. COLLECT-*UML* provides task-based feedback on students' and group solutions as well as collaboration-based feedback intended to make the collaboration process more effective. The collaborative feedback is provided by analyzing students' activities and comparing them to an ideal model of collaboration. COLLECT-UML is one of the rare CSCL systems to provide both domain-level feedback and feedback on collaboration. A significant contribution of the reported work is showing that constraints can be used not only to represent domain knowledge (and the student's model), but are also effective in representing models of meta-cognitive skills.

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

The system's effectiveness in teaching good collaboration and UML class diagrams was evaluated in two classroom experiments. The results of both subjective and objective analysis proved that COLLECT-UML is an effective educational tool:

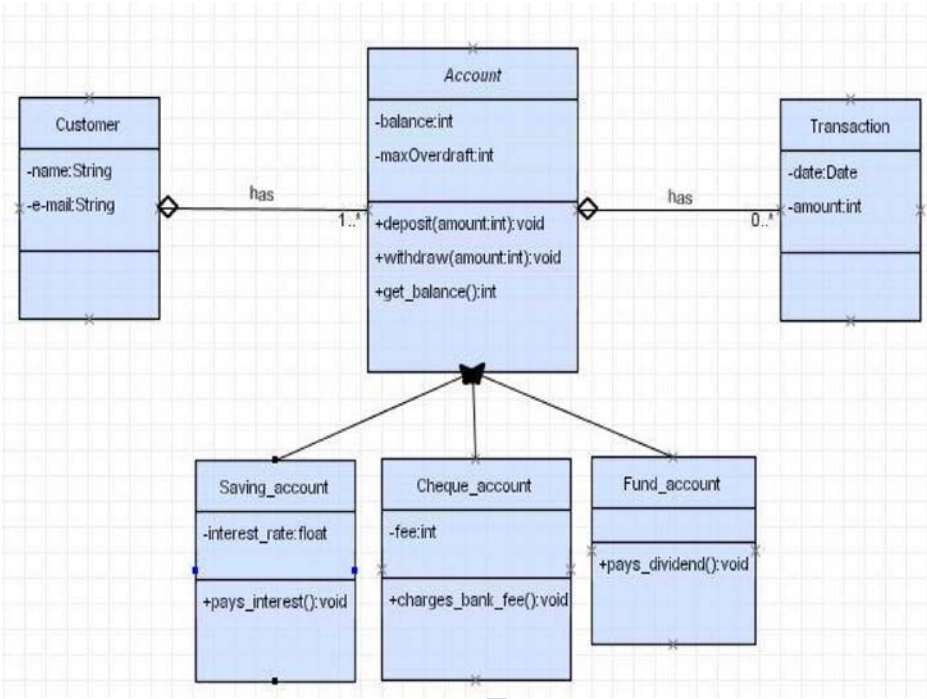
1. The experimental group students acquired more declarative knowledge on effective collaboration, as they scored significantly higher on the collaboration test, with the effect size of 1.3.
2. The collaboration skills of the experimental group students were better, as evidenced by these students being more active in collaboration, and contributing more to the group diagram. The difference between the average number of individual contributions for the control and experimental groups is statistically significant.
3. The experimental group pairs were more balanced in using the various communication categories and had less off-topic conversations.
4. All students improved their problem-solving skills: the participants from the both control and experimental group performed significantly better on the post-test after short sessions with the system, showing that they acquired more knowledge in UML modeling.
5. The students enjoyed working with the system and found it a valuable asset to their learning.

Rummel and Spada's (2005) study shows that groups who collaborated more effectively outperformed their control counterparts on knowledge about aspects of a good collaboration and knowledge about important elements of the domain knowledge (therapy plan). In our full evaluation study, the participants spent less than 1.4 h, on average, interacting with the system and both control and experimental groups were provided with collaborative problem-solving setting and domain-level feedback. Both groups were shown to improve learning. More research is needed to investigate the effect of collaboration-based feedback on learning the domain knowledge.

CBM has previously been used to effectively represent domain knowledge in several ITSs supporting individual learning. The contribution of this research is the use of CBM to model collaboration skills, not only domain knowledge. The results show that CBM is indeed an effective technique for modeling and supporting collaboration in computer-supported collaborative learning environments.

Appendix: Given problem

Draw a UML class diagram for an online banking system. An account keeps track of the balance (the number of cents owned by the customer). It also stores maximum overdraft, a limit on how far the account may be overdrawn. Each customer is known by his/her name and an e-mail address and has one or more accounts. They can deposit and withdraw an amount of money, and can get balance of their accounts. A saving account pays interest and records the interest rate. A checking account charges bank fees and records the amount of fees charged. A fund account pays dividends. An account may have a number of transactions. For each transaction made, the software records the date and the amount. Assume that all the numbers, except for the interest rate, are integers.



Q2

Print will be in black and white

842

References

843

AllegroServe—a Web application server. Retrieved 21.8.2006 from http://www.franz.com.

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based tutor. In R. Khosla, R. Hewlett & L. Jain (Eds.) *Proc. KES 2005* (pp. 458–464). New York: Springer.

Baghaei, N., & Mitrovic, A. (2006). A constraint-based collaborative environment for learning UML class diagrams. In M. Ikeda, K. Ashley & T. W. Chan (eds.) *Proc. ITS 2006* (pp. 176–186).

Baghaei, N., Mitrovic, A., & Irwin, W. (2005). A constraint-based tutor for learning object-oriented analysis and design using UML. In C. Looi, D. Jonassen, & M. Ikeda (Eds.) *Proc. ICCE 2005*(pp. 11–18).

Baghaei, N., Mitrovic, A., & Irwin, W. (2006). Problem-solving support in a constraint-based utor for UML class diagrams. *Technology, Instruction, Cognition and Learning Journal, 4*(2) (in press).

Baker, M., de Vries, E., Lund, K., & Quignard, M. (2001). Computer-mediated epistemic interactions for co-constructing scientific notions: Lessons learned form a five-year research program. In P. Dillenbourg, A. Eurelings, & K. Hakkarainnen (Eds.), *European Perspectives on CSCL (CSCL 2001)*. Maastricht, Netherlands, 2001.

Baker, M. J., & Lund, K. (1997). Promoting reflective interactions in a computer-supported collaborative learning environment. *Journal of Computer Assisted Learning, 13*(3), 175–193.

Barros, B., & Verdejo, M. F. (2000). Analysing student interaction processes in order to improve collaboration: The DEGREE approach. *Artificial Intelligence in Education, 11*, 221–241.

Bloom, B. S. (1984). *The 2-sigma problem:* The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13*, 4–16.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999) *The unified modelling language user guide*. Reading: Addison-Wesley.

Brusilovsky, P., & Peylo, C. (2003). Adaptive and intelligent Web-based educational systems. *Artificial Intelligence in Education, 13*, 159–172.

Q3

Q4

JrnlID 11412_ArtID 9018_Proof# 1 - 02/08/2007

Constantino-Gonzalez, M., & Suthers, D. (2002). Coaching collaboration in a computer mediated learning environment. In G. Stahl (Ed.), *Computer support for collaborative learning: Foundations for a CSCL Community. Proceedings of CSCL 2002* (pp. 583–584). Hillsdale, NJ: Lawrence Erlbaum Associates.

Constantino-Gonzalez, M. A., Suthers, D., & Escamilla de los Santos, J. (2003). Coaching web-based collaborative learning based on problem solution differences and participation. *Artificial Intelligence in Education, 13*(2–4), 263–299.

Dillenbourg, P. (2003). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. In A. P. Kirschner (Ed), *Three worlds of CSCL. Can we support CSCL* (pp. 61–91). Heerlen: Open Universiteit Nederland.

Dimitracopoulou, A. (2005). Designing collaborative learning systems: Current trends & future research agenda. In T. Koschmann, D. D. Suthers, & T. W. Chan (Eds.), *Proceedings of CSCL 2005. Computer support for collaborative learning: The Next 10 Years!* (pp. 115–124). Mahwah, NJ: Lawrence Erlbaum Associates.

Doise, W., & Mugny, G. (1984). The social development of the intellect. *International Series in Experimental Social Psychology, 10*. London: Pergamon Press.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Feidas, C., Komis, V., & Avouris, N. (2001). Design of collaboration-support tools for group problem solving. In N. Avouris & N. Fakotakis (Eds.), *Advances in Human–Computer Interaction* (pp. 263–268). Patras, Greece.

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modelling language*. Reading: Addison-Wesley, 3rd edition.

Gogoulou, A., Gouli, E., Grigoriadou, M., & Samarakou, M. (2005). ACT: A Web-based adaptive communication tool. In T. Koschmann, D. D. Suthers, & T. W. Chan (Eds.), *Proceedings of CSCL 2005. Computer support for collaborative learning: The Next 10 Years!* (pp. 180–189). Mahwah, NJ: Lawrence Erlbaum Associates.

Hermann, F., Rummel, N., & Spada, H. (2001). Solving the case together: The challenge of net-based interdisciplinary collaboration. In P. Dillenbourg, A. Eurelings & K. Hakkarainnen (Eds.), *First European Conference on Computer-Supported Collaborative Learning* (pp. 293–300). Maastricht, Netherlands.

Inaba, A., & Mizoguchi, R. (2004). Learners' roles and predictable educational benefits in collaborative learning; An ontological approach to support design and analysis of CSCL. In J. Lester, R. M. Vicari & F. Paraguacu (Eds.) *ITS 2004* (pp. 285–294).

Jarboe, S. (1996). Procedures for enhancing group decision making. In B. Hirokawa & M. Poole (Eds.), *Communication and Group Decision Making* (pp. 345–383). Thousand Oaks, CA: Sage Publications.

Jerman, P., Soller, A., & Muhlenbrock, M. (2001). From mirroring to guiding: A review of state of the art technology for supporting collaborative learning. In P. Dillenbourg, A. Eurelings & K. Hakkarainen (Eds.) *European Perspectives on CSCL (CSCL 2001)* (pp. 324–331).

Jermann, P., Soller, A., & Lesgold, A. (2004). Computer software support for CSCL. In P. Dillenbourg (Ed.) and Strijbos J. W., Kirschner, P. A. and Martens R. L. (Vol. Eds.), *Computer-supported collaborative learning*: Vol 3. *What we know about CSCL. and implementing it in higher education* (pp. 141–166.). Kluwer Academic Publishers: Boston, MA. Q4

Katz, S., Aronis, J. & Creitz, C. (1999) Modeling pedagogical interactions with machine learning. *Proc. 9th International Conference on Artificial Intelligence in Education* (pp. 543–550.). LeMans, France.

Lazonder, A., Wilhelm, P., & Ootes S. (2003). Using sentence openers to foster student interaction in computer-mediated learning environments. *Computers & Education, 41*, 291–308. Q4

Martin, B., & Mitrovic, A. (2002) Authoring Web-based tutoring systems with WETAS. In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson & C.-H. Lee (Eds.) *ICCE 2002* (pp. 183–187).

Martin, B., & Mitrovic, A. (2003). Domain modeling: art or science? In U. Hoppe, F. Verdejo & J. Kay (Eds.) *Proc. 11th Int. Conference on Artificial Intelligence in Education* (pp. 183–190). Amsterdam: IOS Press.

Mayo, M., & Mitrovic, A. (2001). Optimising ITS behaviour with Bayesian networks and decision theory. *Artificial Intelligence in Education, 12*(2), 124–153.

McManus, M., & Aiken, R. (1995). Monitoring computer-based problem solving. *Int. Journal of Artificial Intelligence in Education, 6*(4), 307–336.

Mitrovic, A. (1998). Learning SQL with a Computerised Tutor. *29th ACM SIGCSE Technical Symposium* (pp. 307–311).

Mitrovic, A. (2002). NORMIT, a Web-enabled tutor for database normalization. In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, & C.-H. Lee (Eds.) *Proc. International Conference on Computers in Education* (pp. 1276–1280). Los Alamitos, CA: IEEE Computer Society.

Mitrovic, A. (2003). An intelligent SQL tutor on the Web. *Artificial Intelligence in Education, 13*(2–4), 173–197.
Mitrovic, A. (2005). The effect of explaining on learning: A case study with a data normalization tutor. In C.-K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) *Proc. 12th Int. Conf. Artificial Intelligence in Education* (pp. 499–506). Amsterdam: IOS Press.
Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-based tutors: A success story. In L. Monostori, J. Vancza, & M. Ali (Eds.), *Proc. 14th Int. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-2001* (pp. 931–940). Berlin: Springer.
Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language. *Artificial Intelligence in Education, 10*(3–4), 238–256.
Mitrovic, A., Suraweera, P., Martin, B., & Weerasinghe, A. (2004). DB-suite: Experiences with three intelligent, Web-based database tutor. *Journal of Interactive Learning Research, 15*(4), 409–432.
Nielsen, J. (1993). *Usability engineering*. San Diego, CA: Academic.
Ogata, H., Matsuura, K., & Yano, Y. (2000). Active knowledge awareness map: Visualizing learners activities in a Web based CSCL environment. *Int. Workshop on New Technologies in Collaborative Learning* (pp. 89–97).
Ohlsson, S. (1994). Constraint-based student modelling. In J. Greer & G. McCalla (Eds.) *Student modelling: the key to individualized knowledge-based instruction* (pp. 167–189), Berlin: Springer.
Plaisant, C., Rose, A., Rubloff, G., Salter, R., & Shneiderman, B. The design ofhistory mechanisms and their use in collaborative educational simulations. *3rd International Conference on Computer Support for Collaborative Learning (CSCL 1999)* (pp. 348–359).
Rosatelli, M., Self, J., & Thirty, M. (2000). LeCS: A collaborative case study system. *Proc. 5th International Conference on Intelligent Tutoring Systems (ITS 2000)* (pp. 242–251).
Rummel, N., & Spada, H. (2005). Learning to collaborate: An instructional approach to promoting collaborative problem-solving in computer-mediated settings. *Journal of the Learning Sciences, 14*(2), 201–241.
Soller, A. (2001). Supporting social interaction in an intelligent collaborative learning system. *International Journal of Artificial Intelligence in Education, 12*, 40–62.
Soller, A., & Lesgold, A. (2000). Knowledge acquisition for adaptive collaborative learning environments. *AAAI Fall Symposium: Learning How to Do Things*, Cape Cod, MA.
Sommerville, I. (2004) *Software engineering*. Pearson/Addison-Wesley, 7th ed.
Suraweera, P., & Mitrovic, A. (2002). KERMIT: A Constraint-based tutor for database modeling. In S. Cerri, G. Gouarderes & F. Paraguacu (Eds.) *ITS 2002* (pp. 377–387).
Suraweera, P., & Mitrovic, A. (2004). An intelligent tutoring system for entity relationship modelling. *Artificial Intelligent in Education, 14*(3–4), 375–417.
Vizcaino, A. (2005). A simulated student can improve collaborative learning. *International Journal of Artificial Intelligence in Education, 15*, 3–40.
Webb, N. M., Troper, J. D., & Fall, R. (1995). Constructive activity and learning in collaborative small groups. *Journal of Educational Psychology, 87*, 406–423.