# A COLLABORATIVE CONSTRAINT-BASED INTELLIGENT SYSTEM FOR LEARNING OBJECT-ORIENTED ANALYSIS AND DESIGN USING UML

---

A THESIS
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
IN THE
UNIVERSITY OF CANTERBURY
by

Nilufar Baghaei

---

**Examining Committee**

| | |
|---|---|
| Associate Professor Antonija Mitrovic | Supervisor |
| Doctor Warwick Irwin | Associate Supervisor |
| Associate Professor Peter Brusilovsky (University of Pittsburgh) | Examiner |
| Doctor Peter Andreae (Victoria University of Wellington) | Examiner |

University of Canterbury
2007

*This thesis is dedicated to my family.*

# Acknowledgements

I would like to thank my supervisor, Assoc. Prof. Dr. Antonija Mitrovic, for her invaluable support and guidance during my research and reading drafts of this thesis. I am highly thankful to my associate supervisor, Warwick Irwin, for helping me run the experiments and providing valuable feedback. Thanks also to Assoc. Prof. Dr. Andy Cockburn for his helpful comments and suggestions on the HCI aspects of the project.

I wish to thank my loving and patient husband for supporting and encouraging me during my study and for taking an active interest in my research. Thanks also to my sister and brother for their support and understanding. But most of all, I am extremely grateful to my parents for their unconditional love and support throughout my life and for risking and sacrificing all to give me a brighter future. Without their enthusiastic support and continuous encouragement, I would have never been able to stand at this point in my life.

During my time as a student, I published a number of papers based on the research and ideas presented here. I would like to thank the anonymous reviewers for their helpful comments and criticism. Thanks to the 224 and 420 students who participated in the experiments during 2005 and 2006 and provided me with useful feedback on different aspects of the system. I am also grateful to the past and present members of the ICTG group for the helpful discussions and for making my PhD study a pleasant and memorable experience.

This research was funded by University of Canterbury Doctoral Scholarship and New Zealand Federation of Graduate Women Fellowship.

# Abstract

Web-based collaborative learning is becoming an increasingly popular educational paradigm as more individuals who are geographically isolated seek higher education. As such students do not meet face to face with their peers and teachers, support for collaboration becomes extremely important. Successful collaboration means asking questions to gain a better understanding of the main concepts, elaborating and justifying opinions and sharing and explaining ideas. When group members' combined skills are sufficient to complete the learning task, effective group work can result in greater overall achievement than individual learning.

Intelligent Tutoring Systems (ITS) have been shown to be highly effective at increasing students' performance and motivation. They achieve their intelligence by representing pedagogical decisions about how to teach as well as information about the learner. Constraint-based tutors are a class of ITSs that use Constraint-based Modelling (CBM) to represent student and domain models. Proposed by Ohlsson, CBM is based on learning from performance errors, and focuses on correct knowledge.

In this thesis, we present COLLECT-$\mathcal{UML}$, a collaborative constraint-based ITS that teaches object-oriented analysis and design using Unified Modelling Language (UML). While teaching how to design UML class diagrams, COLLECT-$\mathcal{UML}$ also provides feedback on collaboration. Being a constraint-based tutor, COLLECT-$\mathcal{UML}$ represents the domain knowledge as a set of syntax and semantic 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. We then extended the system to provide support for collaboration as well as domain-level support. The enhancement process included implementation of the shared workspace, modification of the pedagogical module to support groups of users, designing and implementing a

group-modelling component, and developing a set of meta-constraints which are used to represent an ideal model of collaboration. The effectiveness of the system was 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 enjoyed working with the system and found it a valuable asset to their learning. The results, therefore, show that CBM is an effective technique for modelling and supporting collaboration in computer-supported collaborative learning environments.

# Table of Contents

# Chapter 1

# Introduction

To have joy one must share it. Happiness was born a twin.
— Indian Proverb

Web-based collaborative learning is becoming an increasingly popular educational paradigm as more individuals who are working or are geographically isolated seek higher education. As such students do not meet face to face with their peers and teachers, the support for collaboration becomes extremely important [Constantino-González and Suthers, 2002]. Effective collaborative learning includes both learning to effectively collaborate, and collaborating effectively to learn, and therefore a collaborative system must be able to address collaboration issues as well as task-oriented issues [Soller, et al. 2005].

There have been several definitions for collaborative learning. The broadest (but unsatisfactory) definition is that it is a *situation* in which *two* or *more* people *learn* or attempt to learn something *together.* A more comprehensive definition given by [Roschelle and Teasley, 1995] states as follows: "… a coordinated, synchronous activity that is the result of a continued attempt to construct and maintain a shared conception of a problem". Collaborative learning provides an environment to enliven and enrich the learning process [Kumar, 1996]. Introducing interactive partners into an

educational system creates more realistic social contexts, thereby increasing the effectiveness of the system. Such an environment eliminates the isolation feeling, and would help sustain the student's interests and motivations.

Collaboration issues include the distribution of roles among students (e.g. critic, mediator, idea-generator) [Burton, 1998], equality of participation, and reaching a common understanding [Teasley and Roschelle, 1993], while task-oriented issues involve the understanding and application of key domain concepts. The educational systems are responsible for regulating the interaction, and guiding the students towards effective collaboration and learning.

In the last decade, many researchers have contributed to the development of Computer-Supported Collaborative Learning (CSCL) and advantages of collaborative learning over individualised learning have been identified [Inaba and Mizoguchi, 2004]. Some particular benefits of collaborative problem-solving include: encouraging students to verbalise 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 articulate their reasoning, and elaborate and reflect upon their knowledge [Soller, 2001; Webb, Troper and Fall, 1995]. These benefits, however, are only achieved by active and well-functioning learning teams [Jarboe, 1996]. Numerous systems for collaborative learning have been developed; however, the concept of supporting peer-to-peer interaction in CSCL systems is still in its infancy. A variety of strategies for computationally supporting online collaborative learning have been proposed and used, while more studies are needed that test the utility of these techniques [Soller, et al. 2005]. Full-scale evaluations of these systems are yet to be seen.

This thesis describes an Intelligent Tutoring System (ITS) that uses Constraint-Based Modelling (CBM) approach to support both problem-solving and collaborative learning. CBM has been used successfully in several tutors supporting individual learning, but has not been used in the past to analyse and model group interactions.

This chapter forms a high-level overview of the thesis. Intelligent Tutoring Systems are introduced in Section 1.1. Section 1.2 presents CBM followed by a description of

collaborative learning in Section 1.3. Section 1.4 outlines the thesis contribution and a guide to the rest of the thesis is provided in Section 1.5.

## 1.1 Intelligent Tutoring Systems

Computers have been used in education for more than three decades. Computer-based training (CBT) and computer-aided instruction (CAI) were the first such systems deployed as an attempt to teach using computers. In these types of systems, the instruction was not individualised to the learner's needs. Instead, the decisions about how to move a student through the material were script-like, not taking the learner's abilities into account. While both CBT and CAI may be somewhat effective in helping students, they do not provide the same kind of individualised attention that a student would receive from a human tutor [Bloom, 1984], as they do not reason about the domain and the learner. This has prompted research in the field of intelligent tutoring systems (ITSs).

ITSs achieve their intelligence by representing pedagogical decisions about how to teach as well as information about the learner. They have been shown to be highly effective at increasing students' performance and motivation [Beck, Stern, and Haugsjaa, 1996]. The goals of learning no longer were solely based on acquiring skills and facts but started to include the strategies and process used by the student to reach mastery of a knowledge domain [Garito, 1991]. Learning should be influenced by social interaction, activity and construction of understanding [McTaggart, 2001].

ITSs have been successfully developed for a variety of domains including mathematics, physics, programming, databases, design tasks and learning new languages. Examples are Andes Physics Tutor (problem solving in introductory college physics ) [VanLehn, et al. 2005], Algebra Cognitive Tutor (problem solving in a high-school algebra course) [Anderson, et al. 1995], AutoTutor (problem solving in college physics and other domains) [Graesser, et al. 2003; Graesser, et al. 1999], Sherlock (troubleshooting a large piece of simulated electrical equipment) [Katz, et al. 1998], Steve (teaching hierarchical, multi-step procedures, such as how to start a large air compressor) [Johnson, et al. 1998], SQL-Tutor [Mitrovic, 1998], COLLECT-$\mathcal{UML}$ (Object-Oriented software design) [Baghaei, Mitrovic and Irwin, 2005; Baghaei,

Mitrovic and Irwin, 2006**;** Baghaei and Mitrovic, 2006] and KERMIT (database design) [Suraweera and Mitrovic, 2001; 2004].

## 1.2 Constraint-Based Modelling

Student modelling can be defined as the process of gathering relevant information in order to identify and represent the knowledge state of a student. In an ideal case, the model of a learner should illustrate their knowledge, preferred learning strategies, areas of interest besides that of instruction, preferred presentation style, etc. Several techniques for student modelling have been developed for particular domains, the generality of which is yet to be determined by applying them elsewhere.

The task of building a student model is extremely difficult and laborious, due to huge search spaces involved. One of the student modelling approaches that focuses on reducing the complexity of the task is Constraint-Based Modelling [Ohlsson, 1994]. CBM is based on Ohlsson's theory of learning from performance errors [Ohlsson, 1996], stating that we learn when we catch ourselves or are caught by someone else making mistakes. CBM focuses on correct knowledge rather than describing the student's problem solving procedure as in model tracing [Mitrovic, et al. 2003]. The basic assumption is that diagnostic information is not hidden in the sequence of student's actions, but in the situation (or the problem state) that the student arrived at. CBM does not represent student's actions, but the effects of their actions instead [Mitrovic, et al. 2001].

A number of constraint-based tutoring systems have been successfully developed for supporting individual learning, covering a wide range of domains. Examples are SQL (the database query language) [Mitrovic, 1998], database modelling [Suraweera and Mitrovic, 2001; 2004], data normalization [Mitrovic, 2002; 2005], punctuation tutor (CAPIT) [Mayo, et al. 2000] and English language tutor (LBITS) [Martin, 2002].

## 1.3 Collaborative Learning

In one-to-one computer-based tutoring, the system interacts with one student and attempts to personalise the tutoring to the needs of the student. On the other hand, in an

one-to-many collaborative learning environment, the system interacts with a group of students, imparting the subject knowledge using a collaborative learning strategy.

Performing team tasks well means not only having the skills to carry out the task, but also collaborating well with team-mates, which includes asking questions to gain a better understanding of the main concepts, elaborating and justifying opinions and sharing and explaining ideas. When group members' combined skills are sufficient to complete the learning task, effective group work can result in greater overall achievement than individual learning [Heller, Keith and Anderson, 1992; Joiner, 1995; Soller and Lesgold 2000]. Students learning in effective teams benefit through both enhanced learning of the task, and improvement in the social interaction skills they need throughout their lives.

The promise of collaborative learning is to allow students to learn in relatively realistic, cognitively motivating and socially enriched learning contexts, compared to other tutoring paradigms. With CSCL, students can discuss the strategies with a group of fellow students who can motivate, criticise, compete and direct towards better understanding of the subject matter.

There are a number of research studies and implemented systems available in the literature to emphasize the effectiveness of collaboration and the compelling need for the infusion of collaborative techniques in learning environments, some of which are described in the next chapter. For further information on other studies, refer to [Miyake, 1986; Blaye, et al. 1990; Chan, 1991; Dimitracopoulou and Petrou, 2004].

## 1.4 Thesis Contributions

Automatic analysis of interaction and group learning through a distance collaborative learning system is at the forefront of educational technology research [Jermann, Soller and Lesgold, 2004]. This is the guiding role of the computer, and probably the most challenging function to program. It demands some ability to understand and assess the interaction, as well as a set of diagnostic rules that recommend remedial action.

Understanding student interaction has been a major challenge in CSCL research, not unlike the challenge that understanding natural language poses to the field of artificial intelligence. Furthermore, correctly diagnosing the state of interaction does

not guarantee that students will pay attention to the feedback proposed by the system, and that this feedback will have the intended effect [Crook, 1994]. For these reasons, CSCL systems tend to provide minimal advice based on the analysis of student actions taken on workspaces and statistics of messages that student send to each other (e.g. frequencies of different types of messages, such as Requests or Acknowledgements).

Previous research shows that collaborative learning provides an environment to enrich the learning process by introducing interactive partners into an educational system and creating more realistic social contexts. However, there are no recipes that guarantee successful use of technology to support collaborative learning due to the complexity of analysing collaborative interaction. There have been a large number of studies proposing and developing collaborative learning tools and different approaches to analysing the collaborative interaction, as discussed in Chapter 2 (Sections 2.3 and 2.4); yet the concept of supporting peer-to-peer interaction in computer-supported collaborative learning systems is still in its infancy.

Constraint-Based Modeling has previously been used to effectively represent domain knowledge in several ITSs supporting individual learning (See Section 2.2). The main contribution of this research is the use of CBM to model collaborative skills, not only domain knowledge. CBM technique is used in this project to model student/group knowledge and represent the ideal model of interaction as a set of meta-constraints. COLLECT-UML (the system implemented during the course of this research) 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. UML is selected as an appropriate task for this research due mainly to its collaborative nature, its appropriateness for discussion and its complexity for novice software engineering designers.

The work evaluates the possibility of giving advice when comparing student work with an expert solution *as well as* group solution, in contrast to the approach usually taken by other previous studies that have either supported tutoring (teaching the domain concepts) or coaching the social interaction (encouraging the students to discuss and

participate). Therefore, the proposed system can be considered as a learning environment that supports both individual and collaborative learning.

The evaluations are an essential part of research, which include testing the correctness of the system, the pedagogical agent, the usability of the interface and feedback generation. The studies have been carried out at the University of Canterbury, in a second-year Software Engineering course.

## 1.5 A Guide to the Thesis

Chapter 2 reviews Intelligent Tutoring Systems, their main components and includes a description of constraint-based modelling technique. It then identifies some of the collaborative learning systems, examines their advantages and disadvantages and describes the approaches used in current literature to analyse the collaborative learning interaction. Chapter 3 presents the single-user version of COLLECT-UML and the evaluation study conducted in May 2005. Chapter 4 describes the extension made to the single-user version to make it support collaboration. The enhancement process included implementation of the shared workspace, modification of the pedagogical module to support groups of users, designing and implementing a group-modelling component, and developing a set of meta-constraints which are used to represent an ideal model of collaboration. Chapter 5 presents the evaluation studies performed using the multi-user version and discusses the results. Finally, the conclusions and future work are given in the last chapter.

In the course of this research, we have prepared and presented nine publications, which are listed in Appendix C.

# Chapter 2

# Background

Human tutoring is widely believed to be the most effective form of instruction, and experimental work confirms that expert human tutors can produce extremely large learning gains [Bloom, 1984]. Ever since computers were invented, they seemed to be capable of becoming untiring and economical alternatives to expert human tutors. This dream has proved difficult to achieve, however significant progress has been made [VanLehn, 2006]. The late 1980s and early 1990s generated a lot of research into Intelligent Tutoring Systems, a particularly effective educational technology, and their application to individualised instruction.

In this research we are primarily interested in tutoring systems that emphasise *learning by doing*, where students are given a problem that they attempt to solve (i.e. collaborative problem solving). The tutor provides rich context-sensitive pedagogical assistance during the problem solving process. It responds in a similar way to a human tutor by transferring the knowledge that the student lacks.

This chapter is organized as follows. Section 2.1 describes an ITS' main components. Section 2.2 presents a description of the constraint-based modelling technique and gives examples of some of CBM tutors developed so far. Section 2.3 identifies some of the collaborative learning systems and examines their advantages

and disadvantages. Finally, Section 2.4 describes the approaches used in current literature to analyse the collaborative learning interaction.

## 2.1 ITS Structure

The various components of an ITS work together to produce an instructional system that can recognise patterns of learner's behaviour and respond with instruction suitable to those patterns. An ITS typically consists of four major components shown in Figure 2.1.

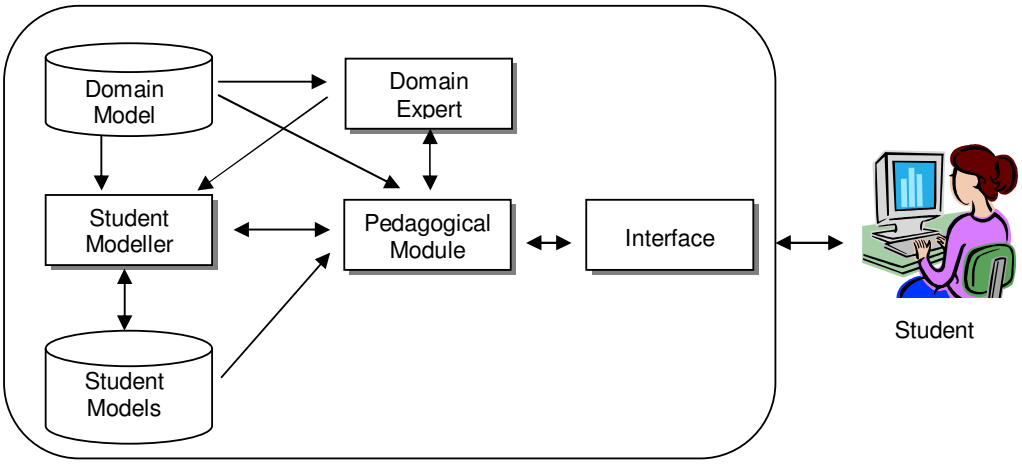



*Figure 2.1: ITS Architecture*

### 2.1.1 Student Modeller

The student model records information about the student. This information reflects the system's belief of the learner's current knowledge state, and helps to lead the student through the domain. The diagnosis done by the student modeller is used by the pedagogical module (described in Section 2.1.2) to recognise errors, generate feedback messages, generate problems, and control progress through the curriculum.

The ability of an ITS to deliver appropriate individualised instruction to a student depends on the type of the information held about the student in the student model. This in turn depends on the type and level of sophistication of the knowledge representation used in the system, and on the effectiveness of the methods used to

extract new information about the student and incorporate the new information into the student model.

An effective intelligent tutor has a good sense of what the student understands, knows and can do. If this information is used to sequence the learning materials, a better student model will result [McTaggart, 2001]. Building a more effective student model will also have an impact on the instructional model, hence making it the most critical component of the ITS.

Researchers use various student-modelling techniques. Model tracing [Anderson, et al. 1996], constraint-based modelling (CBM) [Ohlsson, 1994], stereotypes [Winter and McCalla, 1999] and overlays [Carr and Goldstein, 1977] are some of the popular ones. Model tracing and CBM focus on modelling the student's short-term knowledge, but can be extended to model long-term knowledge. The main difference between the two approaches is that model tracing represents procedural knowledge whereas CBM represents only declarative knowledge. Stereotypes and overlays are used to model long-term student knowledge and differ in the amount of detail offered by each representation. Stereotypes are abstract classifications of the students into groups and overlays are representations of the student's knowledge as a subset of the domain knowledge.

### 2.1.2 Pedagogical Model

The pedagogical or instructional model contains knowledge for making decisions about tutoring tactics. The overlapping of ITS components makes the instructional model highly dependent on the diagnostic processes of the student model for making decisions about what information to present to the student, and when and how to present that information [Buiu, 1999].

Compared with a human tutor who can adopt different methods and strategies, most instructional models rely on a set of tutoring strategies. The pedagogical strategies used in existing research are *enforce correctness* (where the system is in full control), *computer coach* (the student is in control), *Socratic teaching* (the system leads the student to form general principles by posing questions and counter-examples) and *collaborative learning* (where more than one student is involved).

### 2.1.3 Domain Model

The domain model is an organised database of the declarative and procedural knowledge in a specific domain. Developing a domain model that provides comprehensive coverage of the subject material can be a difficult and expensive task. Model-tracing tutors represent knowledge in the form of low-level production rules that completely describe the expected student behaviour [Anderson, et al. 1996]. Constraint-based tutors [Mitrovic, et al. 2001] use a set of constraints to describe the underlying concepts of the domain and can be used to identify whether or not an answer contains any errors (See Section 2.2).

The goal of the ITS is to replicate these knowledge structures in the mind of the learner. The domain model is tied closely with the student model; the system has to search the domain knowledge as it compares the model of a student's learning with that of the domain knowledge.

The way in which the domain model works is not necessarily the way humans solve problems. Humans will not use exhaustive search, but apply techniques appropriate to the problem-solving domain. Newer models for the domain model have been proposed, which realistically simulate human problem solving. These models incorporate knowledge reflective of the facts, procedures, and qualities that humans use in structuring their own representation of knowledge [Orey and Nelson, 1993].

### 2.1.4 Interface

The human-computer interface continues to be an important area of research in Computer Science. A good interface will anticipate the user's actions, be consistent, provide a high level of interaction, structure students' thinking and make use of metaphor [Orey and Nelson, 1993]. The user is learning the interface along with the content, so any additional cognitive load should be minimal.

There are many types of interfaces. A particular style may depend on the learner's ability and the knowledge to be learned. How well an artificial dialog models the teacher-student relationship is the topic of continuing research. The interface is important as a communication medium, as a problem-solving environment that supports the student in the task at hand, and as an external representation of the system's domain and instructional models.

The bandwidth problem refers to the correspondence between the learner's mental states and the observable actions captured by the interface model. As computer systems become more powerful and complex, it will be possible to provide interface models that increase the bandwidth. The result will be better diagnosis of the learner's level and the subsequent actions by the pedagogical models.

## 2.2 Constraint-Based Modelling (CBM)

The task of building a student model is extremely difficult and laborious, due to huge search spaces involved [Mitrovic, et al. 2001]. Various approaches for dealing with the intractability of student modelling have been introduced. Self [1990; 1994] recommends design of the interactions in a way that information necessary for building a student model is provided by the student, and not inferred by the system. Also, it is not useful to be able to identify misconceptions in the student knowledge that cannot be dealt with by the tutor. An ITS should model only what it is capable of using in order to generate remedial or other pedagogical actions.

Constraint-based modelling (CBM) [Ohlsson, 1994] is based on a new theory for domain and student modelling termed learning from performance errors [Ohlsson, 1996]. The theory asserts that learning occurs when we catch ourselves (or are caught by some other party) making mistakes. Ohlsson argues that even though we know what we should do, we are often unable to make the correct decision because there are too many possibilities to consider. Thus merely learning the declarative knowledge is insufficient; we must also learn how to apply the knowledge in order to achieve mastery of the domain.

CBM focuses on correct knowledge rather than describing the student's knowledge (including misconceptions) as with model tracing. The key assumption in CBM is that diagnostic information is in the problem state rather than the path taken to arrive at the problem state. This assumption is supported by the fact that no correct solution can be arrived at by traversing a problem state that violates some domain concepts. Since the space of erroneous knowledge is far greater than the correct knowledge, CBM models a domain by using a set of constraints to represent the correct solutions. They are used to identify correct solutions from the space of all possible solutions. The constraints,

unlike in model tracing, are used to represent declarative knowledge only, such as theorems in mathematics [Suraweera, 2003].

Each constraint consists of two parts: a relevance condition, *Cr*, and a satisfaction condition, *Cs*. The relevance condition identifies the pedagogically relevant states and the satisfaction condition identifies states where this piece of knowledge has been correctly applied. A generic constraint states that *if Cr* is true, *then Cs* has to be also true. Otherwise, something has gone wrong and the feedback message attached to the violated constraint is shown to the student.

Some advantages of CBM are as follows:

1. CBM does not require a runnable expert module, as many other SM approaches do. This is an important advantage of CBM, because in many domains it can be very difficult to build the problem solver. CBM-based computerised tutors are able to generate instructional actions even without being able to solve problems on their own, by focusing on violated constraints [Mitrovic, et al. 2001].

2. Another advantage of CBM is its computational simplicity. Instead of using complex reasoning as required by other diagnostic approaches, CBM reduces student modelling to pattern matching. Conditions are combinations of patterns, and can therefore be represented in compiled forms, such as RETE networks [Forgy, 1982], which are very fast. In the first step, all relevance patterns are matched against the problem state. In the second step, the satisfaction components of constraints that matched the problem state in the first step (i.e., relevant constraints) are matched. If a satisfaction pattern matches the state, then the constraint is satisfied, and the ITS is not to take any action. In the opposite case, the constraint is violated. The student model consists of all satisfied and violated constraints. This technique can be used for both on- and off-line student modelling.

3. Building the knowledge base for CBM is much simpler than for model tracing. The difference in effort is evident in the domain of database modelling. In order

to assess a simple database modelling problem, KERMIT [Suraweera and Mitrovic, 2002; 2004], the constraint-based tutor for database modelling, requires 23 constraints (modelling only a part of the domain). In contrast, a procedural model requires 25 production rules, 10 general chunks and 30 problem-specific chunks, or a total of 65 elements [Mitrovic, et al. 2003]. Moreover, the 30 problem-specific chunks are specific to the problem and cannot be used outside the relevant problem. On the other hand, constraints are not problem-specific and they cover a wider area of a domain. In addition, CBM also requires less time to acquire knowledge. Mitrovic [Mitrovic and Ohlsson, 1999] has reported that an average of 1.1 hours of work was required to identify a constraint. This is a significant saving in effort compared to the ten or more hours required to identify a single production rule [Anderson, et al. 1996].

4. It is crucial when using model tracing that the procedural model be comprehensive or the tutor will not be able to trace the actions of the student. With CBM, the effect of a missing constraint is highly restricted. A missing constraint may result in failing to identify a particular error. As the constraints are modular in nature, the solution can be analysed with the remaining constraints. This reduces the need for large-scale evaluations about the correctness of the domain model and allows the domain to be incrementally developed by incorporating it in a tutoring system.

5. CBM does not require extensive studies of student bugs as in enumerative bug modelling. Buggy productions are used in model-tracing tutors to discover students traversing erroneous paths and provide feedback to describe why the step is incorrect. Building a bug library is an intractable task, typically requiring empirical evaluations of student behaviours. In CBM correct knowledge is encapsulated in the constraints. An incorrect answer results in an error, which would be caught by a constraint. Whilst it may be desirable to analyse student solutions to determine problematic areas so that these are modelled, the analysis

and work effort required is less since the errors do not have to be linked to procedural steps.

6. A limitation of model-tracing tutors is the restrictiveness inherent in their implementation. Although it is possible to allow students to stray from a correct solution path, the extent to which this is practical is limited by the need to determine when the student has gone off track completely. The further away students deviate from the solution path, the harder it is for the system to understand why this occurred and to understand whether they are completely lost. CBM does not have this weakness, since the path taken by students in arriving at a particular state is irrelevant. This transfers control to students and enables them to select their own strategy as well as change strategies during problem solving.

7. Model-tracing tutors usually provide immediate feedback to students when they stray from the solution path. Even though it is possible to delay the feedback to a later stage, there are limitations due to reasons of computational complexity. Thus students cannot be allowed to deviate too much from the solution path since determining the path back to the correct track becomes increasingly difficult. CBM, on the other hand, is not affected by the pedagogical method adopted since it is concerned only with the current problem state and not the traversed path. Although most CBM tutors provide feedback after a request by the student, they are also capable of providing feedback after each step. The only requirement for providing immediate feedback is for the system to identify that the current solution is incomplete so that it knows to evaluate the solution on correctness only and not on completeness. The system can check for completeness when the student declares that the solution has been accomplished [Suraweera, 2003].

CBM has been successfully applied in a number of domains. These tutors, called constraint-based tutors, [Mitrovic, et al. 2001] have been developed in domains such as

SQL (the database query language) [Mitrovic, 1998], database modelling [Suraweera and Mitrovic, 2001; 2004], data normalization [Mitrovic, 2002; 2005], logical database design (mapping conceptual database schemas into relational schemas) [Milik, Marshall and Mitrovic, 2006], logic programming [Le and Menzel, 2005], punctuation [Mayo, et al. 2000] and German adjective endings [Nicholas, 2006]. All four tutors in the database domain, SQL (SQL-Tutor), database design (KERMIT), logical database design (ERM-Tutor) and normalization (NORMIT), were developed for tertiary students as problem-solving environments. Students solve problems presented to them with the assistance of feedback from the system. The punctuation tutor (CAPIT) was developed with the goal of improving the capitalisation and punctuation skills of 10-11 year old school children. LBITS [Martin, 2002] is another constraint-based ITS developed to teach basic English language skills to elementary and secondary school students.

## 2.3 Intelligent CSCL Environments

Many studies indicated that collaborative learning is effective in generating positive outcomes not only in academic performance, but also in supporting the affective and social aspects of learning [Resta, 2004]. Johnson et al. [1998b] said: "College life can be lonely ... College students can attend class without talking to other students ... anyone, no matter how intelligent or creative, can have such feelings ... Contributing to other's well-being increases one's own well-being. Without the sense of belonging, acceptance, and caring that result from cooperative efforts with others, students can remain isolated and vulnerable" (p. 4:11).

In an online collaborative learning environment, learners do not need to be isolated from the other learners. They can make their knowledge public, ask questions, justify their answers, express their ideas, experience multiple perspectives and move to deeper levels of understanding. Panitz [1999] describes various benefits of collaborative learning; some of them are summarized in Table 2.1.

*Table 2.1: Benefits of Collaborative Learning [Panitz, 1999]*

| | |
|---|---|
| **Academic Benefits** | - Promotes critical thinking skills<br>- Involves students actively in the learning process<br>- Models appropriate problem solving techniques<br>- Personalizes large lectures<br>- Motivates students in specific curriculum |
| **Social Benefits** | - Develops a social support system for students<br>- Builds diversity understanding among students and teachers<br>- Establishes a positive atmosphere for modelling and practicing cooperation<br>- Develops learning communities |
| **Psychological benefits** | - Increases students' self esteem through student centred instruction<br>- Reduces anxiety through cooperation<br>- Develops positive attitudes towards teachers |

This section provides examples of three main types of supportive collaborative learning systems, in the context of the collaboration management model [Reimann, 2003; Soller, et al. 2005]. Systems that reflect actions (mirroring systems) collect raw data in log files and display it to the collaborators. Systems that monitor the state of interaction (meta-cognitive tools), model the state of interaction and provide collaborators with visualizations that can be used to self-diagnose the interaction. Finally, coaching or advising systems guide the collaborators by recommending actions students might take to improve their interaction. The next three subsections describe systems that fall into these categories. There are other types of CSCL systems (e.g. peer-to-peer learning and learning with artificial peers) in the literature that will not be mentioned in this thesis, as the focus of this thesis is on systems that supports groups of students learning collaboratively.

### 2.3.1 Reflecting Actions

The most basic level of support a system may offer involves making the students aware of the participants' actions. Actions taken on shared resources, or those that take place in private areas of a workspace may not be directly visible to the collaborators, yet they may significantly influence the collaboration. Raising awareness about such actions could help students maintain a representation of their team members' activities.

Some systems in this category represent actions along a timeline. For example, Plaisant, Rose, Rubloff, Salter and Shneiderman [1999] describe a system in which students learn the basics of vacuum pump technology through a simulation. As the learner manipulates the controls of the simulation, a history of actions is displayed graphically beneath the target variable (e.g. pressure). The data displayed to the student does not undergo any processing or summarising, but directly reflects the actions taken on the interface. These graphical records of actions can then be sent to a tutor or a peer, or replayed by the learner to examine his/her own performance.

PENCACOLAS (PEN Computer Aided COLlAborative System) [Blasco, et al. 1999], a system designed to teach collaborative writing, is another example of an environment that reflects users' actions. It enables groups of students, and an instructor, to generate text synchronously. PENCACOLAS models compositions as problem-solving situations that follow a recursive process including a series of phases (e.g. brainstorming, planning, writing and revision). Students can also interact asynchronously, by revising their peers' compositions or exchanging short messages with each other. The system records all the students' writing events, which are used to analyze the student activity, and to enable the review and evaluation of previous composition phases. Reviewing students' intermediate writing steps may provide important insights into the evolution of their cognitive development. In order to assist with formative evaluation, PENCACOLAS automatically generates file names that identify users, sessions and phases, therefore allowing evaluation of both collaborative and individual work. This will also allow the instructor to perform a self evaluation in which they review their pedagogical interventions [Soller, et al. 2005].

### 2.3.2 Monitoring the State of Interactions

Systems that monitor the state of interaction fall into two categories: those that aggregate the interaction data into a set of high-level indicators, and display them to the participants, and those that internally compare the current state of interaction to a model of ideal interaction, but do not reveal this information to the users. "In the former case, the learners are expected to manage the interaction themselves, having been given the appropriate information to do so. In the latter case, this information is either intended to be used later by a coaching agent, or analysed by researchers in an effort to understand and explain the interaction" [Soller, et al. 2005; Reimann, 2003].

Some of monitoring systems interpret the content of the interaction, instead of focusing on quantitative aspects of the interaction. Analyzing participation rates involves counting words or messages, whereas indicators such as acknowledgement rate and delay (how often users respond to incoming messages, and how long this takes) or role distribution need more sophisticated computation. Studying more complex variables often involves analyzing the semantic aspects of interaction and the patterns of student actions. A structured interface may facilitate the interpretation of actions by the system. For example, users may be required to select a communication category (e.g. propose, encourage, question) when they send messages to each other. MArCo [Tedesco and Self, 2000] is a dialog-oriented system for the detection of meta-cognitive conflicts. The system adopts a dialog game approach with a limited set of possible dialog moves. User utterances must be formulated in a formal language that enables the conversation to be mapped onto a belief-based model (BDI). The analysis mechanism detects disagreements and conflicts between users' beliefs and intentions.

Muehlenbrock and Hoppe [1999] were one of the first to propose actions in shared workspaces as a basis for a qualitative analysis. In contrast to dialog tags, actions on external representations are not only interrelated on a temporal dimension, but also on a structural dimension, i.e. concerning their context of application. This approach has been named *action-based collaboration analysis* [Muehlenbrock, 2000] and is implemented as a plug-in component in the generic framework system CARDBOARD, which includes intelligent support components (CARDDALIS). The interface contains a shared workspace for co-constructive activity and a chat tool with four sentence

openers. Action-based collaboration analysis derives higher-level descriptions of group activities, including conflicts and coordination, based on a plan recognition approach.

One reason for not presenting a visualization of the model of interaction to the students or the teacher is that the evaluation of complex variables includes a margin of error; therefore, it might be more appropriate to abstract the relevant aspects of the model and present them to the users. 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 which 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 determines *when* and *why* the students are having trouble learning the new concepts they share with each other. An instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities.

While this approach has its merits, it has some limitations as well. As with all inductive techniques, the models identified are dependent on sampling and size of the learning set; there is always the possibility that groups not observed yet would lead to significant changes in the model [Reimann, 2003]. Besides this principle problem, there is the problem of what counts as success (i.e. the criterion) and what aspects of the group behaviour are recorded and fed into the induction algorithm. Both types of decisions will largely affect the outcome of the machine learning exercise. In addition, the system does not provide an ideal solution to the problem (assuming that the students reach a correct answer by collaboration) and has no support for individual work (private workspace).

### 2.3.3 Offering Advice

This section describes systems that analyse the state of collaboration using a model of interaction, and offer advice intended to increase the effectiveness of the learning process. The coach in an advising system plays a role similar to that of a teacher in a collaborative learning classroom. The systems described are distinguished by the nature of the information in their models, and whether they provide advice on strictly collaboration issues or both social and task-oriented issues.

A classroom teacher might mediate social interaction by observing and analyzing the group's conversation, and noting, for instance, the levels of participation among group members, or the quality of the conversation [Soller, et al. 2005]. A CSCL system that can advise the social aspects of interaction, therefore, requires some ability to understand the dialog between group members. DEGREE [Barros and Verdejo, 2000], an asynchronous newsgroup-style system, accomplishes this by requiring users to select the type of contribution (e.g. proposal, question, or comment) from a list each time they add to the discussion. The system's model of interaction is constructed using high-level attributes such as *cooperation and creativity* (derived from the contribution types mentioned above), as well as low-level attributes such as the *mean number of contributions.* The system rates the collaboration between pairs of students along four dimensions: *initiative, creativity, elaboration and conformity*. These attributes, alongside others such as the length of contributions, factor into a *fuzzy inference procedure* that rates students' collaboration on a scale from *awful* to *very good*. The system is dependent on users' ability to choose the correct contribution type (proposal, comment, etc.). DEGREE only focuses on the social aspects of collaborative learning and has no support for shared and private actions (it only provides tagged dialog).

Group Leader Tutor [McManus and Aiken, 1995] is intended to promote the collaboration skills identified in Johnson and Johnson [1991] during the course of problem solving discussions between two students. The students send messages to each other by selecting a sentence opener from a menu and then elaborating on this opener with additional text. There was a one to one correspondence between the sentence openers implemented by McManus and Aiken and the skills identified by Johnson and Johnson. An ITS offering advice and feedback on the student's skill is used during the

course of the discussion and generates feedback at the end of the discussion. The tutoring system's suggestions are based on the concept that a conversation can be understood as a series of conversational acts (e.g. Request, Mediate, denoted by the use of the sentence openers) that correspond to users' intentions. Unlike other systems, users are not limited to certain acts based on the system's beliefs. Group Leader monitors contributions from the students and compares them to an ideal model of interaction.

Group Leader uses *state transition matrices* to define what conversation acts should properly follow other acts. It then compares sequences of students' conversation acts to those recommended in four finite state machines developed specifically to monitor discussions about comments, requests, promises and debates. The system analyses the conversation act sequences, and provides feedback on the *students' trust, leadership, creative controversy and communication skills*. It only addresses the social aspect of group learning and there is no support for shared and private actions.

GRACILE [Ayala and Yano, 1998] is an agent-based system designed to help students learn Japanese, addressing both social and task-oriented aspects of group learning. The system maintains user models for each of the students, and forms beliefs about potential group learning opportunities. Group learning opportunities are defined as those that promote the creation of *zones of proximal development* [Vygotsky, 1978], enabling a student to extend her/his potential development level. GRACILE's agents examine the progress of individual learners, propose new learning tasks based on the learning needs of the group, and help to maximize the number of situations in which students may successfully learn from one another. The following systems are described in more details.

#### 2.3.3.1 Belvedere

Belvedere [Suthers and Jones, 1997] is a web-based collaborative environment for teaching scientific enquiry. The core functionality is a shared workspace for constructing inquiry diagrams which relate data and hypotheses by evidential relations (using a graphical structure), and a collaborative enquiry database to record the progress of the enquiry. Provision is also made for tutor input to stimulate lines of enquiry, etc.

The interface is shown in Figure 2.2. The tool also provides a chat facility for unstructured discussions, facilities for integrated use with Web browsers, and two artificial intelligence coaches.

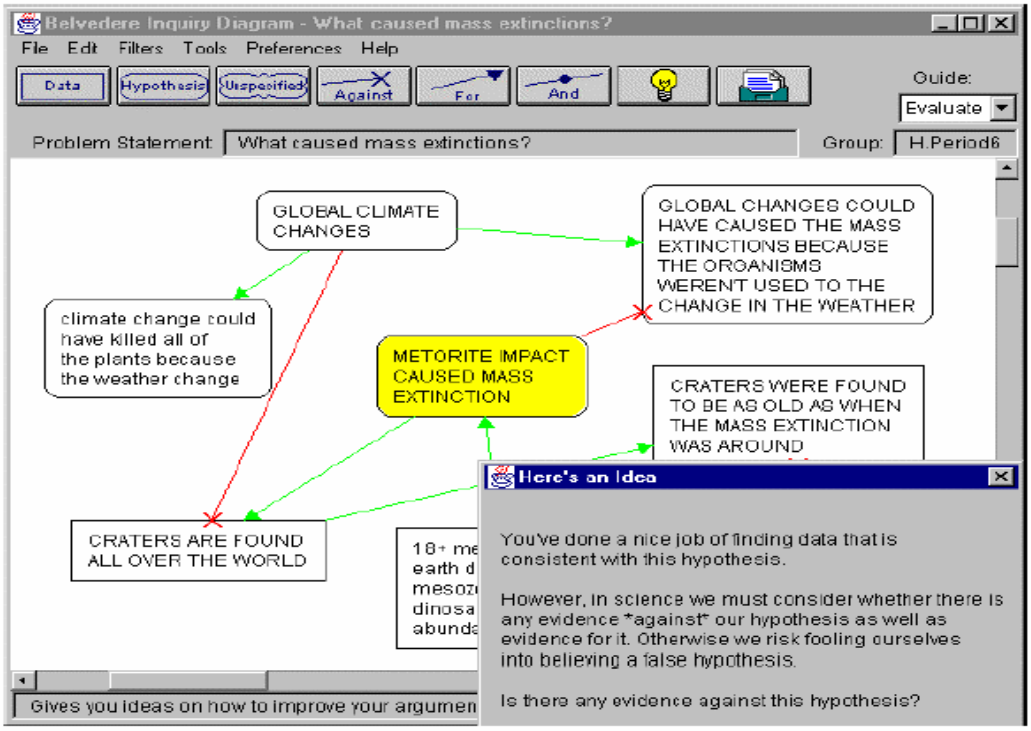



*Figure 2.2: Belvedere Interface [Suthers and Jones, 1997]*

The first coach provides general advice on the structure of the inquiry diagrams from scientific argumentation point of view, helping the students understand the principles of inquiry. The other coach performs various comparisons between the students' diagrams and an inquiry diagram provided by an expert. This coach can provide students with feedback regarding correctness, or confront students with new information (found in the expert's diagram) that challenges students in some way. To avoid interrupting students' thought processes, the coach is minimally intrusive, usually remaining quiet unless students ask for feedback, and flashing its light bulb only when it has critical advice to offer. It coaches critical inquiry by asking questions students may not have thought of, based on criteria of inquiry and argumentation in science.

Belvedere has a distributed architecture and tutoring functionality is distributed between the client and the server. Java is used to deliver the user interface, while the application server is written in a variety of tools.

#### 2.3.3.2 LeCS

LeCS [Rosatelli, Self, and Thirty, 2000] 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. It is similar to GRACILE in that a set of computer agents guide students through the analysis of case studies and address both social and task-oriented aspects of group learning. The agents monitor students' levels of participation, and track students' progression through the task procedure, while addressing students' misunderstandings and ensuring group coordination. The system provides a solution tree, so that the learners can map or visualise the building up of their solution.

LeCS was implemented in the Delphi language and has client-server architecture. The server hosts sessions and the client interacts with sessions. A session is associated with a group of students working collaboratively on the solution of a case study. The clients run on the students' machines. The server can run on one of the student's machine or alternatively on a different machine. The LeCS architecture includes three classes of agents: interface agent, information agent and advising agent. There is one information agent and one advising agent running during a session, but as many interface agents as there are participants logged on.

Figure 2.3 illustrates a screenshot of LeCS interface. The text editor is an individual space where the learners can edit their individual answers. The text editor is used during the part of the solution process when individual learning takes place. The individual answers edited with this tool are used when the learners participate in the case discussion (collaborative learning). The chat tool is fairly similar to the traditional programs of this kind and is where the case study discussion takes place.

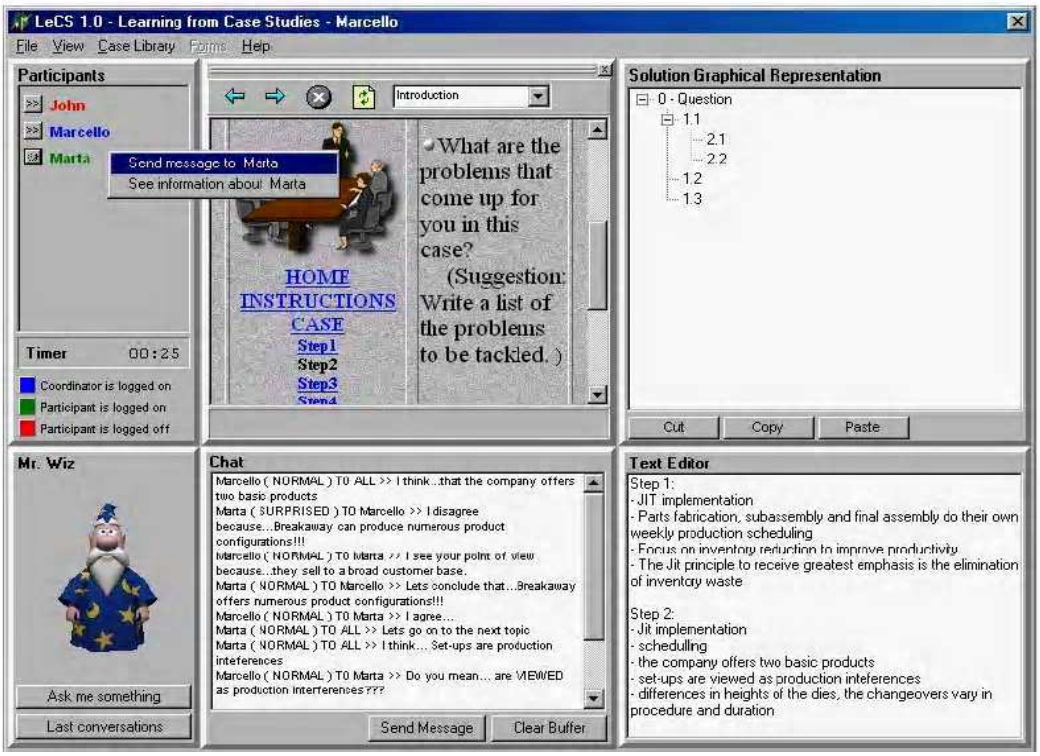



*Figure 2.3: The LeCS Interface [Rosatelli, Self, and Thirty, 2000]*

Some of the limitations of LeCS are as follows:

- The sentence openers in LeCS are only intended to facilitate the discussion
- 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 constraints used in the domain knowledge concerning the case study are very simple
- The information obtained from the chat and text area are not examined
- There is no support for individual modelling
- The dialogue contributions and the edited texts are only used in implementing the solution tree

### 2.3.3.3 COLER

COLER [Constantino-Gonzales and Suthers, 2000] is a web-based learning environment that uses decision trees to coach students collaboratively learning Entity-Relationship modelling, a formalism for conceptual database design. The system identifies learning opportunities based on differences between students' individual and group work and tracking levels of participation. In order to leverage the learning opportunities presented by conflicts, the coach encourages students to negotiate when differences are detected between solutions, and encourages participation in other ways. The work is based on the Socio-Cognitive Conflict Theory [Doise and Mugny, 1984], stating that the students learn from disagreements when they identify and resolve conflicts in their viewpoints, present alternatives, and request and give explanations.

Figure 2.4 illustrates the COLER interface. Students construct their individual solutions in the private workspace (upper right). They use the shared workspace (lower centre) to collaboratively construct ER diagrams while communicating via the chat window (lower right).

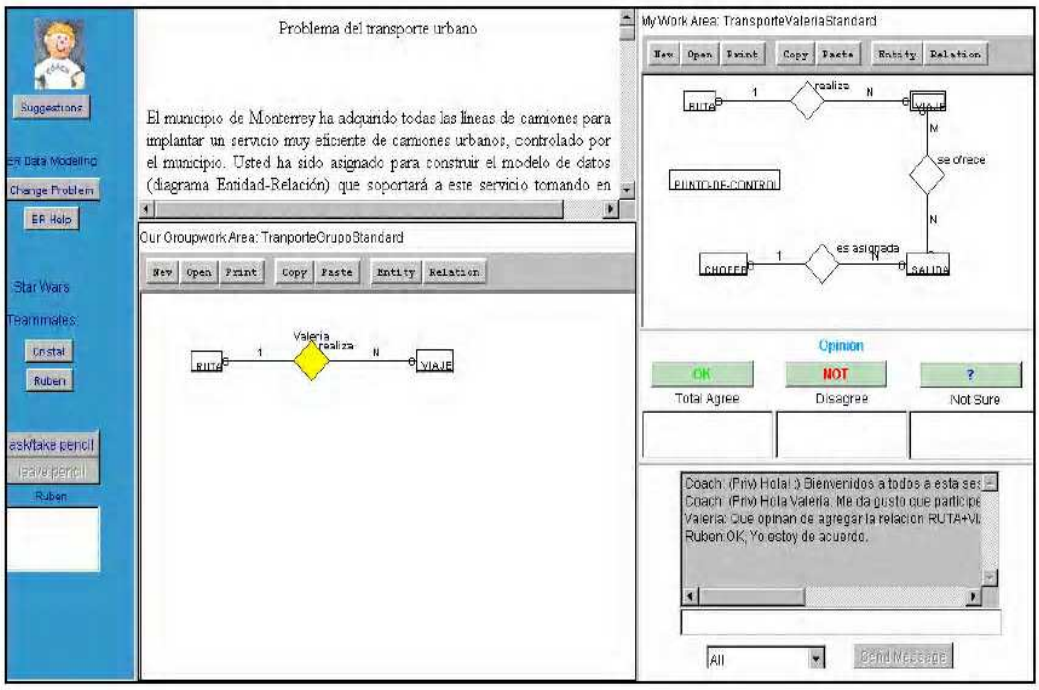



*Figure 2.4: The COLER User Interface [Constantino-Gonzales and Suthers, 2000]*

Seven advice categories (each consisting of several types of advice) are defined. The first two categories, Discussion (in chat) and Participation (in the group workspace), are the main categories related to coaching collaboration. Feedback messages are related to student's pressing of COLER opinion buttons. The ER Modelling category includes suggestions related to common ER modelling mistakes. The Self-Reflection category consists of suggestions that individuals think about a problem or situation. In addition to advice from these categories, COLER can use messages for welcoming and saying goodbye.

The coach's architecture consists of several modules that communicate through a basic blackboard. The main module, the *Collaboration Supervisor* uses an event-based reasoning. It consists of two modules: the *Advice Generator* and the *Advice Selector*. The *Advice Generator* computes the set of appropriate advice based on AND/OR situation trees. Several pieces of advice might be suggested for any given event. The *Advice Selector* chooses the most appropriate advice from this set based on prioritized preferences and random choice. The AND/OR decision trees used by the *Advice Generator* were represented directly in Java code.

Some limitations of the system are as follows:

- The system does not provide ideal solutions to ER problems, assuming that the students would reach a correct answer by discussing and participating during collaborative problem solving
- The information from the chat area is not examined
- The individual work is only assessed against group solution, not against the ideal solution
- Only the action of adding objects to the group diagram is considered as a contribution (updating and deleting actions are not counted as contributions)

#### 2.3.3.4 COMET

COMET [Suebnukarn and Haddawy, 2004] is a collaborative intelligent tutoring system for medical problem-based learning. The system uses Bayesian networks to model individual student knowledge and activity, as well as that of the group. Students

can sketch directly on medical images, search for medical concepts, and sketch hypotheses on a shared workspace. If a student circles or otherwise annotates a region of the image representing a valid hypothesis, it is taken as input to the system and the corresponding hypothesis is added to the hypothesis board. Each student is modelled with an instance of a general Bayesian network student model. The group is reasoned about by combing information from the models of the individual students. The system uses the models and the actions to generate eight hinting strategies to guide group problem solving. The hint strategies are: Focus Group discussion, Create Open Environment for Discussion, Deflect Uneducated Guessing, Avoid Jumping Critical Steps, Address Incomplete Information, Refer to Expert in the Group and Promote Collaborative Discussions.

Figure 2.5 shows the system interface. The system is implemented as a Java client/server combination, which can be used over the Internet or local area networks and supports any number of users.

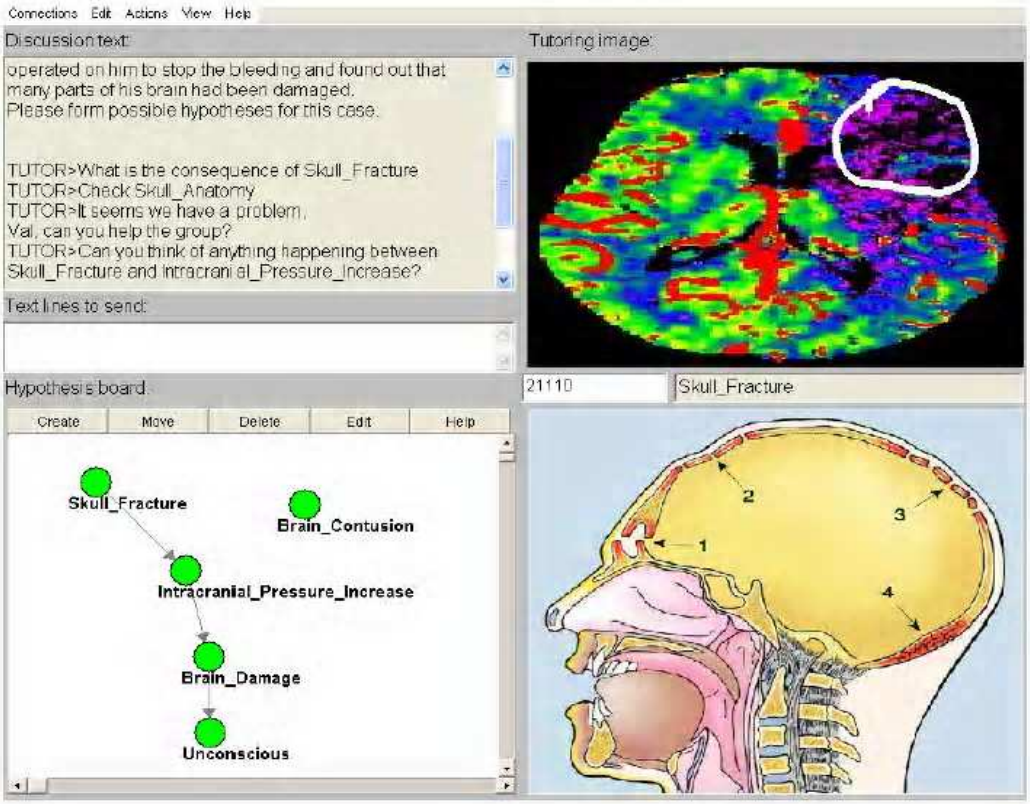



*Figure 2.5: The COMET Interface [Suebnukarn and Haddawy, 2004]*

Some limitations of the system are as follows:

- The system has no language processing capabilities and the text in the chat pane is not taken as input
- The system does not support private actions
- It is not clear from the paper how the tutorial hints are generated
- The paper does not mention anything about the domain knowledge

#### 2.3.3.5 ACT

ACT [Gogoulou, et al. 2005] is a web-based adaptive collaborative learning environment, which supports and guides the learners' communication and collaboration by implementing a structured dialogue either through sentence openers or communication acts. The scaffolding sentence templates (SST) are adapted according to the cognitive skills addressed by the learning activity, the model of collaboration followed and the educational tool used. The learners have the ability to personalize the communication/collaboration process by enriching the provided set of the scaffolding sentence templates with the desired ones. They can also monitor the dialogue progress and reflect on their communication/collaboration by accessing the Dialogue Tree as well as the results of the quantitative analysis of their debate at any time during the elaboration of the activity.

The interface (shown in Figure 2.6) consists of the following areas:

- The *Dialogue Area*: shows the debate which has taken place. The messages are recorded, numbered and presented in a chronological order.
- The *Message Composition Area*: enables the learner to construct the desired message on the basis of the SST provided
- The *Message Submission Area*: enables the learner to submit the message to all or to selected members of the group.

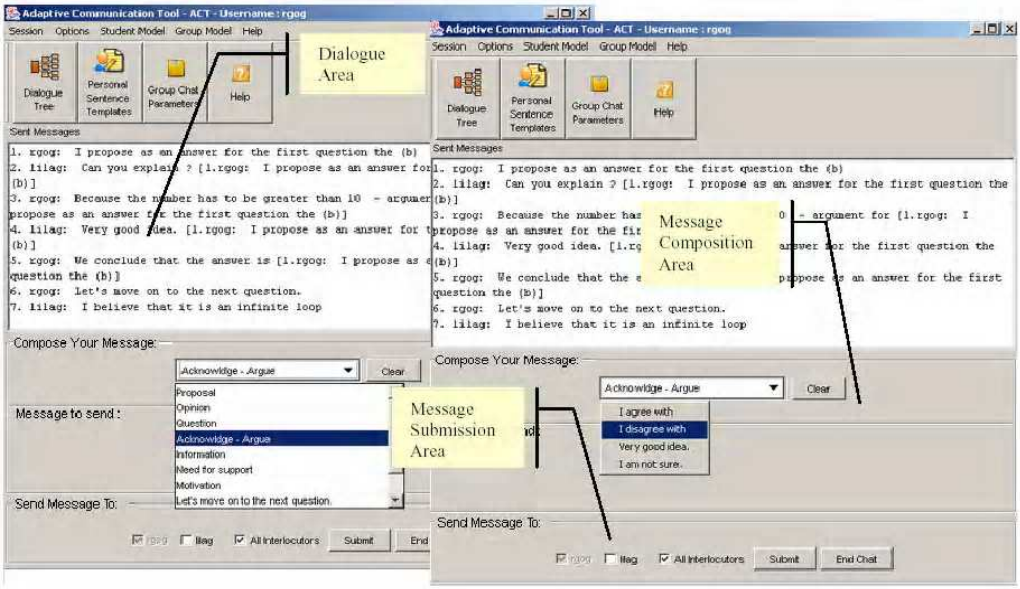



*Figure 2.6: The ACT interface [Gogoulou, et al. 2005]*

### 2.3.3.6 Summary

Table 2.2 summarises these three categories and illustrates their support for collaborative learning.

*Table 2.2: Examples of Different Types of Systems Supporting Collaborative Learning*

| | | |
|---|---|---|
| **Mirroring/Reflecting Tools** | Collect Interaction Data | Plaisant et al. [1999] |
| **Metacognitive/Monitoring Tools** | Construction Model of Interaction, Compare with Desired State | MArCo [Tedesco and Self, 2000] |
| | | EPSILON [Soller and Lesgold, 2000] |
| | | Action-based Collaboration Analysis [Muehlenbrock, 2000] |
| **Advising/Moderating Tools** | Intervene, Advise, Guide | DEGREE [Barros and Verdejo, 2000] |
| | | Group Leader [McManus and Aiken, 1995] |
| | | BELVEDERE [Suthers and Jones, 1997] |
| | | GRACILE [Ayala and Yano, 1998] |
| | | LeCS [Rosatelli, Self, and Thirty, 2000] |
| | | COLER [Constantino-Gonzales and Suthers, 2000] |
| | | COMET [Suebnukarn and Haddawy, 2004] |
| | | ACT [Gogoulou et al. 2005] |

## 2.4 Approaches to Analysing the Collaborative Learning Interaction

In general, a student's understanding of a concept is reflected in their actions and their explanations of these actions. In a one-on-one tutoring environment, this information is available to analyse. The system is able to watch the student solve the problem, perhaps ask pointed questions to evaluate the student's understanding of key concepts, and once in a while, interrupt them if remediation is required. Evaluating the learning of a group of students solving the same problem is a very different scenario. Whereas the key to understanding and supporting computer-supported individual learning lies in evaluating the student's actions, the key to understanding CSCL lies in understanding the rich and complex interactions between individuals [Dillenbourg, 1999; Soller and Lesgold, 2000]. These interaction patterns include information about the students' roles, understanding of the subject matter, engagement, degree of shared understanding, and ability to follow and contribute to the development of ideas and solutions. A CSCL environment that can analyse sequences of learning interaction may be able to determine, for instance, when a student is falling behind in the group.

Understanding and analysing the collaborative learning process needs a fine-grained sequential analysis of the group interaction in the context of the learning goals. The following subsections describe five different computational approaches available in the literature for performing such analysis [Soller and Lesgold, 2000].

### 2.4.1 Finite State Machines

McManus and Aiken's [1995] Group Leader system compares sequences of students' conversation acts to those allowable in a four state finite state machine developed specifically to monitor discussions about comments, requests, promises, and debates. The Group Leader can analyse sequences of conversation acts, and provide 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 it. The Group Leader received a positive response from students, and smoothed the way for further research along these lines.

DEGREE [Barros and Verdejo, 1999] is an asynchronous newsgroup-style environment that enables students to have structured, computer-mediated discussions on-line (See Section 2.3.3 for a detailed description). This work is seminal in combining a finite state approach with fuzzy rubrics to structure and understand the group interaction. "A closer look at interaction sequences containing both task and conversational elements may help in composing rubrics for dynamically evaluating learning activity, enabling a facilitator agent to provide direction at the most appropriate instances" [Soller and Lesgold, 2000].

### 2.4.2 Rule Learners

Katz, Aronis, and Creitz [1999] developed two rule learning systems, String Rule Learner and Grammar Learner that learn patterns of conversation acts from dialog segments that target particular pedagogical goals. The rule learners were challenged to find patterns in the hand-coded dialogs between avionics students learning electronics troubleshooting skills and expert technicians. The conversations took place within the SHERLOCK 2 for electronics troubleshooting.

The String Rule Learner, which searches for patterns common to a training set, discovered that explanations of system functionality often begin with an *Identify* or *Inform* Act. The Grammar Learner, which develops a probabilistic context-free grammar for specified conversation types, learned that explanations of system functionality not only begin with an *Inform* statement, but may continue to include a causal description, or another *Inform* Act followed by a *Predict* Act. Rule learning algorithms such as these are promising for classification and recognition tasks, and may prove useful tools for supporting the sequential analysis of learning conversations.

### 2.4.3 Decision Trees and Plan Recognition

COLER [Constantino-Gonzales and Suther, 2000] coaches students as they collaboratively learn Entity-Relationship modelling. Decision trees that account for both task-based and conversational interaction are used to dynamically give feedback to the group. For further details, refer to Section 2.3.3.

Muhlenbrock and Hoppe [1999] take a plan recognition approach to analysing collaboration processes. In their approach, the system maps actions taken on a shared

workspace to steps in a partially ordered, hierarchical plan. The hierarchical nature of the plan allows users' individual actions to be generalized to problem solving activities (e.g. conflict creation or revision). Muhlenbrock and Hoppe show that the group members' roles can be decided upon by analysing how these general problem solving activities shift focus from one user to another.

### 2.4.4 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 which describe the students' interaction in these two cases (See Section 2.3.2 for more details).

## 2.5 Summary

This chapter discussed the main components of ITSs, presented a description of the constraint-based modelling technique and gave examples of some of CBM tutors developed so far. We then described three main types of supportive collaborative learning systems: systems that reflect actions (mirroring systems), systems that monitor the state of interaction (meta-cognitive tools), and finally, coaching or advising systems which guide the collaborators by recommending actions they might take to improve their interaction. Some examples of collaborative learning systems were provided and their advantages and disadvantages were examined. Finally, we described the approaches used in current literature to analyse the collaborative learning interaction.

# Chapter 3

# COLLECT-UML: Single-User Version

This chapter presents our experiences in implementing COLLECT-UML, a constraint-based ITS in the area of object-oriented (OO) analysis and design using the Unified Modelling Language (UML). The chosen task is very complex, as it requires sound knowledge of requirements analysis, OO design and UML class diagrams. 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 modelling, 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.

We started off by developing a single-user version of the system (described in this chapter) and extended it later on to support collaborative learning. The multi-user version of the system is described in the next chapter.

This chapter is organized as follows. The chosen instructional domain is presented in Section 3.1. Section 3.2 describes the overall architecture of the system, followed by the description of the domain constraints in Section 3.3. Section 3.4 discusses the interface of the single-user version. Feedback generation is discussed in Section 3.5. Section 3.6 presents the results of two evaluation studies performed, followed by a summary in the last section.

## 3.1 Difficulties of Learning Object-Oriented Modelling

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. When OO first entered the mainstream of software engineering, it served (only) as a programming language paradigm. Subsequently, its influence broadened to provide a paradigm for the design of software, known as Object-Oriented Design (OOD), and broadened yet further to encompass Object-Oriented Analysis (OOA). In OO analysis, the same OO principles for structuring systems are used when performing requirements analysis to represent the concepts, behaviours and relationships found in some problem domain.

OO systems consist of *classes* (with *structure* and *behaviour)*, and *relationships* between them. Relationships have multiplicity, names and can be of different types (association, aggregation, composition, inheritance or dependency). In OOA and OOD, these structures exist largely 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. Software engineering courses that teach OO analysis and design typically do so using UML.

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

Considering the fact that the failures of a significant percentage of developed systems are linked to faulty requirements, it is very important to ensure the quality of conceptual models developed in the early stages of systems development. Developing good quality conceptual models is a challenging task for many systems analysts [Bolloju and Leung, 2006]. 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.

Let us illustrate the process of designing a class diagram on a simple example. A student is given the following description of the target system:

*Design a class diagram for a School. A school is known by its name, address, and phone number and has one or more departments. Each department has a name and is assigned a number of instructors. Each instructor has a name and teaches several courses within the department. Each course is known by its name and course ID. A student has a name and student ID and attends a number of courses offered by the department. The school has a number of students and can add students, remove students, add departments and remove departments. Students may enrol in a number of courses, drop courses and transfer credits. Each department can add instructors and remove instructors.*

From the description, the classes *school*, *department*, *student*, *course*, and *instructor* can be identified. The student may start by drawing these classes first. For each class, attributes and methods are described. For example, each department contains a name, and methods to add and remove instructors. All the attributes and methods are explicitly mentioned in the requirements.

The student also needs to identify the relationships between these classes. For example, each school has one or more departments, and this is mentioned in the second sentence of the problem text. The student needs to decide which relationship type would be most appropriate to use. Once all the relationship types are identified, the student needs to determine the multiplicities and names of the relationships.

The UML class diagram for the *School* software system is illustrated in Figure 3.1. As can be seen from this simple case, there are many things that the student has to know and think about when developing a UML diagram. The student must understand both the basic building blocks available and the restrictions specified on them. In real situations, the text of the problem is likely to be much longer, often ambiguous and incomplete. The student must be able to reason about the requirements and use his/her own world knowledge to make reasonable assumptions. UML modelling is not a well-

defined process, and the task is open ended. There is no algorithm to derive the UML class diagram for a given set of requirements. There is no single, best solution for a problem, and often there are several correct solutions for the same requirements. In our experience students typically have many problems learning how to construct good quality OO models.

Although the traditional method of learning UML modelling in a classroom environment may be sufficient as an introduction to the concepts of OO analysis and design, students cannot gain expertise in the domain by attending lectures only. Even if some effort is made to offer students individual help through tutorials, a tutor must cater for the needs of the entire group of students, and it is inevitable that they obtain only limited personal assistance. Therefore, the existence of a computerized tutor, which would support students in acquiring such design skills, would be highly useful.

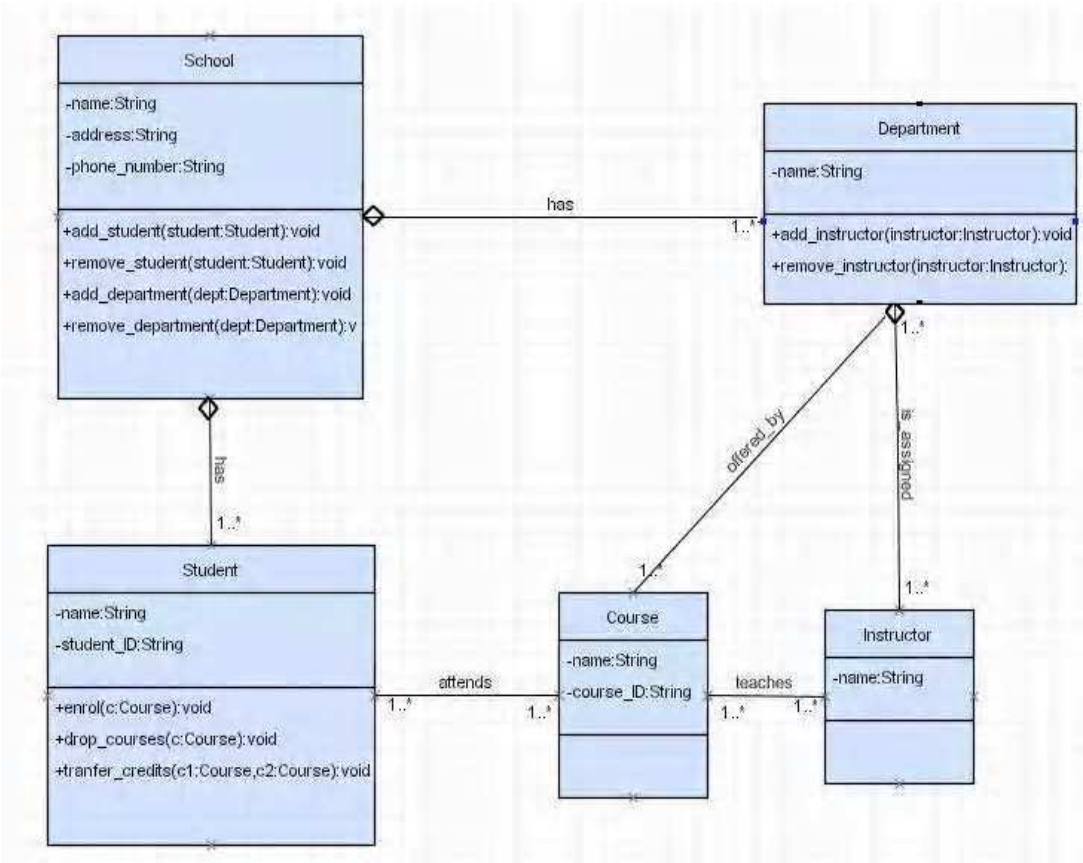



*Figure 3.1: The UML Class Diagram for School*

The most relevant work to this thesis is an ITS for learning UML class diagrams [Blank, et al. 2005] created at the Lehigh University. The system, called CIMEL ITS, is still under development and has not been evaluated yet. It coordinates student learning in web-based multimedia courseware (CIMEL) and the Eclipse IDE[1], each posting student interactions to a server-based CIMEL ITS. The system analyzes the students work and provides real-time guidance when they make mistakes. The plug-in also automatically generates Java code from their class diagrams. The most important difference between their system and COLLECT-UML is that it does not use constraint-based modelling. Instead, it has an expert module that generates design solutions to input problems and evaluates student work. It also has a student model based on a Bayesian network, and a pedagogical agent that chooses feedback based on the student model and the student's learning style.

## 3.2 The Architecture of COLLECT-UML

COLLECT-UML is a web-based problem-solving environment, in which students are required to construct UML class diagrams that satisfy a given set of requirements. 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 OO software design and UML. It assists students during problem solving and guides students towards a correct solution by providing feedback.

COLLECT-UML has a distributed architecture [Mitrovic, 2003], where the tutoring functionality is distributed between the client and the server (Figure 3.2).

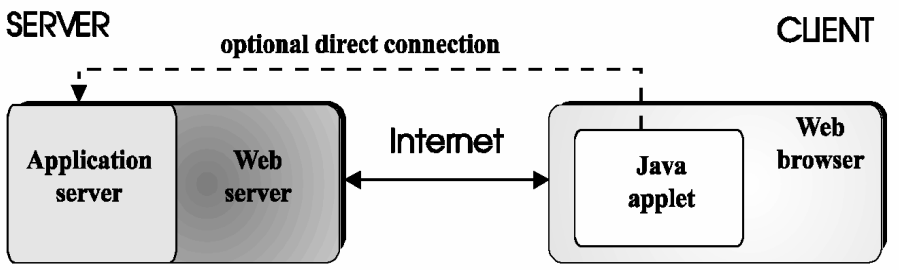



*Figure 3.2: Distributed Architecture*

[1] Eclipse is an open-source and free integrated development environment. For more details, see http://www.eclipse.org.

Because the task is very demanding and interactive, it was desirable to perform some pedagogical action on the client, in order to speed up interaction. The client intervenes in situations when the student makes simple syntax errors, such as submitting a diagram with missing component names. The system is implemented in WETAS [Martin and Mitrovic, 2002; 2003], a constraint-based authoring shell. WETAS itself is implemented in Allegro Common Lisp, which provides a development environment with an integrated Web Server [AllegroServe, 2005].

The system's components are illustrated in Figure 3.3. 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 modeller 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 modeller diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module, which generates appropriate feedback.

COLLECT-UML does not have a problem solver, as developing a general problem solver for UML modelling is extremely difficult. One of the major obstacles that would have to be overcome is natural language processing (NLP), as the problems in the domain are presented using natural language text. However, the NLP problem is far from being solved. Other complexities arise from the nature of the task. There are assumptions that need to be made during the development of UML diagrams. These assumptions are outside the problem description and are dependent on the semantics of the problem itself. Although this obstacle can be avoided by explicitly specifying these assumptions within the problem description, ascertaining these assumptions is an essential part of the process of constructing a solution and would over-simplify the problems.

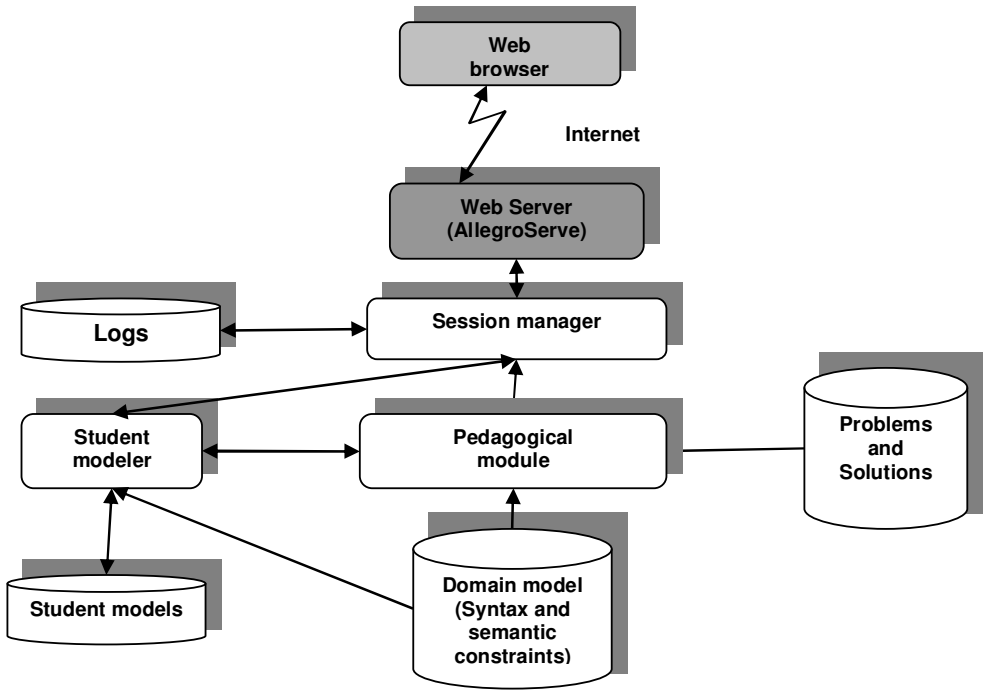



*Figure 3.3: The Architecture of the Single-User Version of the System*

Although there is no problem solver, COLLECT-UML is capable of diagnosing students' solutions. The system contains an ideal solution for each problem, which is compared to the student's solution according to the system's domain model, represented as a set of constraints (described in the next section).

## 3.3 Domain Constraints

The system's domain model contains 88 semantic and 45 syntax constraints that describe the basic principles of the domain. Semantic constraints are usually more complex than syntax constraints. In order to develop constraints, we studied material in textbooks, such as [Fowler, 2004], and also used our own experience in teaching UML and OO analysis and design.

The framework of Lindland et al. [1994] offers a systematic way to analyze the quality of UML artifacts from the syntactic, semantic and pragmatic quality perspectives. Some examples are presented in [Bolloju and Leung, 2006] of common errors belonging to these three quality categories including:

- Missing cardinality details for associations
- Missing attributes and methods
- Inappropriate naming of classes and associations
- Incorrect cardinality specification
- Used aggregation instead of association
- Redundant associations
- Specialization with little distinction among subclasses

The semantic and syntax constraints developed check for all these possible mistakes.

Figure 3.4 illustrates three constraints[2] from the UML domain. Constraint 41 is a syntax constraint; it checks that there are some attributes or methods defined for each class in the student's solution. The constraint contains a message which would be given to the student if the constraint is violated. Constraint 49 is a semantic constraint. Its relevance condition identifies a subclass in the ideal solution, and then checks whether the student's solution 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.

[2] For readability reasons, the constraints are given in the structured English form, rather than in the original form, using the proprietary constraint language.

(41
IF [relevance condition]
the student's solution has a class called *C*
THEN [satisfaction condition]
It is necessary that the student's solution has an attribute or a method of *C*
ELSE
Message: "**Check your classes. Each class must have at least one attribute or method.**"

(49
IF [relevance condition]
the ideal solution has a subclass called *C*,
AND the student's solution has a class with the same name
THEN [satisfaction condition]
It is necessary that *C* is a subclass in the student's solution too.
ELSE
Message: "**Check whether you have defined all required subclasses. Some subclasses are missing**."

(52
IF [relevance condition]
The ideal solution has a superclass called *C1,*
AND the ideal solution has a subclass of *C1* called *C2*,
AND the ideal solution has a method of *C1* named *m,*
AND the ideal solution has a method of *C2* named *m,*
AND the student's solution has a superclass called *C1,*
AND the student's solution has a subclass of *C1* called *C2,*
AND the student's solution has a method of *C1* named *m*
THEN [satisfaction condition]
It is necessary that the student's solution has a method of *C2* named *m*
ELSE
Message: "**Check your inheritance relationships. Some of your subclasses must override one or more methods defined in the superclass. The ability of a subclass to override a method in its superclass allows a class to inherit from a superclass whose behavior is similar, and then override methods as needed.**"

*Figure 3.4: Examples of Constraints from COLLECT-UML*

Constraint 52 is also a semantic constraint. Its relevance condition identifies a superclass and a subclass in the ideal solution, which have the same method defined. Then, the relevance condition looks for a matching class and a superclass in the student's solution, with the same method defined for the superclass. The student's solution is correct if there is a method with the same name defined in the subclass, which overrides the method defined in the superclass.

The short-term student model consists of a list of violated and a list of satisfied constraints for the current attempt. The long-term model records the history of usage

for each constraint. This information is used to select problems of appropriate complexity for the student, and generate feedback.

## 3.4 Interface

Students interact with COLLECT-UML via its interface (Figure 3.5) 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 (Figure 3.6), 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. The applet was implemented using Java 1.4, and contains 4 packages, 75 Java classes and 6853 lines of code. Feedback is presented on the right, while the bottom part allows the student to select the feedback level, and submit solutions.

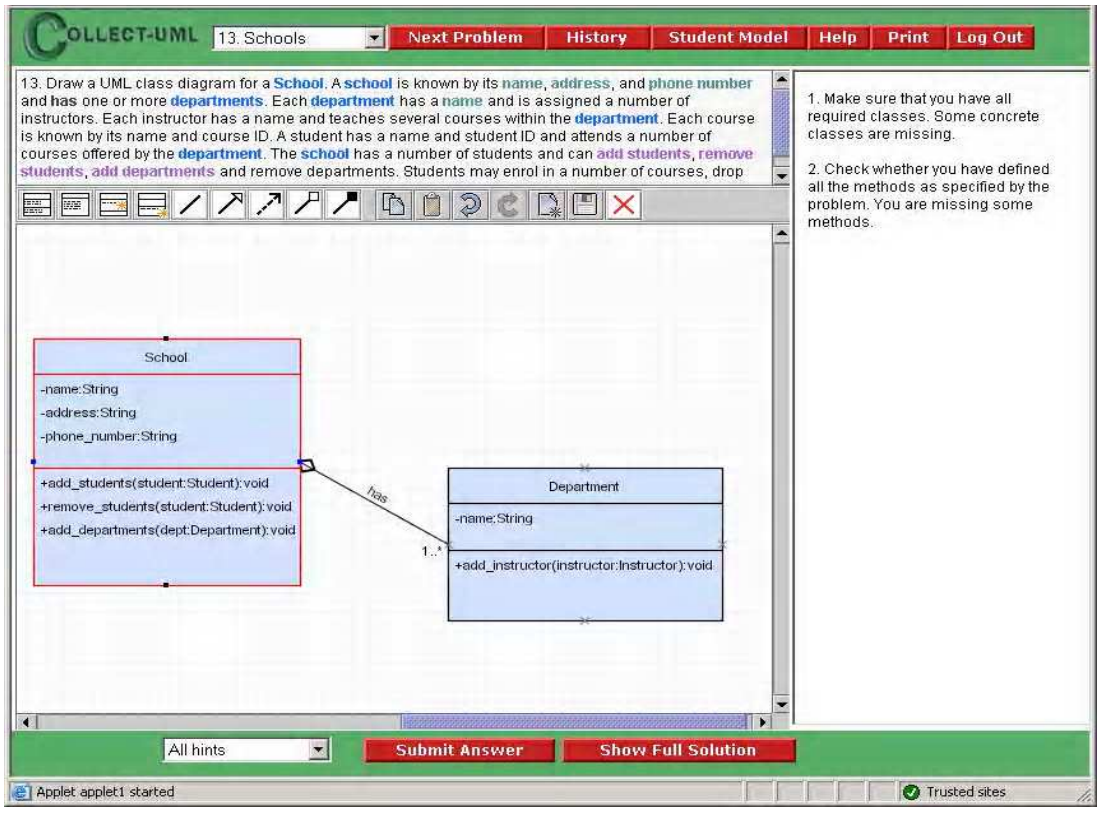



*Figure 3.5: The Interface of COLLECT-UML*

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 can be seen from Figure 3.5, the applet contains a drawing bar with UML constructs. Students can therefore remind themselves of the basic building blocks to use when drawing UML diagrams. The symbols used for UML modelling are shown in Figure 3.7. 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

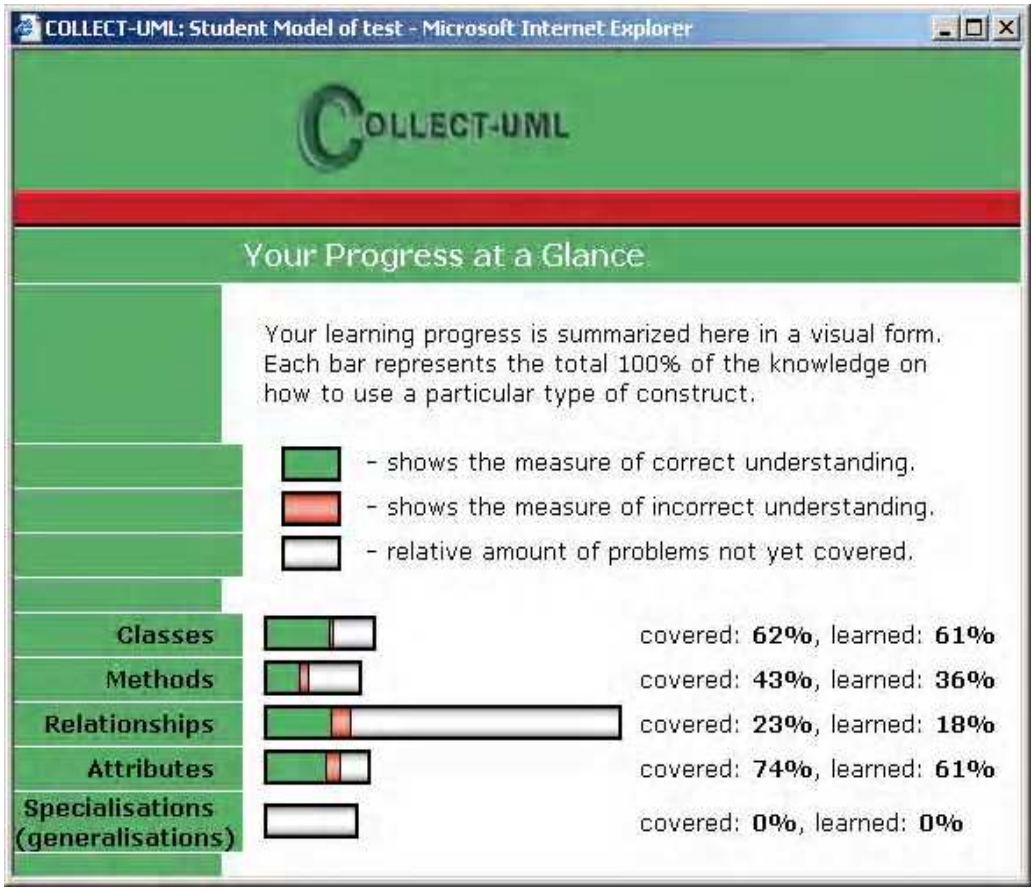



*Figure 3.6: The Open Student Model*

place within the drawing area.

To create a new attribute or a method, the student needs to select the parent class first, and then either click on relevant toolbar icon or right-click on the class and choose the "New ..." menu option (See Figure 3.9, 3.11). They can change the properties of an

existing component (class, attribute, method or relationship) by right-clicking on that component and choosing the relevant menu option (See Figure 3.10, 3.12). To connect two classes of the diagram, they need to select the appropriate type of relationship. A relationship will be shown in red if it is not properly attached to other classes. The system stores geometric information about the way the diagram is laid out, which is used to extract the attribute and methods belong to each class and the relationships between the classes. It is also used for future logins to the system, so that the user can see the exact same diagram, if they want to work on it in multiple sessions.

COLLECT-UML requires the student to name each newly added construct by using a word/phrase from the problem text as its name. A name can be selected by highlighting a phrase from the problem text. It is not possible to name a construct by typing. This is useful from the point of view of the student modeller for evaluating solutions [Suraweera and Mitrovic, 2002] for several reasons. There is no standard that is enforced in naming classes, methods, attributes or relationships. If the student can enter names freely, the names of the components in the student solution may not match the names of construct in the ideal solution (IS), and the task of finding correspondence between the constructs of the SS and IS is difficult. This problem is avoided by forcing the student to use names that come from the problem text directly.

This requirement enforces two of the most important practices in software design: using the end-users' language and reflecting on the requirements. By selecting names for various diagram components directly from the problem text, the student has to think about the requirements. The interface highlights the previously selected parts of the problem text that correspond to various types of UML constructs using different colours, making it easier for the student to review how much of the problem has been covered. Subjective evaluation of the system (described in Section 3.6) showed that several participants pointed out this feature, when asked what they liked in particular about COLLECT-UML.

Currently, the system contains 14 problems, which cover different aspects of UML modelling, and their ideal solutions. Figure 3.8 shows problem 13 (also shown in Figure 3.5), with the difficulty level of 10, the statement of the problem, the internal representation of its ideal solution, followed by the name of the file containing the correct UML class diagram and finally the problem name (*Schools*). Ideal solutions are represented in the form of 6 lists (*RELATIONSHIPS, ATTRIBUTES, METHODS, CLASSES, SUPERCLASSES* and *SUBCLASSES*). Each list contains the items expected to be found in student's solution. Items within lists are separated with the @ symbol. For example, E1A2 is an attribute named *address*, which belongs to class E1 (*school*), is of type string, is private and is not static. This information is represented in ideal solution (in the ATTRIBUTES list) as: *@ E1A2 address E1 String private no.* The problem text is represented internally with embedded tags that specify the mapping to the constructs in the ideal solution. The tags are not visible to the student since they are extracted before the problem is displayed. A solution contains an explicit set of superclasses pre-computed for efficiency reasons.

The applet saves the solutions submitted by students as XML files, which are converted to internal representation using an XSLT style-sheet. The constraints are

| Symbol | Component |
|---|---|
|  | Concrete Class |
|  | Interface |
|  | Attribute |
|  | Method |
|  | Association |
|  | Inheritance |
|  | Dependency |
|  | Aggregation |
|  | Composition |

*Figure 3.7: UML Components Supported by the System*

applied to the internal representation of the solutions and feedback is given to students, using messages attached to the violated constraints.

```
(13                ; problem number
10                 ; difficulty
"Draw a UML class diagram for a <E1> School </E1>. A <E1> school </E1> is known by its
<E1A1> name </E1A1>, <E1A2> address </E1A2>, and <E1A3> phone number </E1A3> and
<R1> has </R1> one or more <E2> departments </E2>. Each <E2> department </E2> has a
<E2A1> name </E2A1> and <R2> is assigned </R2> a number of <E3> instructors </E3>. Each
<E3> instructor </E3> has a <E3A1> name </E3A1> and <R3> teaches </R3> several <E4>
courses </E4> within the <E2> department </E2>. Each <E4> course </E4> is known by its
<E4A1> name </E4A1> and <E4A2> course ID </E4A2>. A <E5> student </E5> has a ..."

(("RELATIONSHIPS" "@ R1 aggregation E1 E2 null 1..* null null has @ R2
aggregation E2 E3 null 1..* null null is_assigned @ R3 association E4
E3 1..* 1..* null null teaches ...")
("ATTRIBUTES" "@ E1A1 name E1 String private no @ E1A2 address E1 String
private no @ E1A3 phone_number E1 String private no @ E3A1 name E3
String private no @ E4A1 name E4 String private no @ E4A2 course_ID E4
String private no @ E5A1 name E5 String private no ...")
("METHODS" "@ E1A4 add_student E1 void public no 1 student_ID String null
null null null @ E1A5 emove_student E1 void public no 1 Student_iD
String null null null null ...")
("CLASSES" "@ E1 School concrete @ E3 Instructor concrete @ E4 Course
concrete @ E5 Student concrete @ E2 Department concrete ")
("SUPERCLASSES" "")
("SUBCLASSES" ""))
"13.jpg"
"Schools")
```

*Figure 3.8: A Sample Problem and its Ideal Solution*

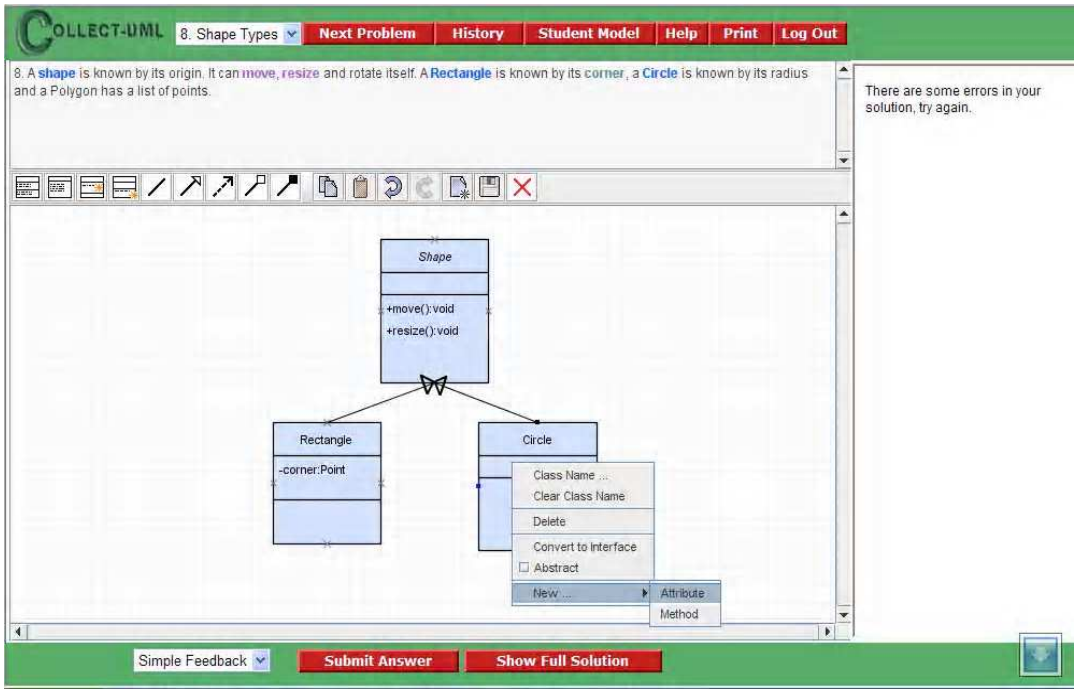



*Figure 3.9: Creating New Attributes*

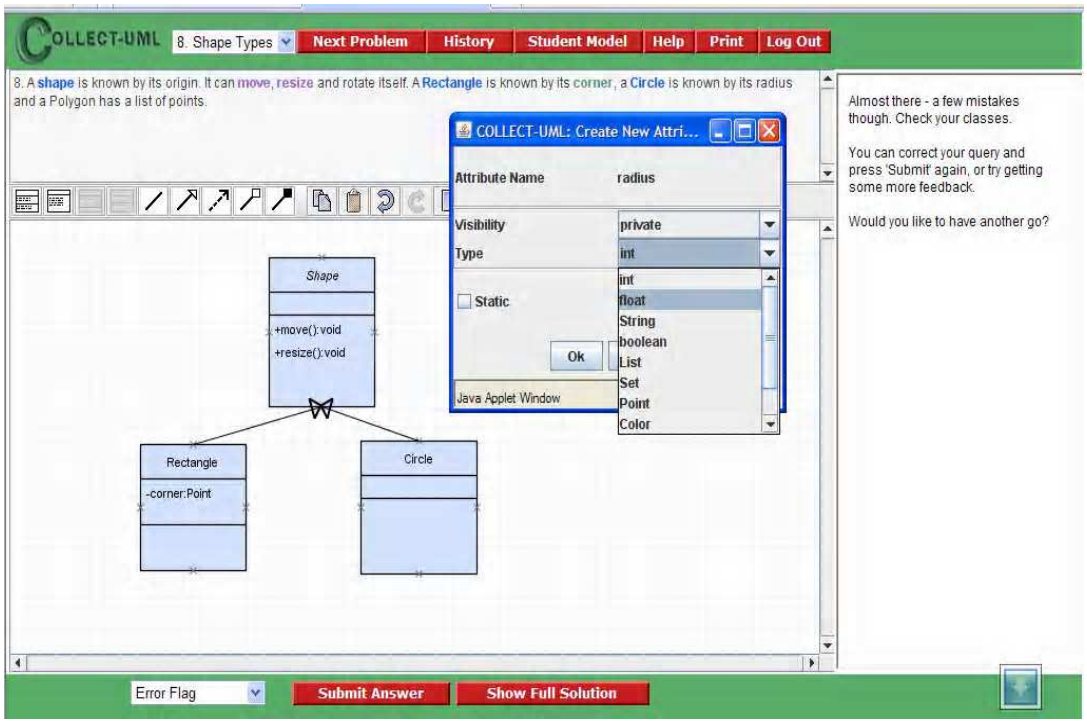



*Figure 3.10: Specifying the Properties of a New Attribute*

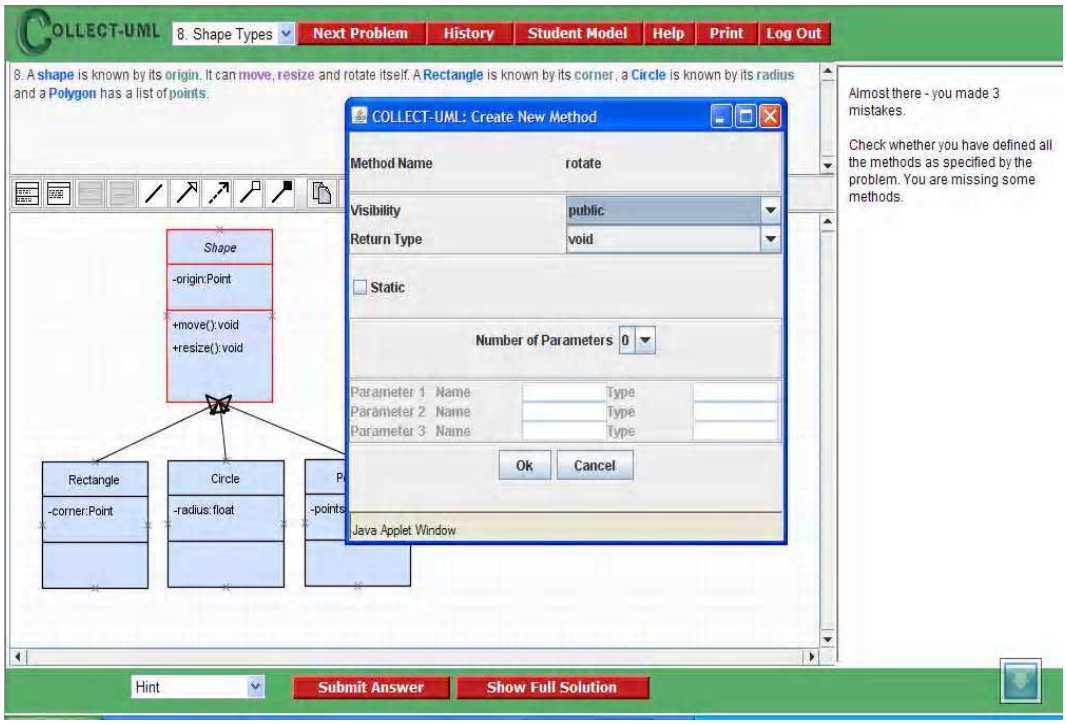



*Figure 3.11: Creating a New Method*

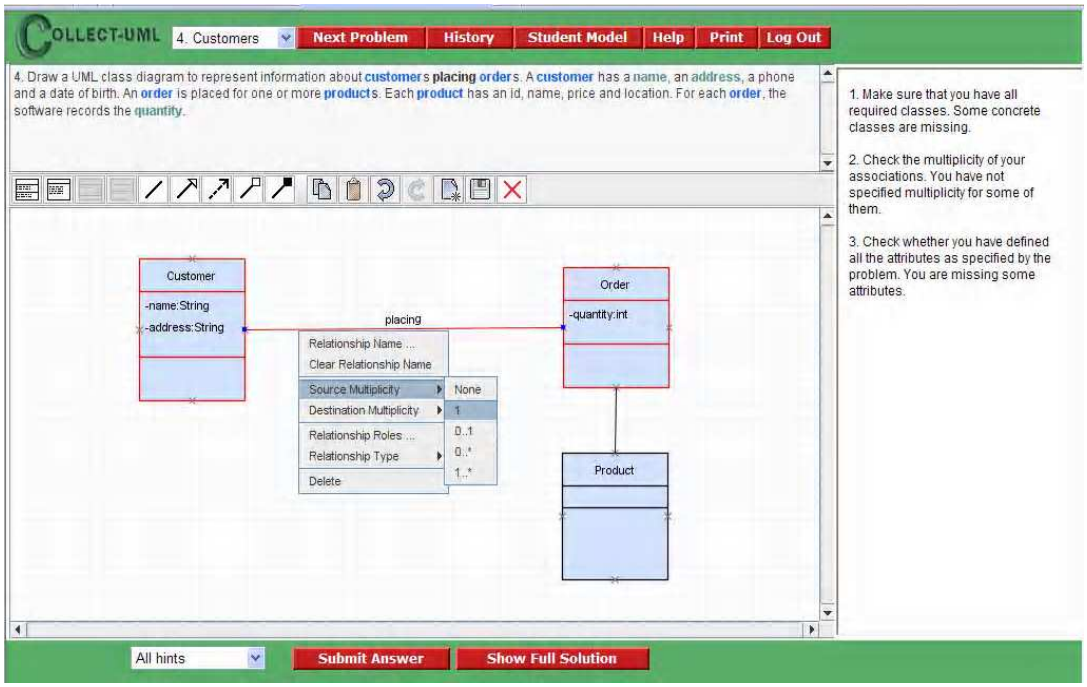



*Figure 3.12: Specifying Multiplicity for an Association Relationship*

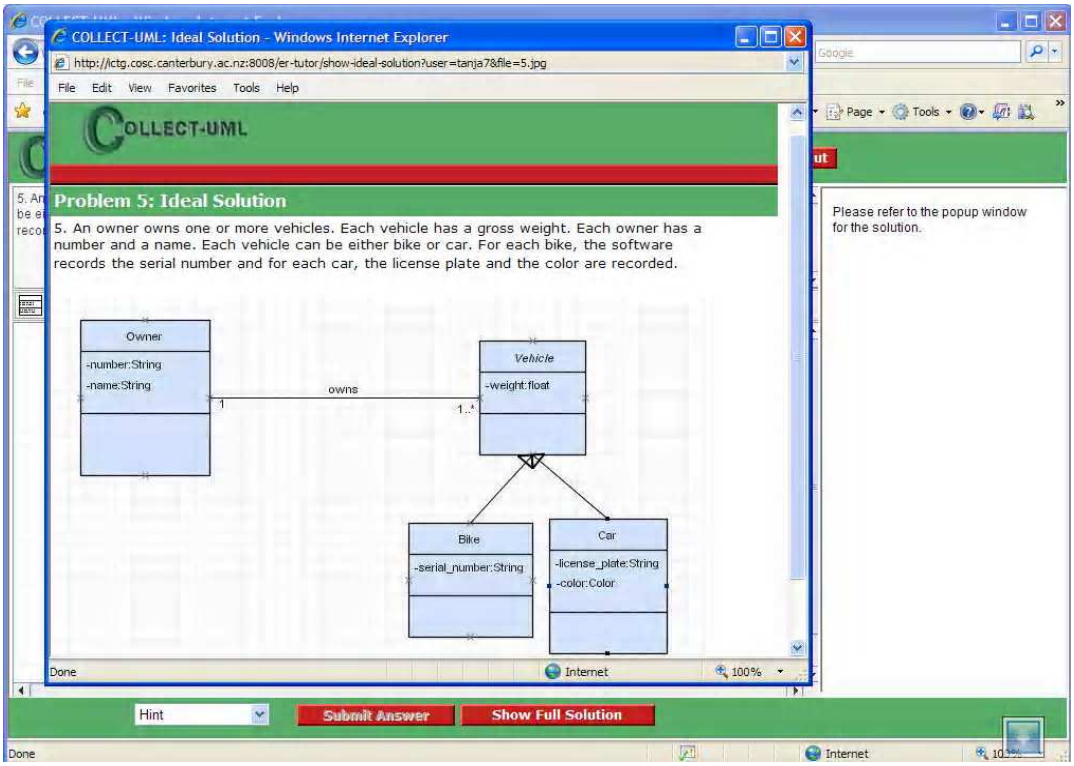



*Figure 3.13: Displaying Full Solution*

## 3.5 Feedback Generation

COLLECT-UML evaluates the student's solution once it is submitted, and provides feedback. During evaluation, the student modeller identifies the constraints that the student has violated. The feedback is offered at five levels of detail: *Simple Feedback*, *Error flag*, *Hint*, *All Hints* and *Full solution*. The first level of feedback simply indicates whether the submitted solution is correct or incorrect (Figure 3.9). The *Error flag* indicates the type of construct (e.g. class, relationship, method, etc.) that contains the error (Figure 3.10). *Hint* offers a feedback message generated from the first violated constraint such as "*Make sure that you have all required classes. Some concrete classes are missing.*" A list of feedback messages on all violated constraints is displayed at the *All hints* level (as shown in Figures 3.5 and 3.12). The UML class diagram of the complete solution is displayed in a separate window (see Figure 3.13) when the user clicks 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 incremented with each submission until the feedback level reaches the *Hint* level. In other words, if the student 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 levels 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.

## 3.6 Evaluation

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

### 3.6.1 Pilot Study

A pilot study was conducted as a think-aloud protocol in March 2005. The study aimed to discover students' perceptions about various aspects of the system, mainly the quality of feedback messages and the usability of the interface. The participants were 12 postgraduate students enrolled in an Intelligent Tutoring Systems course at the University of Canterbury. At the time of the study, the participants had completed 50% of the ITS course lectures, and were expected to have a good understanding of ITS. All participants except two were already familiar with UML modelling.

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 verbalise his/her thoughts while performing a UML modelling task using COLLECT-UML. Participants were able to skip the problems without completing them and to return to previous problems. 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 the drawing area was big enough for them to work on the problems given. Three participants felt that some of the hints provided by the system were not helpful enough for them to correct their mistakes. The difficulty with the feedback came from the students not being able to interpret given messages. For example, the feedback message of constraint 41 (Figure 3.4) is "*Check your classes. Each class must have at least one attribute or method*." If the diagram contains many classes, the student might have difficulty identifying the class that the feedback message is relevant for. We have modified the system to highlight the part of the diagram related to the feedback

message in red, making it easy for students to localize errors. Two participants also expressed their desire to have access to a glossary and a tutorial on how to use the system.

In order to name a new component (class, attribute, method or relationship), the students were required to highlight words/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. Pop-up dialog windows were added to help the users with naming the classes/methods/attributes once they were created.

The majority of the participants felt that the feedback messages helped them to understand the domain concepts they had found difficult. For this study, the feedback level was restricted to *All Hints* only. For the full evaluation study (described in the next section), the system was modified to include five different levels of feedback (i.e. *Simple Feedback*, *Error flag*, *Hint*, *All Hints, Full Solution)*.

### 3.6.2 Evaluation Study

The evaluation study was carried out at the University of Canterbury in May 2005, after COLLECT-UML was enhanced in the light of the findings from the pilot study. The study involved 38 volunteers from students enrolled in the Introduction to Software Engineering course offered by the Computer Science and Software Engineering department. This second year course teaches UML modelling as outlined by Fowler [2004]. The students learnt UML modelling concepts during two weeks of lectures and had some practice during two weeks of tutorials prior to the study.

The study was conducted in two streams of two-hour laboratory sessions. Each participant sat a pre-test, interacted with the system, and then sat a post-test and filled a user questionnaire. The pre-test and post-test (given in Appendix A) each contained four multiple-choice questions, followed by a question where the students were asked to design a simple UML class diagram. Both tests included questions of comparable difficulty, dealing with inheritance and association relationships. We did not have a control group in this study, because we were interested in the functionality of the system as a whole, not a particular feature.

Table 3.1 presents some general statistics about the study. The participants spent two hours interacting with the system, and solved half of the problems they attempted.

*Table 3.1: Some Statistics about the Study*

| | Average | s. d. |
|---|---|---|
| Time spent on problem solving (hours) | 1.52 | 0.43 |
| Attempted problems | 5.71 | 2.59 |
| Solved problems | 47% | 33% |
| Attempts per problem | 7.42 | 4.76 |
| Pre-test | 52% | 21% |
| Post-test | 76% | 17% |

**Learning**

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% (Table 3.1). The students' performance on the post-test was significantly better ($t = 2.71$, $p = 4.33E\text{-}08$).

We have also analyzed the log files, in order to identify how students learn the underlying domain concepts. Figure 3.14 illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all constraints and all participants (*All constraints*). The data points show a regular decrease, which is approximated by a power curve with a close fit of 0.93, thus showing that students do learn constraints over time. The probability of 0.19 for violating a constraint on the first occasion of application has decreased to 0.09 at its tenth occasion, displaying a 47% decrease in probability.

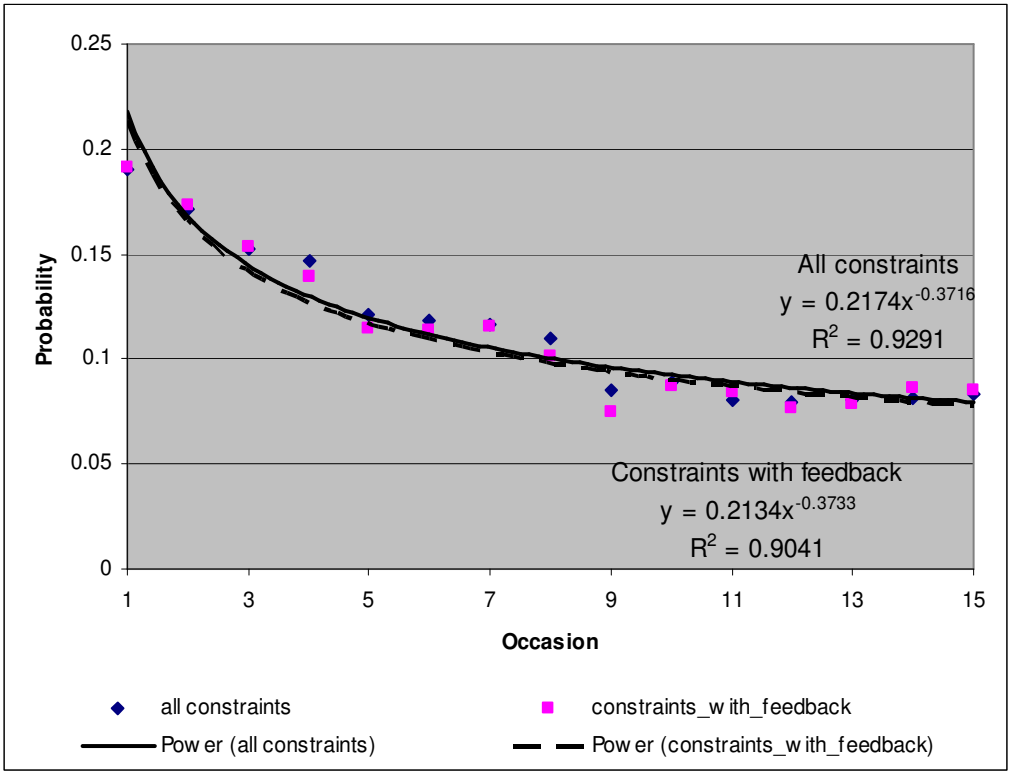



*Figure 3.14: Probability of Constraint Violation*

The other power line in Figure 3.14 labelled *Constraint with feedback* illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all participants, but only for constraints on which participants obtained specific feedback (i.e. when the participants asked for *Hint* or *All Hints* feedback levels). The student logs show that 53% of the participants asked for the *All Hints* feedback level. The *All constraints* learning curve has 2053 data points at the first occasion, while the *Constraints with feedback* curve has 40% less, because students often received feedback other than hints. The learning curve is again very regular, with a very high $R^2$ fit (0.9), and almost identical initial error probability (0.19) and learning rate (-0.37). We believe that the difference between these two curves is small because the participants often could recover from their errors by being shown where the error is (i.e. by being given the *Error Flag* feedback), or could correct slips by being told that there are problems in their solutions (*Simple feedback*).

We found out that 22 constraints 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 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 checks the correct use of dependencies between classes; 5) constraints that check inheritances in students' diagrams, making sure that there are no cycles, and finally 6) a constraint that makes sure each subclass is connected to a superclass.

There were also five constraints that were never satisfied, meaning that the participants did not learn the corresponding domain concepts during the session. The constraints in this group cover aggregation and composition, making sure that the student has used the correct UML construct to represent them. Also this group includes a constraint that checks that multiple inheritances are only specified for interfaces.

We have also looked at learning curves of individual constraints, trying to identify constraints that were especially difficult for the students. Figure 3.15 illustrates the learning curves for three constraints which were hard for participants. The learning rates for all three constraints are much lower that the ones in Figure 3.14, as well as the $R^2$ fit. Constraint 68, the most difficult of the three, checks whether the participant has specified the types of attributes correctly. Constraint 69, the second hardest, checks whether static attributes were specified as such. Finally, constraint 51 checks whether the correct parameters have been specified for methods. In all three 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 predominantly correct. Furthermore, we have noticed that some problem texts do not contain enough detail for the student to be able to complete the relevant parts of the solution, and therefore we believe that the probability of violating these constraints could be decreased by including more detail in the problem descriptions.

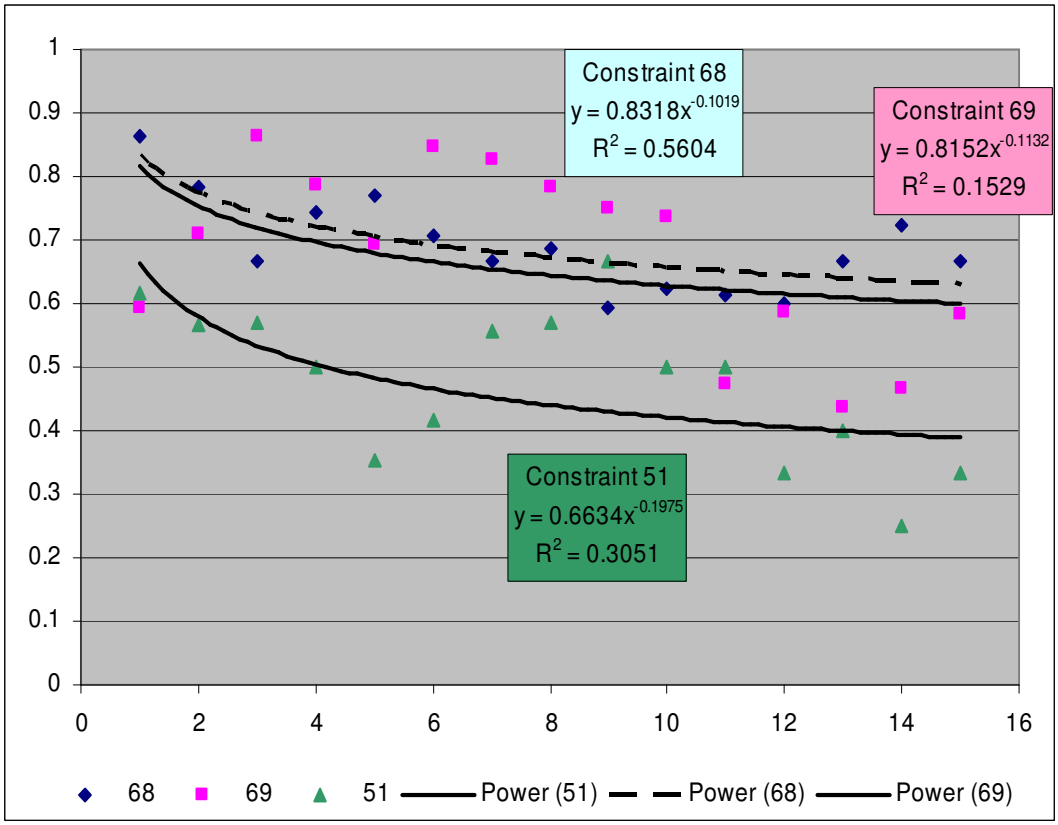



*Figure 3.15: Learning Curves for Three Difficult Constraints*

**Subjective analysis**

All the participants were given a questionnaire (Appendix A.3) at the end of their session to determine their perceptions of the system. Table 3.2 presents a summary of the responses. The students found the interface easy to learn and use. 60% of the participants were familiar with UML modelling from lectures and some work, and the rest had previous experience only from the lectures. Most of the participants (65%) responded they would recommend the system to other students.

The mean response when asked to rate how much they learnt by interacting with COLLECT-UML was 2.9, on the scale of 1 (nothing) to 5 (very much). As Table 3.1 shows, the students spent 1.52 hours on problem solving in average. Some participants indicated that they would have learnt a lot, if they had more time to interact with the system.

Students were offered individualised feedback on their solutions upon submission. The mean rating for the usefulness of feedback was 2.8. 67% of the participants had indicated that they would have liked to see more details in the feedback messages, especially the ones dealing with types of attributes and number of parameters for each method. These two common remarks pointed out that the problem texts did not contain enough information for students to make correct decisions related to these issues, as we have already noted from the analysis of individual constraints' learning curves. The problem texts were later modified in the multi-user version (described in the next chapter), in order to provide such information. The comments we received on open questions also pointed out several features of the system, which can be improved.

*Table 3.2: Mean Responses from the User Questionnaire for the Evaluation Study*

| | Average | s. d. |
|---|---|---|
| Time to learn interface (min.) | 10 | 8 |
| Amount learnt | 2.9 | 0.9 |
| Enjoyment | 2.9 | 1 |
| Ease of using interface | 2.8 | 1 |
| Usefulness of feedback | 2.8 | 1 |

**Discussion**

The results show COLLECT-UML is an effective learning environment. The participants achieved significantly higher scores on the post-test, suggesting that they acquired more knowledge in UML modelling. The learning curves also prove that students do learn constraints during problem solving. Subjective evaluation shows that most of the students felt spending more time with the system would have resulted in more learning 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 such as including short cut keys, including more details in some

of the feedback messages and tool tip boxes, providing tutorials on how to use the system and including general explanations of the full solutions, when they are being displayed to the user.

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 the system in their personal time for practicing UML modelling.

## 3.7 Summary

This chapter discussed the design and implementation of the single-user version of COLLECT-UML, an ITS developed to assist students learning UML modelling. We presented the system's architecture, functionality, and interface. COLLECT-UML's effectiveness in teaching UML class diagrams was evaluated in the two classroom experiments. The results of both subjective and objective analysis proved that COLLECT-UML is an effective educational tool. The participants performed significantly better on a post-test after short sessions with the system, and reported that the system was relatively easy to use. We then extended the system to support multiple students solving UML problems collaboratively. The extensions made to the tutor are described in the next chapter.

# Chapter 4

# COLLECT-$\mathcal{UML}$: Multi-User Version

The collaborative version of the system 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, at the same time providing feedback on collaboration.

Although many tutorials, textbooks and other resources on UML are available, we are not aware of any attempt at developing a CSCL environment for UML modelling. 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. The system 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 OO 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 which 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 determines when and why the students are having trouble learning the new concepts they share with each other. The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities. In this regard, even though the system is effective as a collaboration tool, it would probably not be an effective teaching system for a group of novices with the same level of expertise, as it could be common for a group of students to agree on the same flawed argument (For more details, refer to Section 2.3.2).

This chapter describes the extensions made to the single-user version of the system (described in the previous chapter) to make it support collaboration. Section 4.1 describes the architecture, followed by a description of the interface in Section 4.2. Section 4.3 presents the ideal collaboration model, represented as a set of meta-constraints and compares our model with previously proposed models in the literature.

## 4.1 Architecture

The collaborative teaching strategy used in COLLECT-$\mathcal{UML}$ is based on 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 Figure 4.1. COLLECT-$\mathcal{UML}$ 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 modeller that creates and maintains student models for individual users, the constraint

set, a pedagogical module, and a group modeller, 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 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).

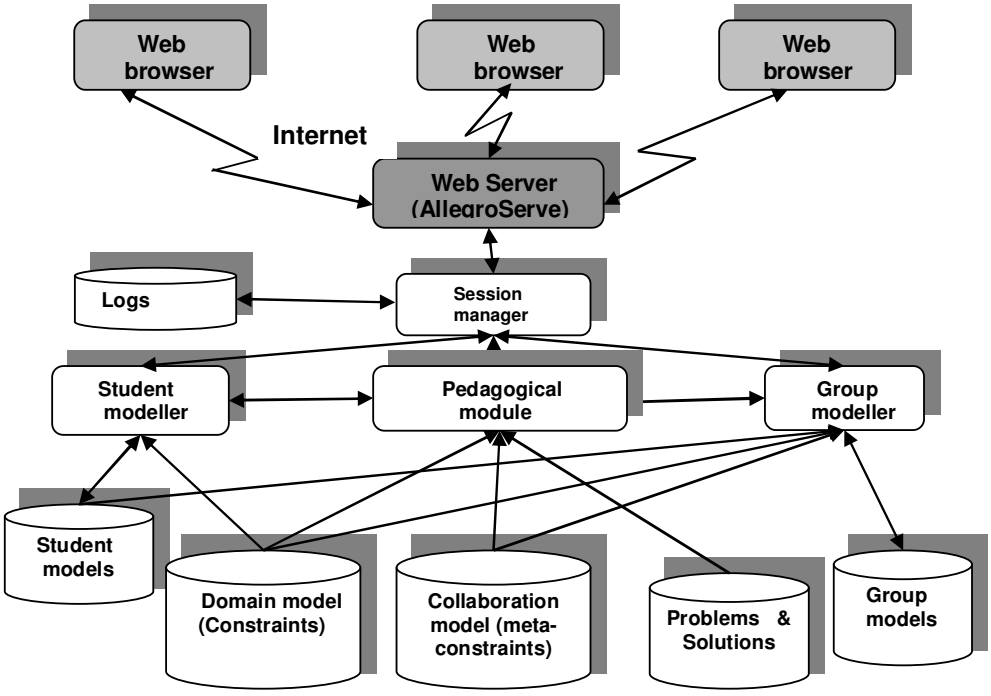



*Figure 4.1: The architecture of COLLECT-UML*

## 4.2 Interface

The student interface is shown in Figure 4.2. 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 start discussing it in the group. The group diagram is initially disabled. It will be 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 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: *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.

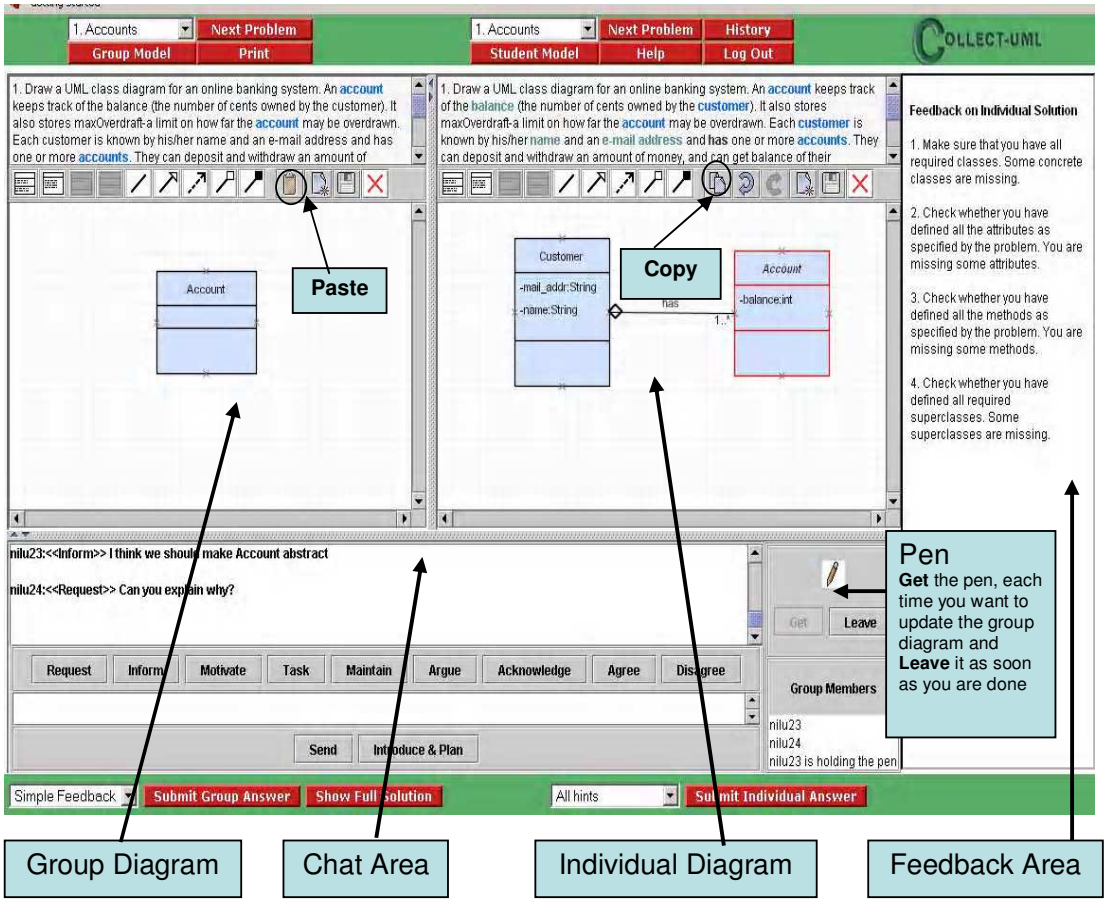



*Figure 4.2: COLLECT-UML Interface*

The collaboration-based advice is given to individual students based on the 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.

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* button can be accessed by all the members to inspect their group model (Figure 4.3). The group model visualizes the group's knowledge of the main OO

concepts being taught (i.e. classes, attributes, methods, relationships, and specialisation) in terms of skill meters, showing how much of the corresponding knowledge they have covered/learned for each concept. It can, therefore, assist students in developing their meta-cognitive skills and subsequently self-regulate their collaborative activity. 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 members.

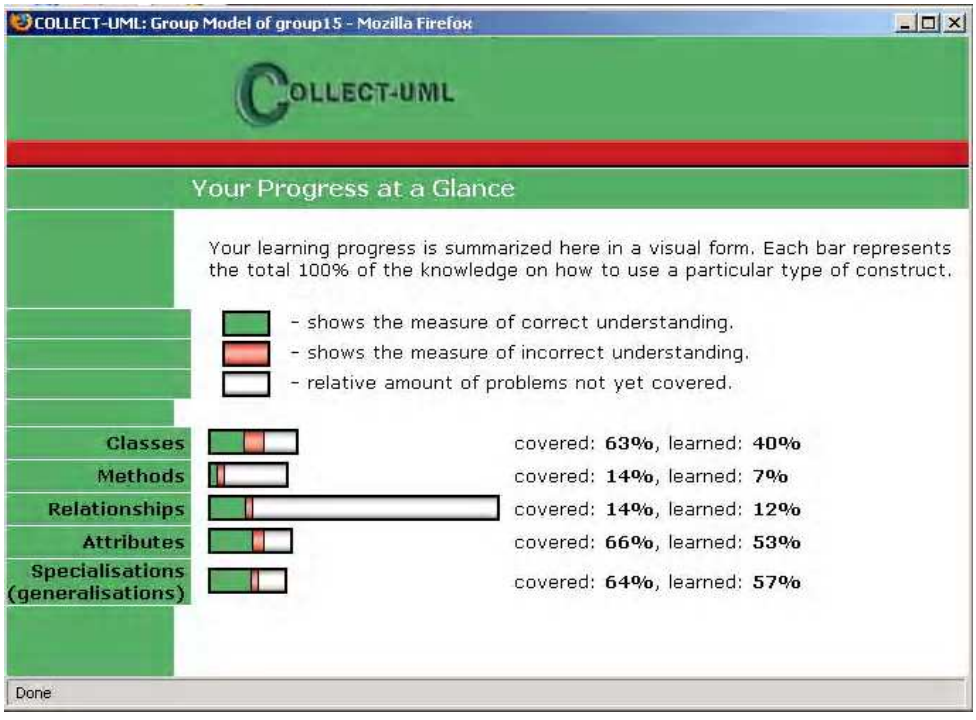



*Figure 4.3: Open Group Model*

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.

### 4.2.1 Communication categories

The use of communication categories structures the students' conversation and eliminates off-task discussions. The structured chat interface with specific communication categories can promote more focus on reflection and the fundamental concepts at stake [Baker, et al. 2001]. As noted by other researchers [Jermann, et al. 2004], the usage of structured dialogue requires extra effort from students in comparison to free-form input (such as email or chat), as students have to qualify their statements in terms of the message types (by selecting the appropriate communication category). Although this kind of interaction is slower and more demanding, it structures the data and thus makes it easier to analyze interactions between the group members.

Communication categories provide a way for students to specify the intention of their conversational contribution [McManus and Aiken, 1995]. Results from various projects indicate that the use of the structured dialogue "supports and increases learners' task-oriented behaviour, 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 communication categories, thus changing the nature of the collaborative interaction.

Some experiments [Baker and Lund, 1997] have shown that the structured section of interfaces containing both structured and free chat tools, were used more frequently than the free section. 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].

Table 4.1 lists the descriptions of the communication categories used in COLLECT-UML. They are adapted from McManus and Aiken's Collaborative Skills Network [1995], shown to be most often exhibited during collaborative learning and problem solving in Soller's studies [2001].

*Table 4.1: Communication Categories and their Descriptions*

| Communication Category | Description |
|---|---|
| Request | Ask for help in solving the problem, or in understanding a team-mates comment. |
| Inform | Direct or advance the conversation by providing information. |
| Motivate | Provide positive feedback. |
| Task | Shift the current focus of the group to a new subtask. |
| Maintain | Support group cohesion and peer involvement. |
| Argue | Reason about suggestions made by team-mates. |
| Acknowledge | Let your team-mates know that you read and/or appreciate their comments. |
| Introduce & Plan | Introduce yourself to your team-mates and plan the session in advance before start collaborating. |
| Disagree | Disagree with the comments or suggestions made by team members. |

### 4.2.2 Turn Taking

A recent study [Rummel and Spada, 2005] has integrated empirical findings from different research approaches to define relevant characteristics of a good collaboration, and the authors considered turn-taking to be 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 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 a turn-taking protocol in most computer-mediated collaboration tools is considered to be one of the limitations of such tools [Feidas, et al. 2001].

### 4.2.3 Private Workspace

Providing a well-balanced proportion of individual and joint work phases is considered crucial for successful collaboration in a recent 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.

## 4.3 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 practised 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 participants were divided into four groups: 1) Dyads who were exposed to an

elaborated worked-out collaboration example (observational learning); 2) Dyads who were given the opportunity of learning from scripted collaborative problem-solving; 3) Dyads who were involved in collaborating on a task similar to the ones they were confronted with in the application phase later (unscripted collaboration) and finally 4) Control group who did not participate in the learning phase. In the second collaboration (application phase), the groups were given the opportunity to apply and test the skills they had acquired in the learning phase. No further help or instruction was given in this phase. The results indicated that instructional support measures (both observing a worked-out collaboration example and collaborating with a script during the learning phase) improved participants' collaborative skills and increased their knowledge about characteristics of good collaboration. Regarding the comparison of the unscripted and the control condition, 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 and support in learning the social interaction skills, just as students learning in the classroom need support from their instructor [Soller, 2001].

The goal of our research is to support collaboration by modelling collaborative skills. COLLECT-$\mathcal{UML}$ 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, at al. 2003; Vizcaino, 2005; Soller, 2001; Rummel and Spada, 2005], and also used our own experience in collaborative work. One of the main contributions of this thesis is

extracting the ideas of good collaboration from the literature and turning them into meta-constraints.

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 4.2 shows different categories of meta-constraints with one or more examples for each category.

*Table 4.2: Collaboration-based Feedback Types*

| 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. |
| Initial Planning – Encouraging Advanced Planning | Would you like to introduce yourself to your team-mates 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. |

There are four different time intervals the meta-constraints are evaluated at: 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 minutes (e.g. asking students to ensure adequate elaboration is provided in their explanations), 8 minutes (e.g. encouraging students to explain to other members why they agree or disagree with a solution), and 10 minutes (e.g. encouraging students to contribute to the construction of the group diagram).

Figure 4.4 presents four meta-constraints. The relevance condition of constraint 227 focuses on methods that are defined for certain classes in the student's individual solution, 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, checking that if the student has added a class, method, attribute or relationship to the group diagram, they must have provided an 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 group solution (classes, methods, attributes or relationships), or to the chat area. If that is not the case, the feedback message asks the student to contribute to the discussion. For more examples of meta-constraints, refer to Appendix D.

(227
IF [relevance condition]
The student's solution has a class *C,*
The student's solution has a method of *C* named *m,*
The group's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the group's solution has a method of *C* named *m*
ELSE
Message: **"Some methods in your individual solution are missing in the group diagram. You may wish to share your work by adding those method(s)/discuss it with other members."**

(229
IF [relevance condition]
The student's collaboration has a class *C,*
OR the student's collaboration has a method *m,*
OR the student's collaboration has an attribute *a,*
OR the student's collaboration has a relationship *r,*
THEN [satisfaction condition]
It is necessary that the student has contributed to the chat area,
AND the student has used one of the following communication categories: *Request*, *Inform*, *Motivate, Task*, *Maintenance* or *Argue.*
ELSE
Message: "**You may wish to give explanation and provide justification each time you make a change to the shared diagram.**"

(238
IF [relevance condition]
The student has contributed to the chat area
THEN [satisfaction condition]
It is necessary that the contribution includes some text
ELSE
Message: **"Ensure adequate elaboration is provided in explanations.**"

(240
IF [relevance condition]
In each session
THEN [satisfaction condition]
It is necessary that the student's contribution includes a class
OR the student's contribution includes a method
OR the student's contribution includes an attribute
OR the student's contribution includes a relationship
OR the student's contribution includes a chat contribution
ELSE
Message: "**Would you like to contribute to the group discussion?"**

*Figure 4.4: 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 id of the student who made that change is sent to the server. Figure 4.5 represents a class location change and an attribute creation event:

```
<event type="ClassLocationChange">
    <parameters>
        <user name="nab49" group ="Group3" problem_number="8"/>
        <bound_component tag="E1" />
        <position xval="295.0" yval="98.0" />
    </parameters>
 </event>

 <event type="AttributeCreation">
    <parameters>
        <user name="nab49" group ="Group3" problem_number="8"/>
            <attribute name="name" tag="E1A1" visibility="private"
               is_static="no" attribute_type="String" Class="E1"/>
            <position xval="295.0" yval="109.0" />
            <dimension xval="106.0" yval="20.0" />
    </parameters>
 </event>
```

*Figure 4.5: Examples of class location change and attribute creation events*

Each chat message sent by group members to each other will also be sent to the server in the XML format (Figure 4.6). Each chat event consists of the student id, the type of communication category they have used and the content of the message.

```
<event type="ChatMessage">
    <parameters>
        <message name="nab49" type="Inform" text="I think name is
                 an attribute of the Customer" />
    </parameters>
 </event>


 <event type="ChatMessage">
    <parameters>
        <message name="abc12" type="Request" text="Can you explain
                 why?" />
    </parameters>
 </event>
...
```

*Figure 4.6: Examples of chat events*

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, in a format shown in Figure 4.7. The internal representation consists of 7 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. This representation is similar to the representation of ideal solution (shown in Figure 3.8), except for a new component (DESC). The meta-constraints are evaluated against these histories, and feedback is given on contributions which involve adding/deleting/updating components in the shared diagram as well as contributions made to the chat area.

```
("RELATIONSHIPS" "@ R1 association E2 E1 1..* 1 null null
                  placing " )
 ("ATTRIBUTES" "@ E1A1 name E1 String private no ")
 ("METHODS" "" )
 ("CLASSES" "@ E1 Customer concrete ")
 ("SUPERCLASSES" "" )
 ("SUBCLASSES" "" ))
 ("DESC" "@ Inform IthinknameisanattributeoftheCustomer ..."))
```

*Figure 4.7: Internal representation of students' contributions on the server*

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 (Figure 4.3).
- **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.

# Chapter 5

# Evaluation

This chapter describes the evaluation studies performed to examine the effectiveness of the multi-user version of the system in teaching UML class diagram and effective collaboration. Section 5.1 describes the pilot study followed by a full evaluation study described in Section 5.2. The overall results are discussed in Section 5.3.

## 5.1 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 8 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 modelling.

The study was carried out in the form of a think-aloud protocol. Each participant was asked to verbalise his/her thoughts while performing a UML modelling task using COLLECT-$\mathcal{UML}$ 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 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. The system included five different levels of feedback (*Simple Feedback, Error Flag, Hint, All Hints, Full Solution*). Since they spent only about 30-40 minutes familiarising themselves with the interface and 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 colour 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 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 synchronized more often.

## 5.2 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 hypothesis of the study was that the support for collaborative learning provided by meta-constraints would help students learn to collaborate. 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 learnt UML modelling concepts during two weeks of lectures and had some practice during two weeks of tutorials prior to the study.

The study was conducted in two streams of two-hour laboratory sessions over two 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 (Figure 5.1) were also handed out. The idea of providing students with such a script and therefore supporting instructional learning came from a recent 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 (Figure 4.2).

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 two individuals in each pair worked on a big, relatively complex problem (given in Appendix B.1) separately and started collaborating with each other whenever they were ready – the group diagram was activated after 10 minutes. We made sure that the members of 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.

### 5.2.1 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). The students logged into the system under their university usercodes (not their real names). Both control and experimental groups received instructions on characteristics of good collaboration at the beginning of the session (Figure 5.1).

Both versions of the system provided five levels of feedback on students' solutions (*Positive/negative, Error Flag, Hint, All Hints, Full Solution*). Tables 5.1 and 5.2 present some general statistics about the second week of the study. Active pairs were those who collaborated (i.e. contributed to the chat area, the group diagram or both). One control pair and three experimental pairs did not collaborate during the session and no contribution was recorded in their logs. Analysing their individual and shared logs show that the members of two experimental pairs worked individually, but they did not collaborate. One experimental pair did not interact with the system at all (they preferred to work on something else during the lab session) and one member of the inactive control pair worked individually and tried to start collaborating by leaving a few sentences in the chat log, but the other member did not respond. Out of twenty 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; some of them got it right on their first submission.

As can be seen from Table 5.2, the experimental group students contributed more to the group diagram, with the difference between the average number of individual contribution for control and experimental group being statistically significant ($t = 2.03$, $p = 0.03$). There was no significant difference on the other reported measures.

*Table 5.1: Some statistics about the study*

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

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 hours for the control group and 1.3 hours for the experimental group.

*Table 5.2: Some statistics about the study*

| | Control | | Experimental | |
|---|---|---|---|---|
| | Average | s.d. | Average | s.d. |
| Group submissions | 5.7 | 6 | 4.6 | 5.1 |
| Meta-constraints applied | -- | -- | 19.4 | 9 |
| **Individual contributions to the group diagram** | **11.7** | **8.6** | **18.7** | **10.6** |
| Individual contribution to the chat area | 22.2 | 15.3 | 23.9 | 11.7 |
| Individual submissions | 19.8 | 20.6 | 16.4 | 18.5 |
| Total time (hours) | 1.4 | 0.3 | 1.3 | 0.4 |

### 5.2.2 Pre- and Post-test Performance

The pre-test and post-test (given in Appendix B.2) 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 also 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 5.3. The numbers reported for the post-test do not include the collaboration question.

*Table 5.3: Mean Collaboration-, pre- and post-test scores*

| | Control | | Experimental | |
|---|---|---|---|---|
| | **Average** | **s. d.** | **Average** | **s. d.** |
| Collaboration-test | 22% | 22% | 52% | 39% |
| Pre-test | 52% | 20% | 49% | 19% |
| Post-test | 76% | 25% | 73% | 25% |
| Gain score | 17% | 28% | 21% | 31% |

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 5.3). 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 than pre-test for both control group ($t = 2.11$, $p = 0.01$) and experimental group ($t = 2.06$, $p = 0.002$). There was no significant difference between the groups on the post-test. The experimental group, 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.

We did not see a significant difference between the average post-test scores of experimental and control group. Rummel & Spada's study [2005] 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 hours in average interacting with the system and both control and experimental groups were provided with collaborative problem-solving setting and domain-level feedback – both shown to improve learning. More research is needed to investigate the effect of collaboration-based feedback on learning the domain knowledge.

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:

*(Average collaboration score* $_{exp}$ *– Average collaboration score* $_{control}$*)/ s.d.* $_{control}$ = 1.3.

*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)

*Figure 5.1: Exemplary collaboration*

### 5.2.3 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-$\mathcal{UML}$ in the second week. Figure 5.2 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 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 has 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 has decreased to 0.12 at its eleventh occasion, displaying a 47.8% decrease in probability.

We cannot make any comments on the difference of the slops of the two curves, as the students spent less than 1.4 hours on average interacting with the system. More studies are needed to investigate the effect of collaboration-based feedback (provided for the experimental group) on learning domain knowledge. The main aim of our study was to show that meta-constraints can effectively model collaborative activities.

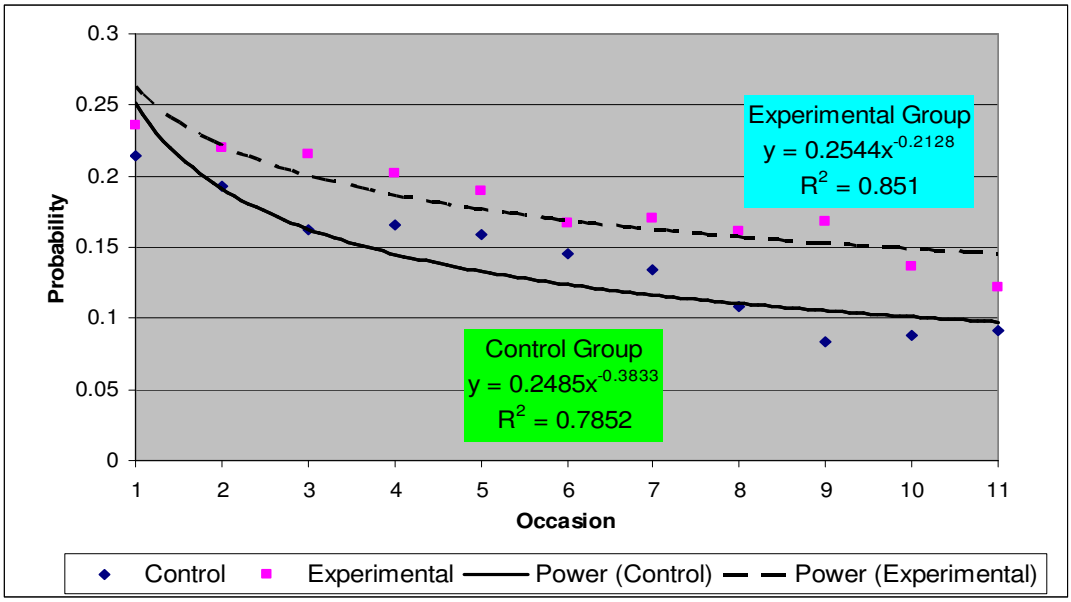



*Figure 5.2: Probability of domain constraint violation for individuals in control and experimental groups*

Figure 5.3 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.

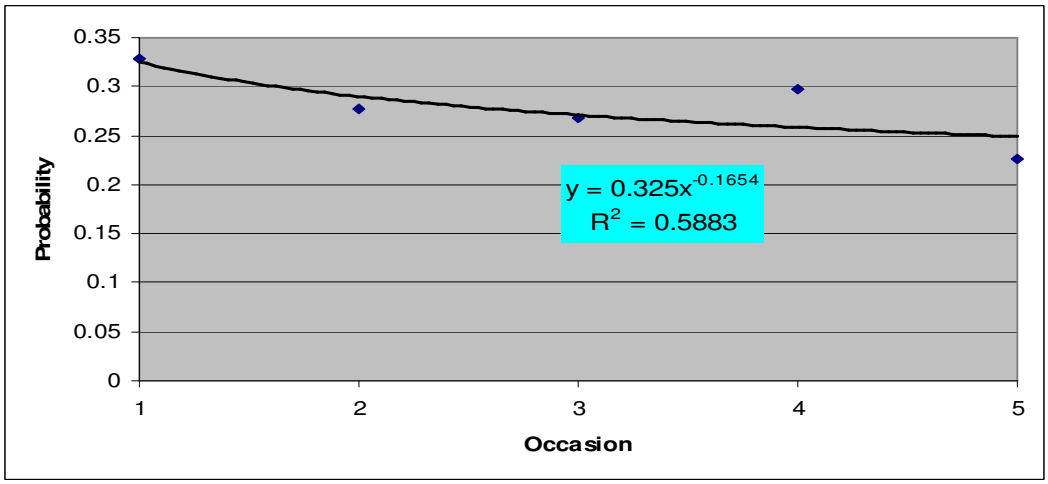



*Figure 5.3: Probability of meta-constraint violation for experimental group*

We found out that 20 domain constraints (out of 76 constraints which were relevant for the problem) were never violated by the participants, meaning that the students already knew the corresponding domain concepts. None of these constraints were enforced by the interface. They 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. 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 which 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 minutes), 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 they discuss this with other members.

### 5.2.4 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 control and experimental group's interactions (ignoring off-topic conversations) are shown in Figures 5.4 and 5.5 respectively. The experimental group was more balanced in this respect, as the students participated more in group maintenance (by using *Maintain* opener) and task management activity (*Task* opener), requesting information, arguing and disagreeing with other members compared with

control group. *Inform, Acknowledge* and *Introduce and Plan* contributions occurred more in the control group.

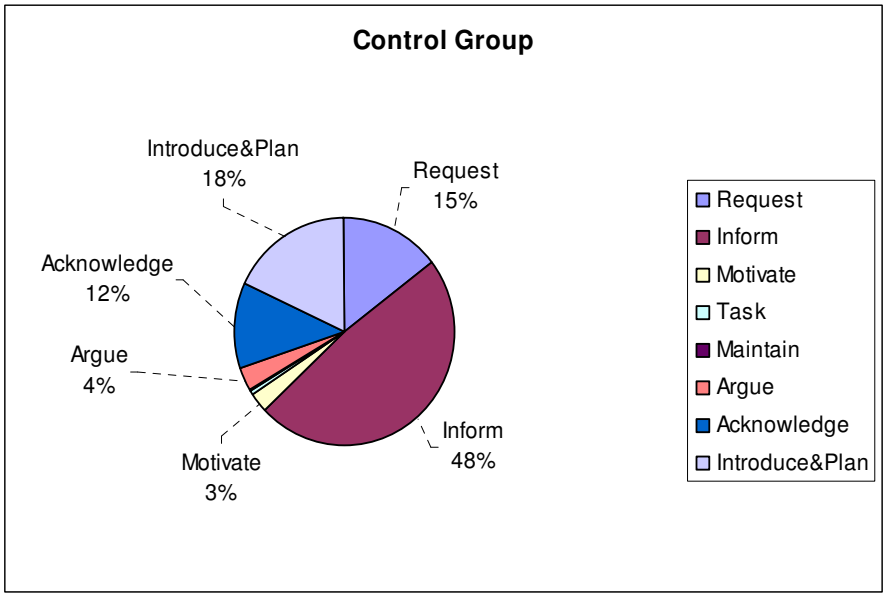



*Figure 5.4: Use of communication categories by control group*

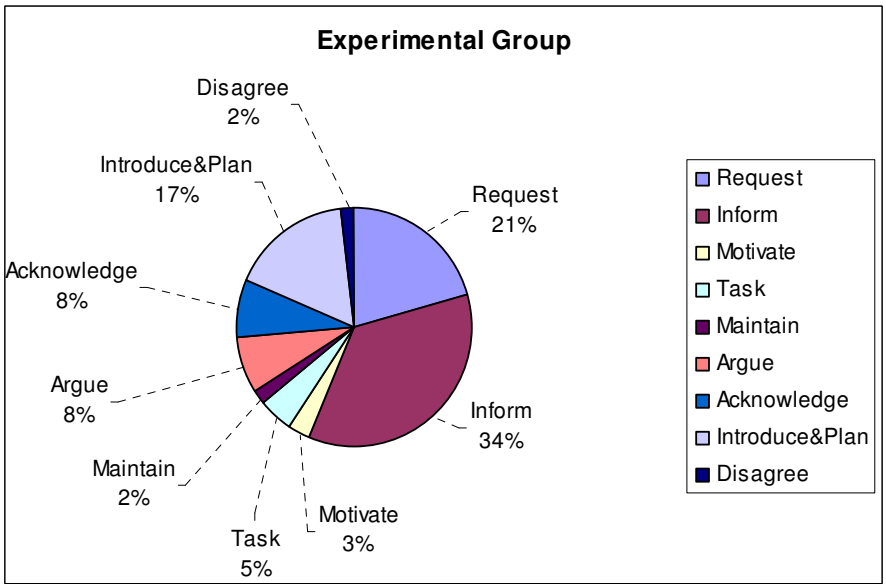



*Figure 5.5: Use of communication categories by experimental group*

### 5.2.5 Examples of Good and Bad Collaboration

We chose two pairs from the control and experimental groups to show examples of good and bad collaboration. There was no difference between the average pre-test marks of the two pairs (60% for control pair and 58% for the experimental pair), however the experimental pair did much better on the post-test (average of 85% compared to 60% scored by the control group). We call the control pair who did not collaborate effectively, Group A, and the experimental pair who did collaborate effectively Group B. Figures 5.6 and 5.7 illustrate the probability of domain constraint violation for Group A and B respectively. As it can be seen, the data points in Figure 5.7 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 Group A. The probability of 0.3 for control pair violating a constraint on the first occasion has decreased to 0.29 at its seventh occasion, which is almost the same as initial error probability.

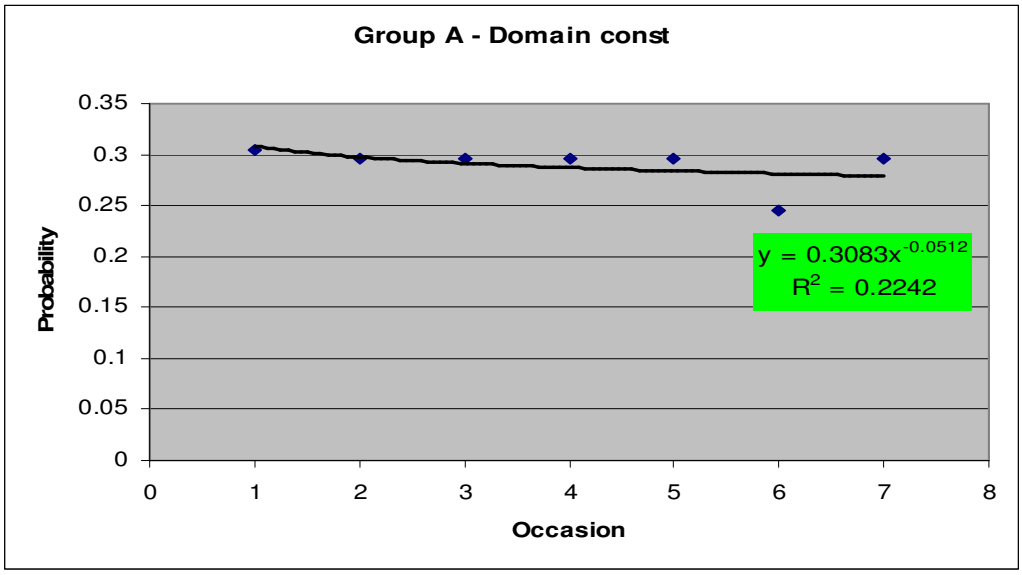



*Figure 5.6: Probability of domain constraint violation for Group A*

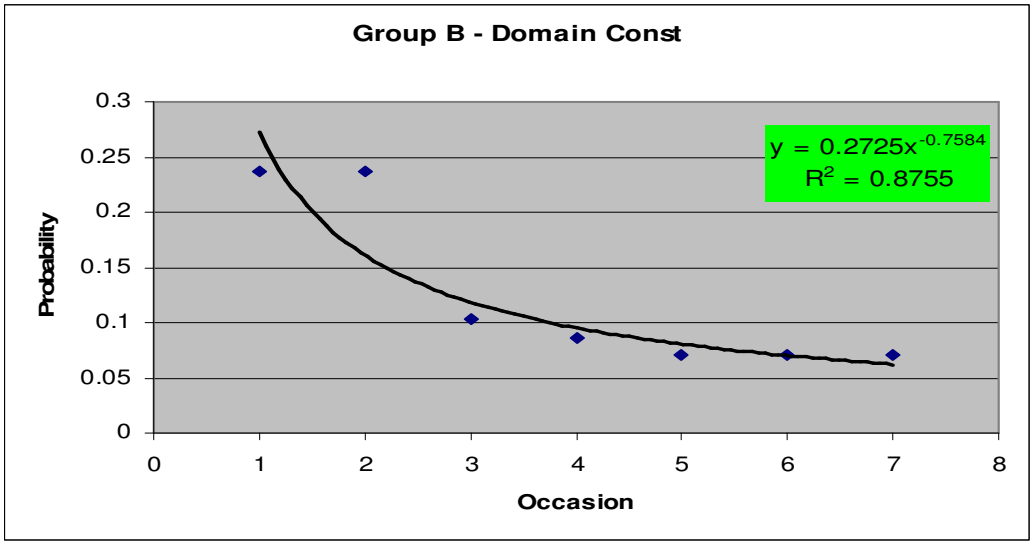



*Figure 5.7: Probability of domain constraint violation for Group B*

Figure 5.8 and 5.9 show the probability of meta-constraint violation for the two members in the experimental pair (the control pair 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 over time.

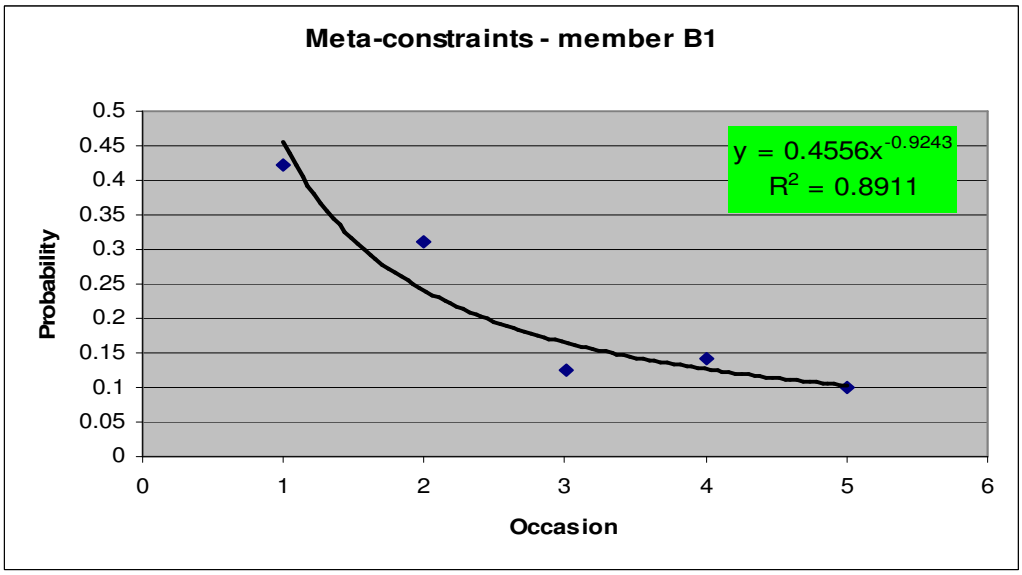



*Figure 5.8: Probability of meta- constraint violation for Member B1 of Group B*

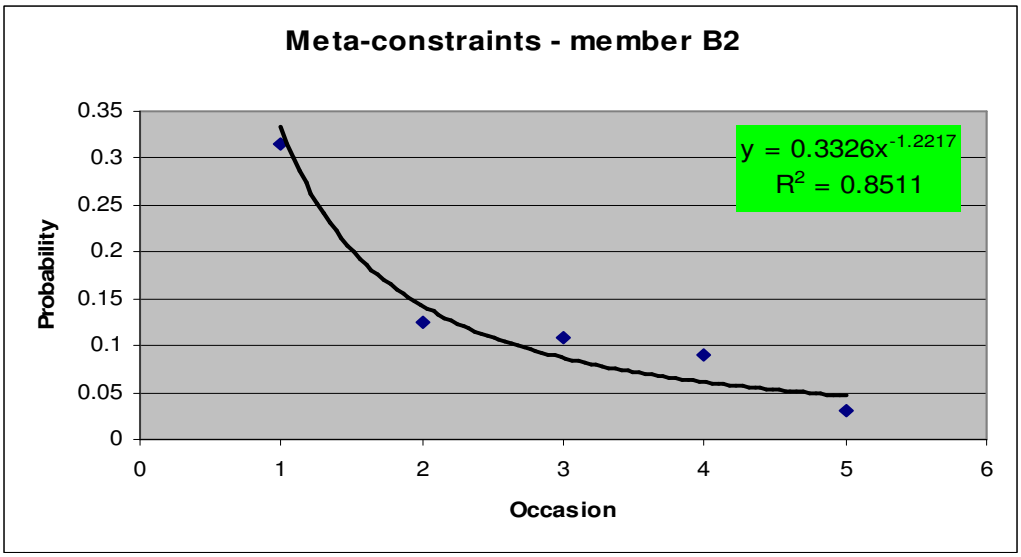



*Figure 5.9: Probability of meta- constraint violation for Member B2 of Group B*

We also looked at the use of communication categories by the two pairs. As Figures 5.10 and 5.11 show, the experimental pair was much more balanced in using different communication categories. The control pair used the *Inform* communication categories extensively and spent very little time on planning the session in advance.

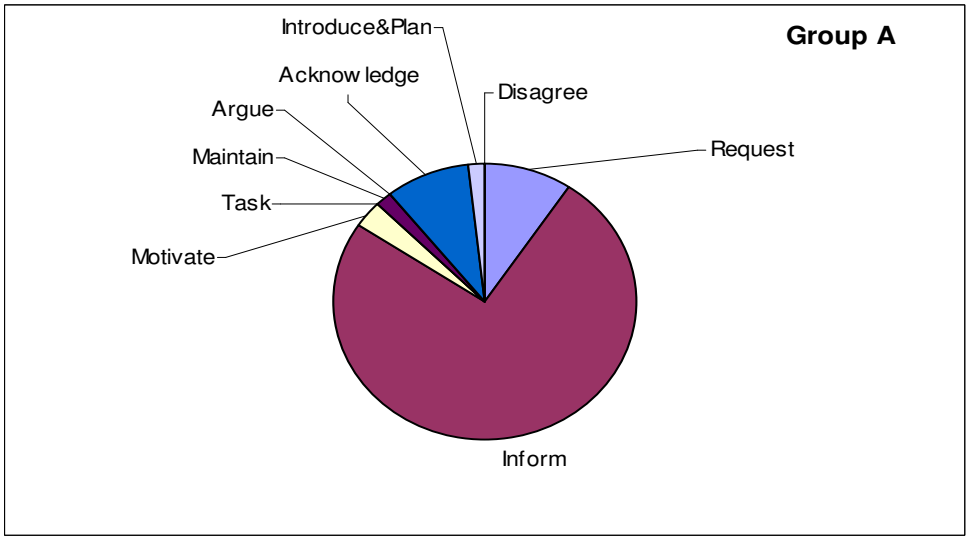



*Figure 5.10: Use of communication categories by group A*

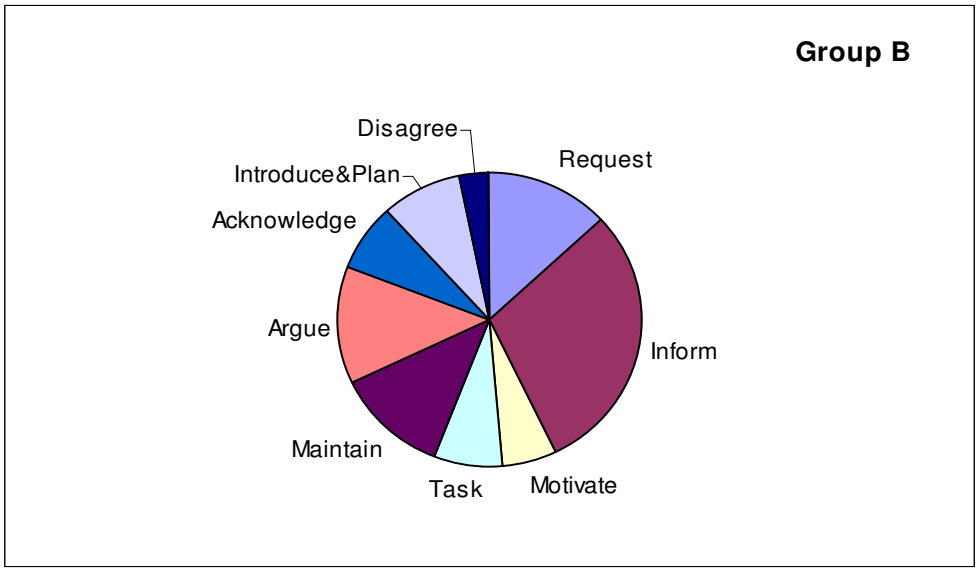



*Figure 5.11: Use of communication categories by group B*

Figure 5.12 and 5.13 show excerpts of the collaboration logs of the control and experimental pairs respectively. The diamonds show 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 Figure 5.12, one member of the control pair is more active than the other. Since they were part of the control group, they were not receiving feedback on their collaboration activities. We have highlighted the parts where getting collaboration feedback would have been useful. For example, a collaboration feedback would have been generated at 12:27 asking member A1 to provide 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 communication category. Collaboration feedback could have also encouraged member A2 to be more active and to provide justification each time they made a change to the shared diagram.

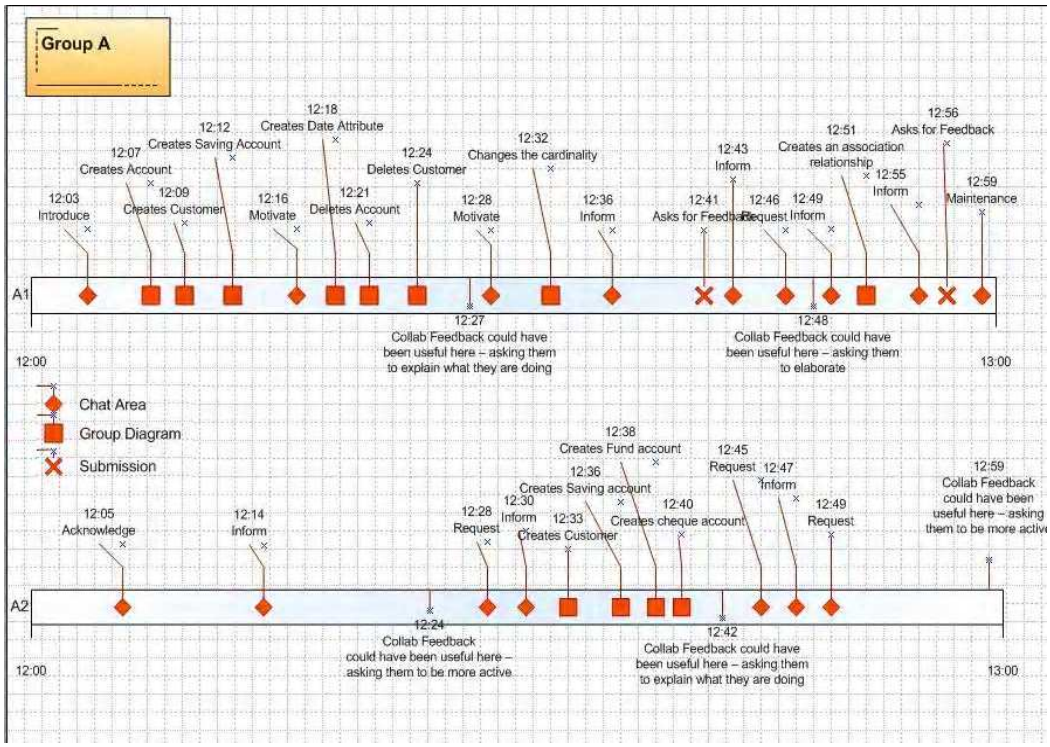



*Figure 5.12: Part of the collaboration log of Group A (Bad Control)*

We have highlighted the parts where meta-constraints were generated for the pair in the experimental group (Figure 5.13) and how they were used by the members to make their collaboration process more effective. Examples are given at 12.18 when the collaboration 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* communication category. 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* communication category) and changes the relationship to aggregation instead. The change is then justified by using an *Inform* communication category at 12:53.

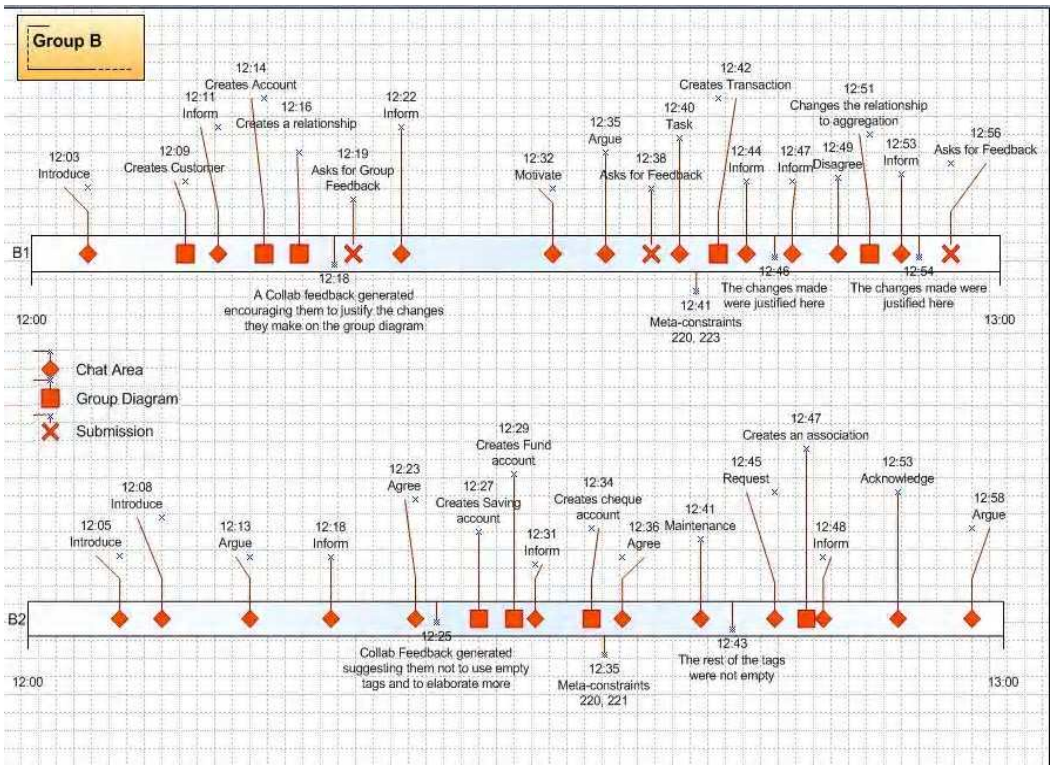



*Figure 5.13: Part of the collaboration log of Group B (Good Experimental)*

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* communication category at 12:23. He did not use an empty communication category from that point on. He 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 looked at a good control pair (group C) who collaborated effectively compared with other pairs and also did well in the UML diagram. Figure 5.14 shows the probability of domain constraint violation for group C. The data points show a regular decrease which is approximated by a power curve with a $R^2$ fit of 0.91, initial error probability (0. 13) and learning rate (-0.93), thus showing that the members learn domain constraints over time.

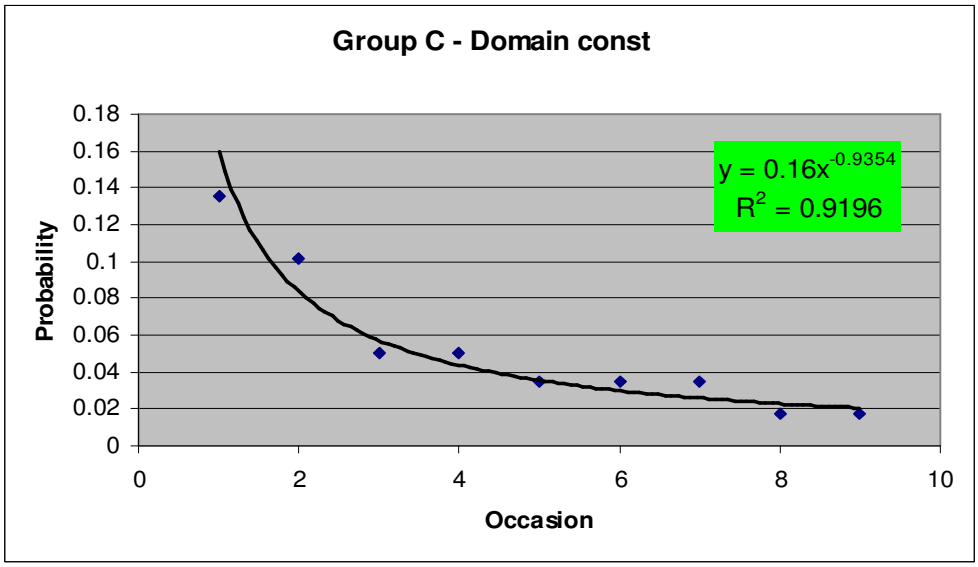



*Figure 5.14: Probability of domain constraint violation for Group C*

Figure 5.15 shows an excerpt of the collaboration log of Group 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 explanation and provide justification after making changes to the shared diagram. As shown in Figure 5.15, 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.

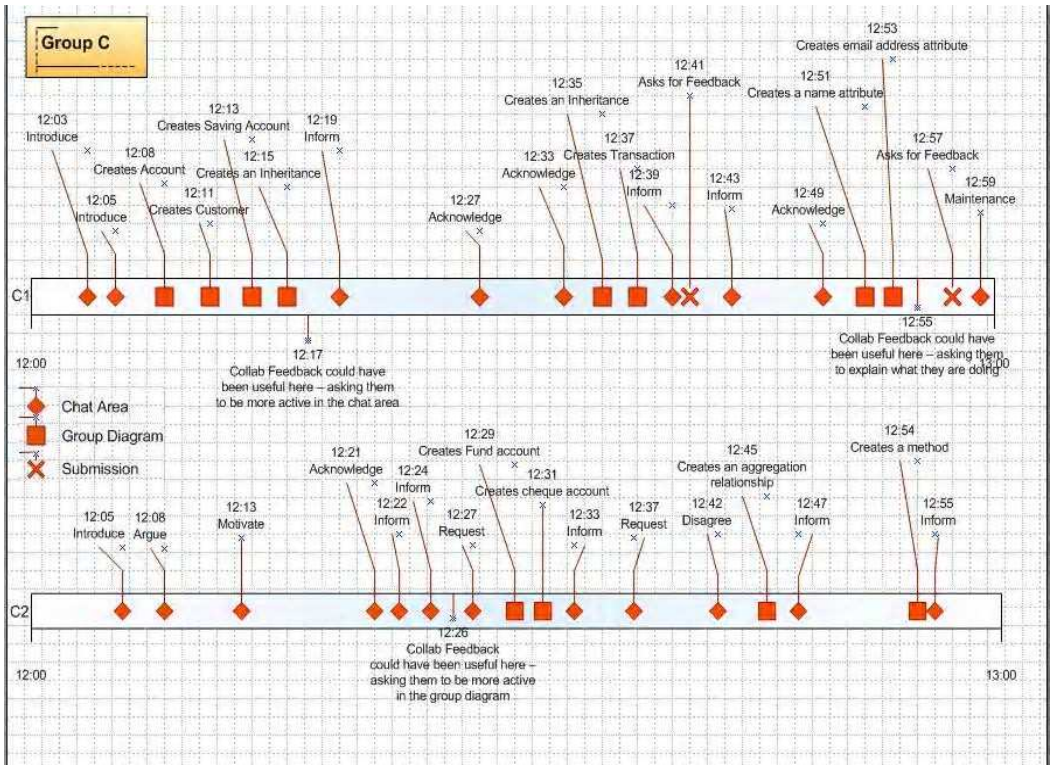



*Figure 5.15: Part of the collaboration log of Group C (Good Control)*

### 5.2.6 Subjective Analysis

The participants were given a questionnaire (Appendix B.3) at the end of the session to determine their perceptions of the system. Table 5.5 presents a summary of the responses. 73% of the control group and 41% of the experimental group were familiar with UML modelling 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 learnt by interacting with COLLECT-$\mathcal{UML}$ 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 minutes to learn the interface and become comfortable using it.

*Table 5.5: Mean responses from the user questionnaire for the evaluation study*

| | Control | | Experimental | |
|---|---|---|---|---|
| | **Average** | **s. d.** | **Average** | **s. d.** |
| 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 individualised 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. 55% 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 (Appendix B.4) show that the students liked the system and thought it improved their knowledge, and also pointed out several possible improvements.

## 5.3 Summary

The results show that COLLECT-$\mathcal{UML}$ is an effective learning environment. The participants performed significantly better on the post-test after short sessions with the system, suggesting that they acquired more knowledge in UML modelling. 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. 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-$\mathcal{UML}$ was an effective teaching tool. A number of students who participated in the study inquired about the possibility of using the system after the study, for practicing UML modelling and preparing for the exam.

# Chapter 6

# Conclusions

The thesis presented COLLECT-$\mathcal{UML}$, a new design for a true merger of ITS and intelligent CSCL systems. The system is a collaborative constraint-based ITS that teaches object-oriented analysis and design using Unified Modelling Language (UML). While teaching how to design UML class diagrams, COLLECT-$\mathcal{UML}$ is also an innovative CSCL system, providing feedback on collaboration and aiming to make the collaboration process more effective. Being a constraint-based tutor, COLLECT-$\mathcal{UML}$ represents the domain knowledge as a set of syntax and semantic constraints. However, it is the first system to also represent a higher-level skill such as collaboration using the same formalism. We started off 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.

We then extended the system to provide support for collaboration as well as domain-level support. The enhancement process included implementation of the shared workspace, modification of the pedagogical module to support groups of users, designing and implementing a group-modelling component, and developing a set of meta-constraints which are used to represent an ideal model of collaboration.

The collaborative feedback is provided by analyzing students' activities in chat area and group diagram and comparing them to an ideal model of 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. The meta-constraints are divided into two 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.), and constraints that monitor students' contributions to the chat area and the use of communication categories.

COLLECT-UML is one of the rare systems to provide both domain-level feedback (on individual and group work) and feedback on collaboration. A significant contribution of the thesis 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.

## 6.1 Results

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 experimental group students collaborated more, 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 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 modelling.
5. The students enjoyed working with the system and found it a valuable asset to their learning.

In our full evaluation study, the participants spent less than 1.4 hours in average interacting with the system. Both control and experimental groups were provided with collaborative problem-solving setting and domain-level feedback, which have been shown to improve learning in our study of the single-user version. The goal of the final study was to show that collaborative feedback improves collaborative skills only; we did not investigate the effect of collaboration on the quality of the domain knowledge learned. 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 modelling and supporting collaboration in computer-supported collaborative learning environments.

## 6.2 Future Directions

The system can be enhanced in future to supporting the following features:

- The collaboration model can be enhanced by assigning the students roles such as questioner, clarifier, mediator, informer, and facilitator (among others), and possibly rotate these roles around the group for each consecutive dialogue segment.
- When a student requests help, the system can encourage his/her team-mates to respond in a timely manner.
- Colours and emoticons can be included in the chat area to make it more realistic and interesting for students to use.

- Displaying the word *chatting* at the status bar of the window, while the user is typing in the text box to let the other team members know that he/she will soon be contributing to the conversation.
- The system can provide the students with collaborative learning skill usage statistics (e.g. 10% Inform, 50% Request, and 40% Argue).
- There are situations where a group member would want to know the status of progress of other members. In that case, the system can diagnose other members' collaborative/domain knowledge and present concise and meaningful feedback, hence supporting open student model for other members.
- The evaluation studies performed using multi-user version involved pairs of students. There has been some research showing that groups of three to five collaborators can make the learning process more effective. With the current implementation of the system, there is no limit on the number of group members. Further studies can be conducted with larger groups to evaluate the effect of group size on student's learning.
- An evaluation study can be conducted to investigate the effect of collaboration-based feedback on learning the domain knowledge. In this kind of study, the student would not receive domain-level feedback. The experimental group would receive feedback on their collaborative activities, while the control group would not receive any feedback from the system. This kind of experimental design would identify the effect of collaboration on learning.
- The system can be extended to provide tutorials on how to use the features.
- General explanations of the full solutions can be provided, when the solutions are being displayed to the user.

# References


[Ainsworth, Bibby and Wood, 1999] Ainsworth, S.E., Bibby, P.A. and Wood, D.J. *Analysing the costs and benefits of mutli-representational learning environments*. In M.W. van Someren, Reimann, P., Boshuizen, H.P.A., and T. de Jon (Eds.), Learning with multiple representations, Oxford: Elsevier Science, pp.120-134.

[AllegroServe, 2005] AllegroServe - a Web Application Server. Retrieved 31.5.2005 from http://www.franz.com/

[Anderson, 1985] Anderson, J.R., Jeffries, R. *Novice LISP Errors: Undetected Losses of Information from Working Memory*. Human Computer Interaction, 22. pp.403-423.

[Anderson, et al. 1995] Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. *Cognitive Tutors: Lessons Learned.* The Journal of the Learning Sciences, 4(2). pp.167-207.

[Anderson, et al. 1996] Anderson, J.R., Corbett, A., Koedinger, K. and Pelletier, R. *Cognitive Tutors: Lessons Learned*. Journal of Learning Sciences, 4 (2). pp.167-207.

[Ayala and Yano, 1998] Ayala, G, and Yano, Y. *A collaborative learning environment based on intelligent agents.* Expert Systems with Applications, 14. pp.129-137.

[Baghaei and Mitrovic, 2005] Baghaei, N. and Mitrovic, A. *COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based tutor.* In Khosla, R., Howlett, R. and Jain L. (Eds.), (KES 2005), pp.458-464.

[Baghaei, Mitrovic and Irwin, 2005] Baghaei, N., Mitrovic, A. and Irwin, W. *A Constraint-Based Tutor for Learning Object-Oriented Analysis and Design using UML.* In Looi, C., Jonassen, D. and Ikeda M. (Eds.), (ICCE 2005), pp.11-18.

[Baghaei, 2006] Baghaei, N. *A Collaborative Constraint-based Adaptive System for Learning Object-Oriented Analysis and Design using UML.* 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2006), (Dublin, Ireland, 2006), pp.398-403.

[Baghaei and Mitrovic, 2006] Baghaei, N. and Mitrovic, A. *A Constraint-based Collaborative Environment for Learning UML Class Diagrams.* 8th International Conference on Intelligent Tutoring Systems (ITS 2006), (Jhongli, Taiwan, 2006), pp.176-186.

[Baghaei, Mitrovic and Irwin, 2006] Baghaei, N., Mitrovic, A. and Irwin, W. *Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams*, Technology, Instruction, Cognition and Learning Journal, 4(2), 133-137..

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

[Baker, et al. 1997] Baker, M.,J. and Lund, K. *Flexibly structuring the interaction in a CSCL environment*, COAST research team, UMR-GRIC, CNRS/University of Lyon, France, 1997.

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

[Baker, et al. 2001] Baker, M., de Vries, E., Lund, K. and Quignard, M. *Computer-mediated epistemic interactions for co-constructing scientific notions: Lessons learned form a five-year research program.* In Dillenbourg, P., Eurelings, A. and Hakkarainnen, K. (Eds.), European Perspectives on CSCL (CSCL 2001), pp.89-96.

[Baker, et al. 2003] Baker, M., Quignard, M., Lund, K. and Sejourne, A. *Computer-supported collaborative learning in the space of debate*. In Hoppe U. (Eds.), Computer Support for Collaborative Learning: Designing for Change in Networked Learning Environments, CSCL 2003, (Bergen, Norway, 2003), pp.(in press).

[Blank, et al. 2005] Blank, G., Parvez, S., Wei, F. and Moritz S. *A Web-based ITS for OO Design*. In Looi, C-K, McCalla, G., Bredeweg, B. and Breuker, J. (Eds.), 12th International Conference on Artificial Intelligence in Education (AIED 2005), Available at: http://www.cse.lehigh.edu/~cimel/papers/AIEDworkshop-poster.pdf

[Blasco, et al. 1999] Blasco, M., Barrio, J., Dimitriadis, Y., Osuna, C., González, O., Verdú, M., and Terán, D *From co-operative learning to the virtual class. An experience in composition techniques*. ultiBASE journal. Available at: http://ultibase.rmit.edu.au/Articles/dec99/blasco1.htm

[Barros and Verdejo, 2000] Barros, B. and Verdejo, M.F. *Analysing student interaction processes in order to improve collaboration: The DEGREE approach*. International Journal of Artificial Intelligence in Education, 11. pp.221-241.

[Beck, Stern and Haugsjaa, 1996] Beck, J., Stern, M. and Haugsjaa, E. *Applications of AI in Education.* The ACM's First Electronic Publication, http://www.acm.org/crossroads/xrds3-1/aied.html

[Blaye, et al. 1990] Blaye, A., Light, P., Joiner, R. and Sheldon, S. *Collaboration as a facilitator of planning and problem solving on a computer-based task*. CITE Report 90, Institute of Educational Technology, Open University, Milton Keynes, U.K, 1990.

[Bloom, 1984] Bloom, B.S. The *2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring*. Educational Researcher, 13. pp.3-16.

[Bolloju and Leung, 2006] Bolloju, N. and Leung, F. S. *Assisting novice analysts in developing quality conceptual models with UML*. Communications of the ACM, pp.108-112.

[Booch, et al. 1999] Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modelling Language User Guide*. Reading: Addison-Wesley.

[Brusilovsky and Peylo, 2003] Brusilovsky, P. and Peylo, C. *Adaptive and Intelligent Web-based Educational Systems.* International Journal of Artificial Intelligence in Education, 13. pp.159-172.

[Buiu, 1999] Buiu, C. Artificial intelligence in education – state of the art and perspectives. (ZIFF Papiere 111). Institute for Research into Distance Education, (ERIC Document Reproduction Service No. ED 434903), Fern University, Germany, 1999.

[Burton, 1998] Burton, M. *Computer modelling of dialogue roles in collaborative learning activities*. PhD thesis, Computer Based Learning Unit, The University of Leeds, UK, 1998.

[Carr and Goldstein, 1977] Carr, B. and Goldstein, I. *Overlays: a Theory of Modeling for Computer-aided Instruction*. Technical Report, AI Lab Memo 406, MIT, USA, 1997.

[Chan, 1991] Chan, T. *Integration-kid: A learning companion system*. In Mylopoulos, J. and Reiter., R. (Eds.), 12th International Conference on Artificial Intelligence, (Sydney, Australia, 1991), pp.1094-1099.

[Constantino-Gonzales and Suthers, 2000] Constantino-Gonzalez, M., and Suthers, D. *A coached collaborative learning environment for Entity-Relationship modeling*. 5th International Conference on Intelligent Tutoring Systems (ITS 2000), (Montreal, Canada, 2000), pp.324-333.

[Constantino-González and Suthers, 2002] Constantino-Gonzalez, M., and Suthers, D. *Coaching Collaboration in a Computer-Mediated Learning Environment.* Computer Support for Collaborative Learning (CSCL 2002), (NJ, USA, 2002), pp.583-584.

[Crook, 1994] Crook, C. *Computers and the Collaborative Experience of Learning*. Routledge London, UK, 1994.

[Dillenbourg, 1999] Dillenbourg, P. *What do you mean by "Collaborative Learning"*. In Dillenbourg, P. (Eds.), Collaborative Learning: Cognitive and Computational Approaches, Amsterdam: Elsevier Science. pp.1-19.

[Dillenbourg, 2002] Dillenbourg, P. *Over-scripting CSCL: The risks of blending collaborative learningwith instructional design*. In Kirschner P. A. (Eds.), Three worlds of CSCL, Can we support CSCL . Heerlen, Open Universiteit Nederland, pp.61-91.

[Dillenbourg, 2003] Dillenbourg, P. *Over-scripting CSCL: The risks of blending collaborative learning with instructional design.* In Kirschner P. A. (Eds.), Three worlds of CSCL. Can we support CSCL, pp.61-91.

[Dimitracopoulou and Petrou, 2004] Dimitracopoulou, A. and Petrou, A. *Advanced Collaborative Distance Learning Systems for young students: Design issues and current trends on new cognitive and meta-cognitive tools*. THEMES in Education International Journal. (in press).

[Dimitracopoulou, 2005] Dimitracopoulou, A. *Designing Collaborative Learning Systems: Current Trends & Future Research Agenda.* In Koschmann, T., Suthers, D. and Chan T.W. (Eds.), 6th International Conference on Computer Supported Collaborative Learning (CSCL 2005), pp.115-124.

[Doise and Mugny, 1984] Doise, W. and Mugny G. *The Social Development of the Intellect.* Oxford: Pergamon.

[Ericsson and Simon, 1984] Ericsson, K. A. and Simon, H. A. *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: The MIT Press.

[Feidas, et al. 2001] Feidas, C., Komis, V. and Avouris, N. *Design of collaboration-support tools for group problem solving*. In Avouris, N. and Fakotakis, N. (Eds.), Advances in Human-Computer Interaction, pp.263-268.

[Feidas, et al. 2002] Dimitracopoulou, A., Avouris, N., Komis, B. and Feidas, C. *Towards open object-oriented models of collaborative problem solving interaction.* In Jermann, P., Mühlenbrock, M. and Soller, A. Workshop Proceedings (Eds.), 4th International Conference on Computer Supported Collaborative Learning (CSCL 2002), (Boulder, Colorado, 2002), pp.52-28.

[Forgy, 1982] Forgy, C.L. *Rete: a fast algorithm for the many pattern/many object pattern match problem.* Artificial Intelligence, 19. pp.17-37.

[Garito, 1991] Garito, M.A. *Artificial intelligence in education: evolution of the teaching learning relationship.* British Journal of Educational Technology, 22(1). pp.41-47.

[Gogoulou, et al. 2005] Gogoulou, A., Gouli, E., Grigoriadou, M. and Samarakou, M. *ACT: A Web-based Adaptive Communication Tool.* In Koschmann, T., Suthers, D. and Chan T.W. (Eds.), 6th International Conference on Computer Supported Collaborative Learning (CSCL 2005), pp.180-189.

[Graesser, et al. 1999] Graesser, A. C. , Wiemer-Hastings, K., Wiemer-Hastings, P. , Kreuz, R. and the Tutoring Research Group. *AutoTutor: A simulation of a human tutor.* Journal of Cognitive Systems Research , 1. pp.35-51.

[Graesser, et al. 2003] Graesser, A.C., Jackson, G.T., Mathews, E.C., Mitchell, H.H., Olney, A., Ventura, M., Chipman, P., Franceschetti, D., Hu, X., Louwerse, M.M., Person, N.K., and the Tutoring Research Group. *Why/AutoTutor: A test of learning gains from a physics tutor with natural language dialog.* In Alterman R. and Hirsh D. (Eds.), 25th Annual Conference of the Cognitive Science Society. Boston, MA: Cognitive Science Society.

[Heller, Keith and Anderson, 1992] Heller, P., Keith, R., and Anderson, S. *Teaching problem solving through cooperative grouping. Part 1: Group versus individual problem solving.* American Journal of Physics, 60(7). pp.627-636.

[Hermann, et al. 2001] Hermann, F., Rummel, N. and Spada, H. *Solving the case together: The challenge of net-based interdisciplinary collaboration.* In Dillenbourg, P., Eurelings, A. and Hakkarainnen, K. (Eds.), 1st European Conference on Computer-Supported Collaborative Learning (CSCL 2001), pp.293-300.

[Inaba and Mizoguchi, 2004] Inaba, A. and Mizoguchi, R. *Learners' Roles and Predictable Educational Benefits in Collaborative Learning; An Ontological Approach to Support Design and Analysis of CSCL.* 7th International Conference on Intelligent Tutoring Systems (ITS 2004), (Maceio, Brazil, 2004), pp.285–294.

[Jarboe, 1996] Jarboe, S. *Procedures for enhancing group decision making*. In Hirokawa, B. and Poole, M. (Eds.), Communication and Group Decision Making, pp.345-383.

[Jermann, Soller and Lesgold, 2004] Jermann, P., Soller, A. and Lesgold, A. *Computer software support for CSCL*. In Dillenbourg P. (Series Eds.) 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, Kluwer Academic Publishers, Boston, MA. pp.141-166.

[Johnson and Johnson, 1991] Johnson D.W. and Johnson, R.T. *Learning together and alone*. Englewood Cliffs, Prentice Hall, New Jersey.

[Johnson, et al. 1998] Johnson, W. L., Rickel, J., Stiles, R., and Munro, A. *Integrating pedagogical agents into virtual environments.* Presence: Teleoperators and Virtual Environments, 7(6). pp.523-546.

[Johnson, et al. 1998b] Johnson, D. W., Johnson, R. T. and Smith, K. A. *Active Learning: Cooperation in the college classroom*. Edina, Minnesota: Interaction Book Company, 2nd Ed.

[Joiner, 1995] Joiner, R. *The negotiation of dialogue focus: An investigation of dialogue processes in joint planning in a computer based task*. In Malley, C. O' (Eds.), Computer Supported Collaborative Learning, Berlin, pp.203-222.

[Katz, et al. 1998] Katz, S., Lesgold, A., Hughes, E., Peters, D., Eggan, G., Gordin, M. and Greenberg., L. *Sherlock 2: An intelligent tutoring system built upon the LRDC Tutor Framework*. In Bloom C.P. and Loftin R.B. (Eds.), Facilitating the Development and Use of Interactive Learning Environments, pp.227-258.

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

[Kumar, 1996] Kumar, V. *Computer-Supported Collaborative Learning: Issues for Research.* Graduate Symposium, University of Saskatchewan, Canada, 1996. http://www.cs.usask.ca/grads/vsk719/academic/890/project2/project2.html

[Le and Menzel, 2005] Le, N. T. and Menzel, W. *Constraint-based Error Diagnosis in Logic Programming.* In Looi, C., Jonassen, D. and Ikeda M. (Eds.), (ICCE 2005).

[Lindland, et al. 1994] Lindland, O. I., G. Sindre, and Sølvberg, A. *Understanding quality in conceptual modelling*. IEEE Software, 11(2). pp.42-49.

[Martin, 2002] Martin, B. *Intelligent Tutoring Systems: The Practical Implementation of Constraint-based Modelling*. PhD thesis, Department of Computer Science, University of Canterbury, Christchurch, NZ, 2002.

[Martin and Mitrovic, 2002] Martin, B. and Mitrovic, A. *Authoring Web-Based Tutoring Systems with WETAS.* In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, C-H Lee (Eds.), International Conference on Computers in Education (ICCE 2002), pp.183-187.

[Mayo, et al. 2000] Mayo, M., Mitrovic, A. and McKenzie, J. *CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation*. Advanced Learning Technology: Design and Development Issues, (Los Alamitos, CA, 2000), IEEE Computer Society, pp.151-154.

[McManus and Aiken, 1995] McManus, M. and Aiken, R. *Monitoring computer-based problem solving*. International Journal of Artificial Intelligence in Education, 6(4). pp.307-336.

[McTaggart, 2001] McTaggart, J. *Intelligent Tutoring Systems and Education for the Future*. CI 512X Literature Review, Department of Mathematics and Computer Science, Drake University, Des Moines , Iowa, USA, 2001.

[Milik, Marshall and Mitrovic, 2006] Milik, N., Marshall, M. and Mitrovic, A. *Teaching Logical Database Design in ERM-Tutor.* In Ikeda, M., Ashley, K. and T.-W. Chan (Eds.), International Tutoring Systems (ITS 2006), pp.707-709.

[Mitrovic, 2003] Mitrovic, A. *An intelligent SQl tutor on the Web* International Journal of Artificial Intelligence in Education, 13 (2-4). pp.173-197.

[Mitrovic and Ohlsson, 1999] Mitrovic, A. and Ohlsson, S. *Evaluation of a Constraint-based Tutor for a Database Language*. International Journal on AIED, 10 (3-4). pp.238-256.

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

[Mitrovic, 2002] Mitrovic, A. *NORMIT, a Web-enabled tutor for database normalization*. ICCE 2002, (Auckland, New Zealand, 2002), pp.1276-1280.

[Mitrovic, et al. 2001] Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B. *Constraint-based Tutors: a Success Story*. 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2001), (Budapest, 2001), Springer-Verlag Berlin Heidelberg LNAI 2070, pp.931-940.

[Mitrovic, et al. 2003] Mitrovic, A., Koedinger, K. and Martin, B. *A comparative analysis of cognitive tutoring and constraint-based modeling*. 9th International conference on User Modelling, (Pittsburgh, USA, 2003), pp.313-322.

[Mitrovic, et al. 2004] Mitrovic, A., Suraweera, P., Martin, B. and Weerasinghe, A. *DB-suite: Experiences with Three Intelligent, Web-based Database Tutors.* Journal of Interactive Learning Research, 15(4). pp.409-432.

[Mitrovic, 2005] Mitrovic, A. *The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor*. In Looi, C-K, McCalla, G., Bredeweg, B. and Breuker, J. (Eds.), 12th International Conference on Artificial Intelligence in Education (AIED 2005), pp.499-506.

[Miyake, 1986] Miyake, N. *Constructive interaction and the iterative process of understanding*. Cognitive Science, 10. pp.151-177.

[Muehlenbrock, 2000] Muehlenbrock, M. *Action-based collaboration analysis for group learning*. Ph.D. thesis, Department of Mathematics/Computer Science, University of Duisburg, Germany, 2000.

[Muhlenbrock and Hoppe, 1999] Muhlenbrock, M. and Hoppe, U. *Computer-supported interaction analysis of group problem solving*. 3rd International Conference on Computer Support for Collaborative Learning (CSCL 1999), (California, USA, 1999), pp. 398-405.

[Nicholas, 2006] Nicholas, A. *Dealing with Ambiguity in a Foreign Language ITS.* Honours Report, Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, NZ, 2006.

[Nielsen, 1993] Nielsen, J. *Usability Engineering*. San Diego, CA: Academic Press Inc.

[Ohlsson, 1994] Ohlsson, S. *Constraint-based Student Modelling*. Student Modelling: the Key to Individualized Knowledge-based Instruction, (Berlin, 1994), Springer-Verlag, pp.167-189.

[Ohlsson, 1996] Ohlsson, S. *Learning from Performance Errors*. Psychological Review, 103 (2). pp.241-262.

[Orey and Nelson, 1993] Orey, M.A. and Nelson, W.A. *Development principles for intelligent tutoring systems: integrating cognitive theory into the development of computer-based instruction*. Educational Technology Research and Development, 41(1). pp.59-72.

[Panitz, 1999] Panitz, T. *The Case For Student Centered Instruction Via Collaborative Learning Paradigms*. 1999. Available at: http://home.capecod.net/~tpanitz/tedsarticles/coopbenefits.htm

[Plaisant, et al. 1999] Plaisant, C., Rose, A., Rubloff, G., Salter, R. and Shneiderman, B. *The design of history mechanisms and their use in collaborative educational simulations*. 3rd International Conference on Computer Support for Collaborative Learning (CSCL 1999), (California, USA, 1999), pp.348-359.

[Reimann, 2003] Reimann, P. *How to support groups in learning: More than problem solving*. In Aleven, V. (Eds.), Artificial Intelligence in Education (AIED 2003), (Sydney, Australia, 2003), pp.3-16.

[Resta, 2004] Resta. P. *Benefits of CSCL*. The University of Texas at Austin. 2004. http://www.edb.utexas.edu/resta/syllabus/cscl2003/m0_1.html

[Rosatelli, Self and Thirty, 2000] Rosatelli, M., Self, J., and Thirty, M. *LeCs: A collaborative case study system*. 5th International Conference on Intelligent Tutoring Systems (ITS 2000), (Montreal, Canada, 2000), pp.242-251.

[Roschelle and Teasley, 1995] Roschelle, J. and Teasley, S.D. *The construction of shared knowledge in collaborative problem solving*. In O'Malley, C.E. (Eds), Computer-Supported Collaborative Learning. Berlin, pp. 69-197.

[Rummel and Spada, 2005] Rummel, N. and Spada, H. *Learning to collaborate: An instructional approach to promoting collaborative problem-solving in*

*computer-mediated settings*. Journal of the Learning Sciences, 14(2). pp.201-241.

[Self, 1990] Self, J. A. *Bypassing the intractable problem of student modeling*. In Frasson, C. and Gauthier, G. (Eds.), Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education. Norwood, Ablex, pp.107-123.

[Self, 1994] Self, J. *Formal Approaches to Student Modeling*. In Greer, J.E. and McCalla, G.I. (Eds.), Student Modeling: the Key to Individualized Knowledge-based Instruction. Springer-Verlag, NATO ASI Series, pp.295-352.

[Soller and Lesgold, 2000] Soller, A. and Lesgold, A. *Knowledge acquisition for adaptive collaborative learning environments*. AAAI Fall Symposium: Learning How to Do Things, Cape Cod, MA.

[Soller, 2001] Soller, A.L. *Supporting Social Interaction in an Intelligent Collaborative Learning System*. International Journal of Artificial Intelligence in Education, 12(1). pp.40-62.

[Soller, Martínez-Monés, Jerman and Muehlenbrock, 2005] Soller, A., Martínez-Monés, A., Jermann, P. and Muehlenbrock, M. From Mirroring to Guiding: a Review of State of the Art Technology for Supporting Collaborative Learning. International Journal of Artificial Intelligence in Education, 15(4), pp.261-290.

[Sommerville, 2004] Sommerville, I. *Software Engineering*. Pearson/Addison-Wesley, 7th ed.

[Suebnukarn and Haddawy, 2004] Suebnukarn, S. and Haddawy, P. *A Collaborative Intelligent Tutoring System for Medical Problem-Based Learning*. International Conference on Intelligent User Interfaces, (Madeira, Portugal, 2004).

[Suraweera and Mitrovic, 2001] Suraweera, P. and Mitrovic, A. *Designing an Intelligent Tutoring System for Database Modelling*. 9th International Conference on Human-Computer Interaction (HCII 2001), (New Orleans, LA, 2001), pp.745-749.

[Suraweera and Mitrovic, 2002] Suraweera, P. and Mitrovic, A. *KERMIT: a Constraint-based Tutor for Database Modeling*. In Cerri, S., Gouarderes, G.

and Paraguacu, F. (Eds.), 6th International Conference on Intelligent Tutoring Systems (ITS 2002), pp.377-387.

[Suraweera, 2003] Suraweera, P. *Automatic Acquisition of Knowledge for Intelligent Tutoring Systems*. PhD thesis proposal, Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, NZ, 2003.

[Suraweera and Mitrovic, 2004] Suraweera, P. and Mitrovic, A. *An Intelligent Tutoring System for Entity Relationship Modelling.* International Journal of Artificial Intelligent in Education, 14(3-4). pp.375-417.

[Suthers and Jones, 1997] Suthers, D. and Jones, D. *An Architecture for Intelligent Collaborative Educational Systems.* In de Boulay, B. and Mizoguchi, R. (Eds.), (AIED 1997), pp.55-62.

[Suthers, et al. 2003] Suthers, D., Girardeau, L. and Hundhausen, C. *Deistic roles of external representations in face-to-face and online collaboration*. In Hoppe U. (Eds), Computer Support for Collaborative Learning: Designing for Change in Networked Learning Environments, (CSCL 2003), (Bergen, Norway, 2003), pp.(in press).

[Teasley and Roschelle, 1993] Teasley, S. and Roschelle, J. *Constructing a joint problem space*. In Lajoie, S. and Derry, S. (Eds.), Computers as cognitive tools. Hillsdale, NJ: Lawrence Erlbaum, pp.229-257.

[Tedesco and Self, 2000] Tedesco, P. and Self, J. A. *Using meta-cognitive conflicts in a collaborative problem solving environment*. 5th International Conference on Intelligent Tutoring Systems (ITS 2000), (Montreal, Canada, 2000), pp.232-241.

[VanLehn, et al. 2005] VanLehn, K., Lynch, C., Schultz, K., Shapiro, J. A., Shelby, R. H., Taylor, L., Treacy, D. J., Weinstein, A., and Wintersgill, M. C. *The Andes physics tutoring system: Lessons learned.* International Journal of Artificial Intelligence and Education, 15(3).

[VanLehn, 2006] VanLehn, K. *The behavior of tutoring systems.* International Journal of Artificial Intelligence and Education, 16(3).

[Vizcaino, 2005] Vizcaino, A. *A Simulated Student Can Improve Collaborative Learning.* International Journal of Artificial Intelligence in Education, 15. pp. 3-40.

[Vygotsky, 1978] Vygotsky, L. S. *Mind in society: The development of higher psychological processes.* London: Harvard University Press.

[Webb, Troper and Fall, 1995] Webb, N. M., Troper, J. D. and Fall, R. *Constructive activity and learning in collaborative small groups*. Journal of Educational Psychology, 87. pp.406-423.

[Winter and McCalla, 1999] Winter, M. and McCalla, G. *The emergence of student models from an analysis of ethical decision making in a scenario-based learning environment*. 7th International Conference on User Modelling, (1999), Springer-Verlag, pp.265-274.

[Wortham, 1999] Wortham, D.W. *Nodal and matrix analyses of communication patterns in small groups*. 3rd International Conference on Computer Support for Collaborative Learning (CSCL 1999), (California, USA, 1999), pp.681-686.

# Appendix A

# Single-User Evaluation Forms

## A.1 Pre-Test

**1.** System analysts often examine textual requirements descriptions for domain model information. *Nouns* suggest:

**A.** *Classes* **D.** *Relationships*

**B.** *Attributes* **E.** *A and B*

**C.** *Methods* **F.** *C and D*

**2.** Which type of UML relationship would be used when one object merely *invokes methods* of another object?

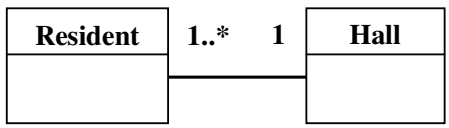
**A.** *Inheritance* **D.** *Aggregation*

**B.** *Dependency* **E.** *All of the above*

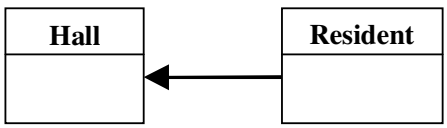
**C.** *Association* **F.** *None of the above*

**3.** Select the most appropriate option that best describes the given situation: "**Residents live in a student hall**"

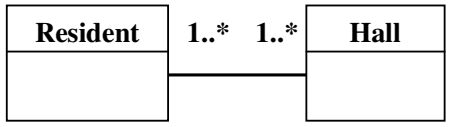
**A**



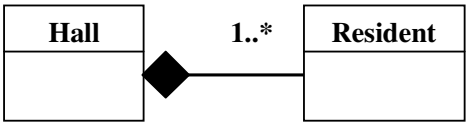

**C**



B

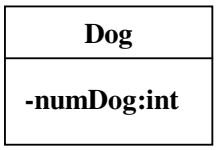



D

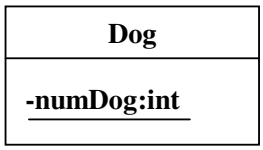



**4.** Which diagram best describes "**numDogs**" attribute? **numDogs** contains a count of the number of Dog instances. This count is accessed only within the class Dog.

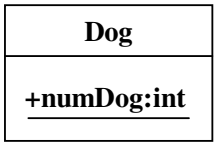
**A.**



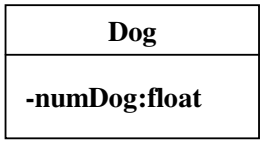

**C.**




**B.**




**D.**




**5.** Draw a UML class diagram to represent order payments. An order has a number and a price. There are two payment options: credit card and cheque. For each payment option, we store the payment date. For credit card option, the card number and the expiry date are recorded. For cheque option, the cheque number is stored.

## A.2 Post-Test

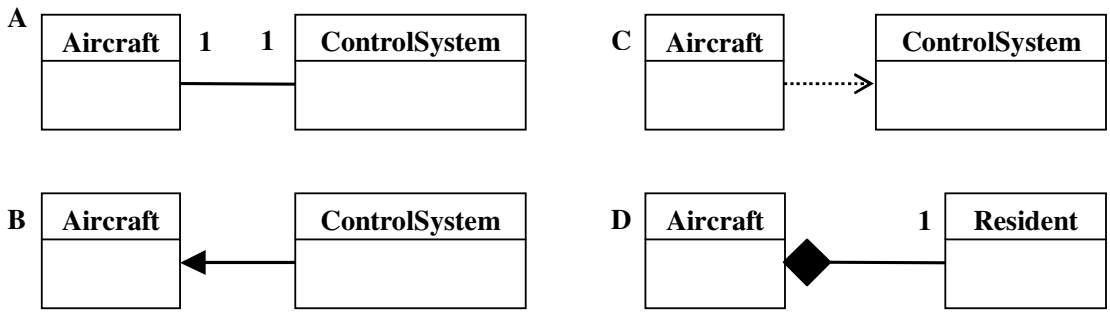
**1.** System analysts often examine textual requirements descriptions for domain model information. *Verbs* suggest:

**A.** *Classes*
**B.** *Attributes*
**C.** *Methods*
**D.** *Relationships*
**E.** *All of the above*
**F.** *None of the above*

**2.** Which type of UML relationship cannot have a relationship name?

**A.** *Composition*
**B.** *Association*
**C.** *Inheritance*
**D.** *Aggregation*
**E.** *All of the above*
**F.** *None of the above*

**3.** Select the most appropriate option that best describes the given situation: "**an aircraft has a control system.**"




**4.** In object-oriented software, attributes usually have a visibility of .....… and methods have a visibility of .....…

**A.** Public, Protected
**B.** Private, Protected
**C.** Private, Public
**D.** Public, Private

**5.** Draw a UML class diagram to represent customers. A customer has a name and an address and places one or more orders. Each order has a number and a date it was received. A customer can be either personal or corporate. For personal customers, the credit card number is recorded and for corporate customers, the credit card rating and limit are stored.

## A.3 Questionnaire

Thank you for using COLLECT-UML. Your feedback will be crucial for further improvements of the system and we would be most grateful, if you could take time to fill in this questionnaire. The questionnaire is anonymous, and you will not be identified as an informant. You may at any time withdraw your participation, including withdrawal of any information you have provided. By completing this questionnaire, however, it will be understood that you have consented to participate in the project and that you consent to publication of the results of the project with the understanding that anonymity will be preserved**.**

1. What is your previous experience with UML modelling? (Please circle one)

   A - Only lectures  B – Lectures plus some work  C – Extensive use

2. How much time did you need to learn about the system's functions? (Please circle one)

| | |
|---|---|
| (a) | Substantial time (most of the session) |
| (b) | 30 minutes |
| (c) | 10 minutes |
| (d) | Less than 5 minutes |

3. How much did you learn about UML modelling from using the system? (Please circle one)

| Nothing | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

   Please comment.

4. Did you enjoy learning with COLLECT-UML? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

5. Would you recommend COLLECT-UML to other students? (Please circle one)

A – No B- Don't know C - Yes

6. Did you find the interface easy to use? (Please circle one and add a comment)

Not at all Very much

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

7. Did you find the feedback from COLLECT-UML useful? (Please circle one and add a comment)

Not at all Very much

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

8. Would you prefer more details in feedback? (Please circle one and comment)

A – No B- Don't know C - Yes

9. Did you encounter any software problems or system crashes? If yes, please specify
10. What did you like in particular about COLLECT-UML?
11. Is there anything you found frustrating about the system?
12. Do you have any suggestions for improving COLLECT-UML?

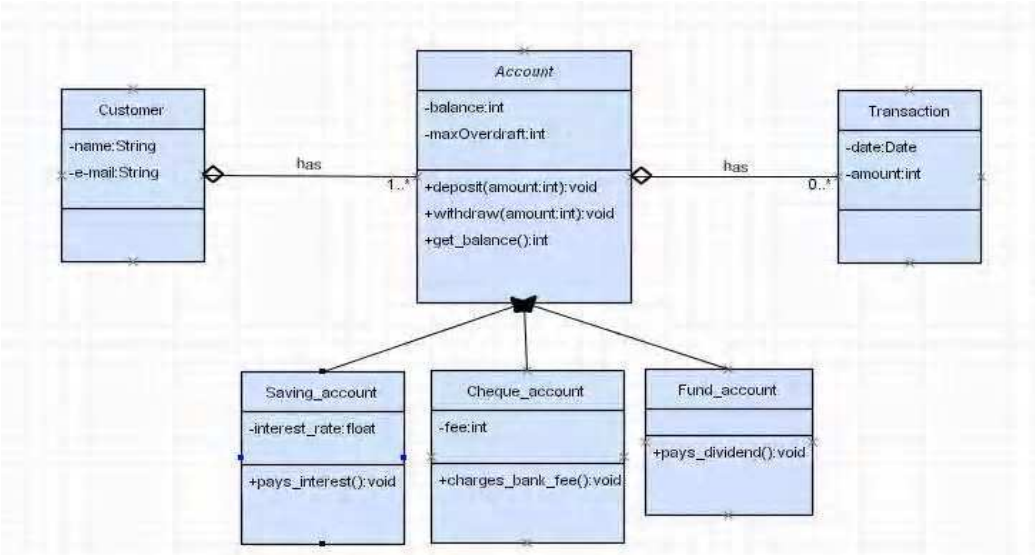
# Appendix B

# Multi-User Evaluation Forms

### B.1 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 over-draft, 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 cheque account charges bank fees and records the amount of fee 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.*

**Stored Solution**



## B.2 Post-Test

**1.** Describe the important aspects of successful collaboration during a collaborative problem-solving session?

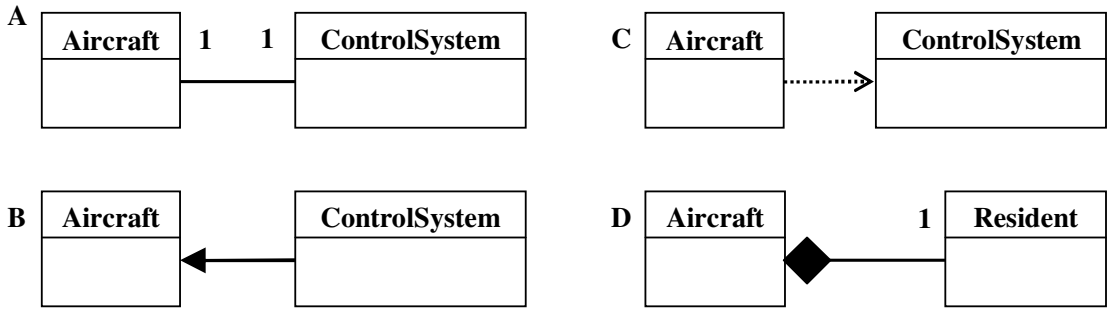
**2.** System analysts often examine textual requirements descriptions for domain model information. *Verbs* suggest:

**D.** *Classes* **D.** *Relationships*
**E.** *Attributes* **E**. *All of the above*
**F.** *Methods* **F.** *None of the above*

**3.** Which type of UML relationship cannot have a relationship name?

**D.** *Composition* **D.** *Aggregation*
**E.** *Association* **E.** *All of the above*
**F.** *Inheritance* **F.** *None of the above*

**4.** Select the most appropriate option that best describes the given situation: "**an aircraft has a control system.**"




**5.** In object-oriented software, attributes usually have a visibility of ..... and methods have a visibility of .....

**A.** Public, Protected
**B.** Private, Protected
**C.** Private, Public
**D.** Public, Private

**6.** Draw a UML class diagram to represent customers. A customer has a name and an address and places one or more orders. Each order has a number and a date it was received. A customer can be either personal or corporate. For personal customers, the credit card number is recorded and for corporate customers, the credit card rating and limit are stored.

## B.3 Questionnaire

Thank you for using COLLECT-$\mathcal{UML}$. Your feedback will be crucial for further improvements of the system and we would be most grateful, if you could take time to fill in this questionnaire. The questionnaire is anonymous, and you will not be identified as an informant. You may at any time withdraw your participation, including withdrawal of any information you have provided. By completing this questionnaire, however, it will be understood that you have consented to participate in the project and that you consent to publication of the results of the project with the understanding that anonymity will be preserved**.**

1. What is your previous experience with UML modelling? (Please circle one)

   A - Only lectures  B – Lectures plus some work  C – Extensive use

2. How much time did you need to learn about the system's functions? (Please circle one)

| | |
|---|---|
| (a) | Substantial time (most of the session) |
| (b) | 30 minutes |
| (c) | 10 minutes |
| (d) | Less than 5 minutes |

3. How much did you learn about UML modelling from using the system? (Please circle one)

   Nothing  Very much

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

   Please comment.

4. Did you enjoy learning with COLLECT-$\mathcal{UML}$? (Please circle one and add a comment)

   Not at all  Very much

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

5. Would you recommend COLLECT-$\mathcal{UML}$ to other students? (Please circle one)

   A – No  B- Don't know  C - Yes

6. Did you find the interface easy to use? (Please circle one and add a comment)

   Not at all  Very much

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

7. Did you find the feedback on collaboration (provided on your collaborative activities) useful? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

8. How effective was your partner's assistance in solving problems?

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

9. Do you think that working in a group can improve your learning?

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

10. Did you find the task-based feedback (provided on your UML solutions) useful? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

11. Would you prefer more details in the task-based feedback? (Please circle one and comment)

A – No B- Don't know C - Yes

12. Did you encounter any software problems or system crashes? If yes, please specify

a. What did you like in particular about COLLECT-UML?

b. Is there anything you found frustrating about the system?

c. Do you have any suggestions for improving COLLECT-UML?

## B.4 Highlights of Students' Feedback

**Suggestions for Improvement:**

- *Not being able to resize the problem text*
- *Not being able to modify a return type of a method, without re-creating it*
- *Not being able to copy a group of items and paste them into the shared diagram at a time*[3]
- *More details wanted in some of the problem texts*
- *Not being able to change the direction of a relationship using a right-click menu*

**Positive Comments:**

**General**

- *Easy to use, intuitive, pretty good, straightforward*
- *Very good learning tool*
- *Really enjoyed working with it*
- *More exciting than learning from a text-book*
- *Very good practical tool, helped me understand UML design*
- *Chat function was very useful*
- *It helped reinforce my knowledge of the diagrams*
- *It does look nice and makes nice diagrams*
- *It's obvious what every button does*
- *I learnt a lot about the relationships between classes*
- *It's fun, because it's easy to construct class diagrams*
- *It drives home a solid and much needed point on naming conventions*
- *Interface is very easy to use, just click, drag and drop*
- *Clarified a few areas of UML I wasn't sure on*
- *Highlighting the problem-text helped me concentrate on the problem*
- *Very good in teaching UML*
- *Nice way of learning through making class diagrams from specifications*
- *Great learning tool, co-operation social environment*
- *Make a stand-alone version that I can take home*
- *Seems quite easy to use once you understood how to use it*
- *Improves learning*

**Feedback**

- *Feedback helped correcting mistakes without giving away too much*
- *Very useful when you are stuck*
- *Helped me find problems I would not have seen otherwise*
- *More details in feedback would take away the task of solving the problem*
- *Seems to give enough info to help, but at the same time makes student think*
- *Feedback encourages us to work on the mistakes*
- *More details in feedback would have made it easy*

**Partner**

- *They may know things that I don't*
- *Larger groups could slightly be more useful*
- *Discussing ideas with a partner is really helpful. Asking them why they do what they did is good. It's important to know how they interpret the same question*
- *Very helpful, Great idea*
- *It's nice to be able to work with someone else*

[3] We thought allowing them to paste only one item at a time would result in more discussions between the team members

- *Always good to get a second opinion on things, you and your partner can learn from each other's strong points*
- *It allows people with better understanding of the subject to help those with less understanding*
- *Discussions with others reinforce ideas*

# Appendix C

# Publications

[1] Baghaei, N. and Mitrovic, A. (2005) *COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based tutor.* In Khosla, R., Howlett, R. and Jain L. (eds.) Knowledge-Based Intelligent Information & Engineering Systems (KES 2005), pp.458-464.

[2] Baghaei, N., Mitrovic, A. and Irwin, W. (2005) *A Constraint-Based Tutor for Learning Object-Oriented Analysis and Design using UML.* In Looi, C., Jonassen, D. and Ikeda M. (eds.) International Conference on Computers in Education (ICCE 2005), pp.11-18.

[3] Baghaei, N., Mitrovic, A. and Irwin, W. (2006) *Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams.* Technology, Instruction, Cognition and Learning (TICL) Journal, 4(1-2), pp.113-137.

[4] Baghaei, N. (2006) *A Collaborative Constraint-based Intelligent System for Learning UML Class Diagrams.* Computing Women Congress (CWC 2006), (Hamilton, New Zealand), 11-19 February.

[5] Baghaei, N., and Mitrovic, A. (2006) *A Constraint-based Collaborative Intelligent System for Learning UML Class Diagrams.* In Ikeda, M., Ashley, K. and Chan T.-W. (eds.)

International Conference on Intelligent Tutoring Systems (ITS 2006), (Jhongli, Taiwan), pp.176-186.

[6] Baghaei, N. (2006) *A Collaborative Constraint-based Adaptive System for Learning Object-Oriented Analysis and Design using UML.* In Wade, V., Ashman, H. and Smyth B. (eds.) International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2006), (Dublin, Ireland), pp.398-403.

[7] Baghaei, N., Mitrovic, A. and Irwin, W. *Supporting Collaborative Learning and Problem-Solving in a Constraint-based CSCL Environment for UML Class Diagrams.* International Journal of Computer-Supported Collaborative Learning (CSCL), 2007. 2(2-3), pp. 159-190.

[8] Baghaei, N. and Mitrovic, A. *From Modelling Domain knowledge to Metacognitive Skills: Extending a Constraint-based Tutoring System to Support Collaboration.* In Conati C., McCoy K. and Paliouras G. (eds.) International Conference on User Modeling (UM 2007), (Corfu, Greece), pp.217-227.

[9] Baghaei, N. and Mitrovic, A. *Evaluating a Collaborative Constraint-based Environment for UML Class Diagrams*. In Luckin R., Koedinger K. and Greer J. (eds.) International Conference on Artificial Intelligence in Education (AIED 2007), (Los Angeles, CA, USA), pp.533-535.

# Appendix D

# Example Meta-Constraints

## Category 1: Initial Planning

(244
IF [relevance condition]
In each session
THEN [satisfaction condition]
It is necessary that the student has used *Introduce* communication category
ELSE
Message: "**Would you like to introduce yourself to your team-mates and plan the session?"**

(245
IF [relevance condition]
In each session
THEN [satisfaction condition]
It is necessary that the student's contribution includes a class
OR the student's contribution includes a method
OR the student's contribution includes an attribute
OR the student's contribution includes a relationship
ELSE
Message: "**You may wish to think about the problem and construct a UML diagram in your individual workspace first, before joining the group discussion."**

## Category 2: Use of Communication Categories

(236
IF [relevance condition]
The student has contributed to the chat area,
THEN [satisfaction condition]
It is necessary that the student has used one of the following communication categories:

*Request*, *Inform*, *Motivate, Task*, *Acknowledge, Maintenance* or *Argue.*
ELSE
Message: "**You seem to just agree and/or disagree with other members. You may wish to challenge others ideas and ask for explanation and justification."**

(237
IF [relevance condition]
The student has contributed to the chat area,
AND the student has used either *Agree* or *Disagree* communication categories
THEN [satisfaction condition]
It is necessary that the contribution includes some text
ELSE
Message: "**You may wish to explain to other members why you agree or disagree with a solution."**

(238
IF [relevance condition]
The student has contributed to the chat area
THEN [satisfaction condition]
It is necessary that the contribution includes some text
ELSE
Message: **"Ensure adequate elaboration is provided in explanations.**"

(239
IF [relevance condition]
The student's collaboration has a class *C,*
OR the student's collaboration has a method *m,*
OR the student's collaboration has an attribute *a,*
OR the student's collaboration has a relationship *r,*
THEN [satisfaction condition]
It is necessary that the student has contributed to the chat area,
AND the student has used one of the following communication categories: *Request*, *Inform*, *Motivate, Task*, *Maintenance* or *Argue*.
ELSE
Message: "**You may wish to give explanation and provide justification each time you make a change to the shared diagram.**"

## Category 3: Comparing Individual Diagrams with the Group Diagram and vice versa

(220
IF [relevance condition]
The student's solution has a class *C*
THEN [satisfaction condition]
It is necessary that the group's solution has a class *C*
ELSE
Message: **"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."**

(222
IF [relevance condition]
The student's solution has a class *C,*
The student's solution has a composition relationship of *C* named *r,*
The group's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the group's solution has a composition relationship of *C* named *r*
ELSE
Message: **"Some relationship types (compositions) in your individual solution are missing in the group diagram. You may wish to share your work by adding those composition(s)/discuss it with other members."**

(226
IF [relevance condition]
The student's solution has a class *C,*
The student's solution has an attribute of *C* named *a,*
The group's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the group's solution has an attribute of *C* named *a*
ELSE
Message: **"Some attributes in your individual solution are missing in the group diagram. You may wish to share your work by adding those attribute(s)/discuss it with other members."**

(227
IF [relevance condition]
The student's solution has a class *C,*
The student's solution has a method of *C* named *m,*
The group's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the group's solution has a method of *C* named *m*
ELSE
Message: **"Some methods in your individual solution are missing in the group diagram. You may wish to share your work by adding those method(s)/discuss it with other members."**

(228
IF [relevance condition]
The group's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the student's solution has a class *C*
ELSE
Message: **"Some classes in the group diagram are missing from your individual diagram. You may wish to discuss this with other members."**

(229
IF [relevance condition]

The group's solution has a class *C,*
The group's solution has an association relationship of *C* named *r,*
The student's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that student's solution has an association relationship of *C* named *r*
ELSE
Message: **"Some association relationship in the group diagram are missing from your individual diagram. You may wish to discuss this with other members."**

(234
IF [relevance condition]
The group's solution has a class *C,*
The group's solution has an attribute of *C* named *a,*
The student's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the student's solution has an attribute of *C* named *a*
ELSE
Message: **"Some attributes in the group diagram are missing from your individual diagram. You may wish to discuss this with other members."**

(235
IF [relevance condition]
The group's solution has a class *C,*
The group's solution has a method of *C* named *m,*
The student's solution has a class *C,*
THEN [satisfaction condition]
It is necessary that the student's solution has a method of *C* named *m*
ELSE
Message: **"Some methods in the group diagram are missing from your individual diagram. You may wish to discuss this with other members."**

## Category 4: Contribution to the Group Diagram

(240
IF [relevance condition]
In each session
THEN [satisfaction condition]
It is necessary that the student's contribution includes a class
OR the student's contribution includes a method
OR the student's contribution includes an attribute
OR the student's contribution includes a relationship
OR the student's contribution includes a chat contribution
ELSE
Message: "**Would you like to contribute to the group discussion?"**

(241

IF [relevance condition]

The student has contributed to the chat area

THEN [satisfaction condition]

It is necessary that the student's contribution includes a class
OR the student's contribution includes a method
OR the student's contribution includes an attribute
OR the student's contribution includes a relationship

ELSE

Message: "**Would you like to contribute to the construction of the group diagram?"**

# KES 2005 Paper

The following paper [Baghaei and Mitrovic, 2005] was published and presented at the KES 2005 conference in Melbourne, Australia.

# COLLECT-UML: Supporting Individual and Collaborative Learning of UML class diagrams in a Constraint-Based Intelligent Tutoring System

Nilufar Baghaei and Antonija Mitrovic

Intelligent Computer Tutoring Group
Department of Computer Science & Software Engineering
University of Canterbury, Private Bag 4800, New Zealand
{N.Baghaei, T.Mitrovic}@cosc.canterbury.ac.nz

**Abstract.** Automatic analysis of interaction and support for group learning through a distance collaborative learning system is at the forefront of educational technology. Research shows that collaborative learning provides an environment to enrich the learning process by introducing interactive partners into an educational system. Many collaborative learning environments have been proposed and used with more or less success. Researchers have been exploring different approaches to analyse and support the collaborative learning interaction. However, the concept of supporting peer-to-peer interaction in Computer-Supported Collaborative Learning (CSCL) systems is still in its infancy, and more studies are needed that test the utility of these techniques. This paper proposes an Intelligent CSCL system that uses Constraint-Based Modeling (CBM) approach, to support collaborative learning addressing both collaborative issues and task-oriented issues. The system supports the tertiary students learning Object-Oriented Analysis and Design using UML. The CBM approach is extremely efficient, and it overcomes many problems that other student modeling approaches suffer from [5]. CBM has been used successfully in several tutors supporting individual learning. The comprehensive evaluation studies of this research will provide a measure of the effectiveness of using CBM technique in Intelligent CSCL environments.

## 1 Introduction

E-learning is becoming an increasingly popular educational paradigm as more individuals who are working or are geographically isolated seek higher education. Support for collaboration is especially important in this context, because the lack of face-to-face interaction complicates the collaboration and students have few opportunities to practice collaborative skills [1]. There have been several definitions for collaborative learning. The broadest (but unsatisfactory) definition is that it is a *situation* in which *two* or *more* people *learn* or attempt to learn something *together* [2]. A more comprehensive definition given by [3] states as follows: “ a coordinated, synchronous activity that is the result of a continued attempt to construct and maintain a

shared conception of a problem". Since effective collaborative learning includes both learning to effectively collaborate, and collaborate effectively to learn, the facilitator must be able to address social or collaboration issues as well as task-oriented issues [4]. Collaboration issues include the distribution of roles among students (e.g. critic, mediator, idea-generator), equality of participation, and reaching a common understanding, while task-oriented issues involve the understanding and application of key domain concepts. The educational systems are responsible for regulating the interaction, and guiding the students towards effective collaboration and learning.

In the last decade, many researchers have contributed to the development of CSCL and advantages of collaborative learning over individualised learning have been identified. Numerous systems for collaborative learning have been developed (discussed in the following section); however, the concept of supporting peer-to-peer interaction in CSCL systems is still in its infancy.

This paper proposes an Intelligent CSCL system that uses Constraint-Based Modeling (CBM) approach, to support collaborative learning addressing both collaborative issues and task-oriented issues. The CBM approach is extremely efficient, and it overcomes many problems that other student modeling approaches suffer from [5]. The system supports the tertiary students learning Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD) using UML.

Section 2 reviews some of the collaborative learning systems that have been developed. Section 3 outlines the objectives of this research followed by the intended approach in Section 4. Current and Future work is discussed in the last section.

## 2 Related Work

This section discusses examples of three types of CSCL systems, in the context of the collaboration management model [4]:

- *Reflecting Actions:* The most basic level of support a system may offer involves making the students aware of the participants' actions. Actions taken on shared resources or those that take place in private areas of a workspace may not be directly visible to the collaborators, yet they may significantly influence the collaboration. Raising awareness about such actions may help students maintain a representation of their team-mates' activities. The system described in [6] is an example.
- *Monitoring the State of Interactions*: Such systems fall into two categories: those that aggregate the interaction data into a set of high-level indicators, and display them to the participants, and those that internally compare the current state of interaction to a model of ideal interaction, but do not reveal this information to the users. In the former case, the learners are expected to manage the interaction themselves, having been given the appropriate information to do so. In the latter case, this information is either intended to be used later by a coaching agent, or analysed by researchers in an effort to understand the interaction [4]. MArCo [7] and EPSILON [8] are examples of such systems.
- *Offering Advice:* This will include the CSCL systems that analyse the state of collaboration using a model of interaction, and offer advice intended to increase the

effectiveness of the learning process. The coach in an advising system plays a role similar to that of a teacher in a collaborative learning classroom. The systems can be distinguished by the nature of the information in their models, and whether they provide advice on strictly collaboration issues or both social and task-oriented issues. Examples of the systems focusing on the social aspects of collaborative learning include Group Leader Tutor [9] and DEGREE [10] and examples of the systems addressing both social and task-oriented aspects of group learning are LeCS [11], COLER [12] and COMET [13].

## 3 Research Objectives

The main objective of this research is to develop an intelligent collaborative system for the tertiary students learning object-oriented analysis and design using UML. It focuses on CBM, which uses constraints to represent the domain knowledge. Constraints are used to identify errors in the student solution. CBM technique will be used to model student/group knowledge, and represent the domain knowledge as a set of syntax and semantic constraints.

A number of evaluation studies will be conducted to test the feedback messages (both task-based and collaborative-based feedback), the pedagogical agent, and the usability of the interface.

The research will evaluate the possibility of giving advice with comparing student work with an expert solution *as well as* group solution, in contrast to the approach usually taken by previous studies that have either supported tutoring (teaching the domain concepts) or coaching the social interaction (encouraging the students to discuss and participate*)*. Therefore, the proposed system can be considered as a learning environment that supports both individual and collaborative learning.

## 4 Research Approach

The project is divided into two main parts: support for problem solving and support for social interaction between group members. The system will support two or more students working together on a problem from networked machines. Communication windows are available so that students can send advice, suggest alternative actions, comment on their partner's actions, etc. One of the learners would act as moderator, who will be responsible for the final submission of the solution to the system after getting approval from all the group members. The following subsections provide more detailed information on individual components of the system.

### 4.1 Architecture

The system will use *distributed architecture* [14], where the tutoring functionally is distributed between the client and the server. The application server consists of a

student modeler, which creates and maintains student models for all users, a domain module, a pedagogical module and a group modeler.

The user interface is Java-based and may perform some teaching functions. The architecture of the proposed system is illustrated in Figure 1. The system is being implemented in Allegro Common Lisp.

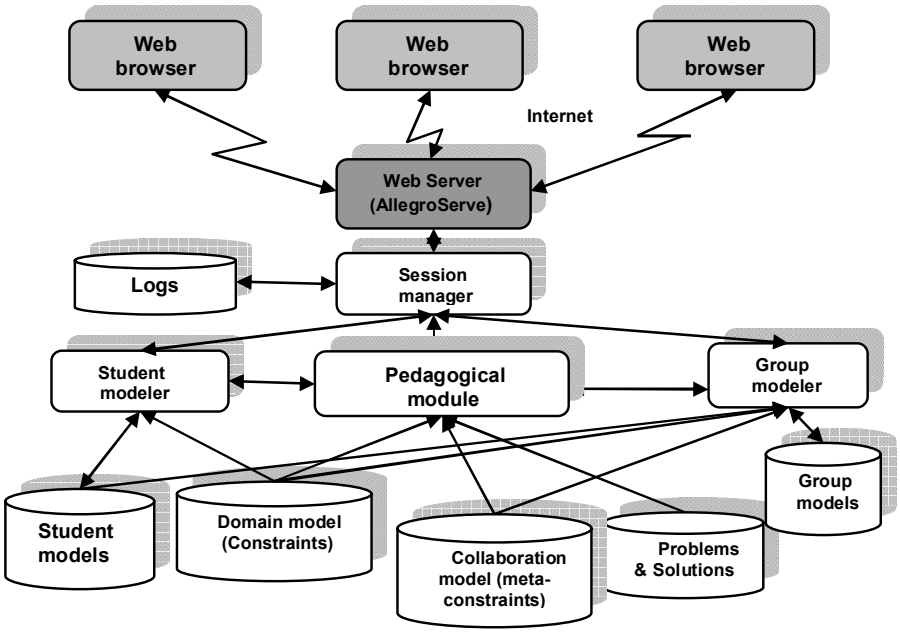



**Fig. 1.** The architecture for the proposed system

### 4.2 Interface

The interface of the system is divided into three main parts: the section on the right hand side will be the individual workspace, the one in the middle will be the shared workspace and the one on the left hand side will show the system feedback messages to the group members. The text of the problem is being displayed at the top and sentence openers and a chat area is provided at the bottom of the shared workspace to facilitate synchronous communication between the group members.

When a solution is submitted, the pedagogical module generates feedback on it, offers the possibilities of working on the same problem (if there were mistakes in the solution), logging off, or going on to the next problem, which will be selected by the system. The student is also able to view the history of the session, and specify the kind of feedback required. Both student's individual solution and group solution (submitted by the moderator) will be compared with ideal solution stored in the system.

The input data (for the system to further analyse) are: Shared workspace, Private work area (for individual members of the group), Chat facility with use of sentence openers and three buttons (OK, NO, Not sure) which will be used by each individual member to express their opinions whenever the shared workspace is updated. Only one student, the one who holds the pencil, can update the shared diagram at a given time. Figure 2 illustrates the current state of the interface implementation, supporting individual learning.

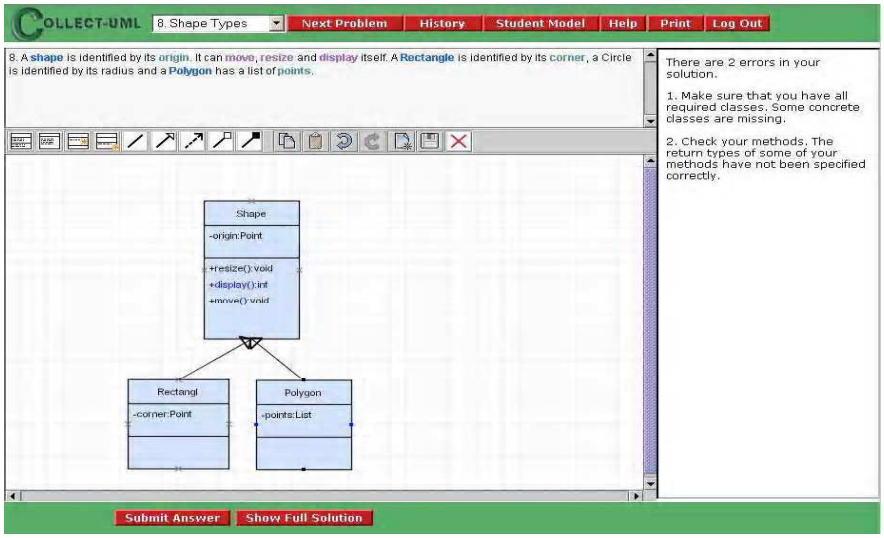



**Fig. 2.** The current state of COLLECT-UML interface

### 4.3 Group Modeller

A new component will be added to the system, which will be responsible for:

- Recording history of students' social interactions with each other through the chat facility,
- Proposing new learning tasks based on the learning needs of the group,
- Recording history of group technical knowledge as a whole; a group solution is considered the one that is submitted by one student as a representative (after all the team members agree on the solution being submitted).

### 4.4 Pedagogical Module

The pedagogical module provides support for individual and collaborative learning:

- The student can work in the private area and submit his/her solution to the system; the solution will be compared with an ideal solution and the system provides feedback for improving their individual performance; feedback would be on demand and task-based in this case.
- The group members will collaborate and submit a final solution to the system; the solution is compared with expert solution and feedback will be provided. Feedback would be on demand and task-based.
- The group solution will be compared with each member's solution and discussion items based on the differences will be recommended. This will provide participation and negotiations between the group members.
- Feedback messages will be given to each group member (from time to time), encouraging them to discuss, participate, acknowledge, etc. The feedback messages will be generated based on student performance in chat area, agreement buttons and shared workspace.

### 4.5 Domain and Collaboration Model

The Domain model has already been implemented as a set of syntax and semantic constraints (88 semantic and 45 syntax constraints so far). Each constraint consists of a relevance condition, a satisfaction condition and a feedback message. The feedback messages are presented when the constraint is violated. It is common in any design problem to have two or more correct solutions for a problem, especially if the problem is complex. The system will contain only one correct solution to the problem. However, the system will be capable of recognizing alternative correct solutions, as there are constraints that check for equivalent constructs in the student's and ideal solutions. Figure 3 illustrates an example of a semantic constraint for the UML domain.

Meta constrains compare student's interaction with an ideal model of interaction and will provide collaborative-based feedback messages; the social-interaction comments generated by the system can be discarded by the student. Meta-constraints will be evaluated from time to time by taking into account the content of chat area, contribution to the group diagram, and comparison of group and individual diagram.

## 5 Conclusions and Future Work

The single-user version of the system has already been designed and implemented. It will be extended to support collaborative learning as the next step. The enhancement process will include implementation of shared workspace, modification of pedagogical module to support both individual and group of users, designing and implementing the group modeling component and developing meta-constraints. The comprehensive evaluation of the system will be carried out at the University of Canterbury, in a second-year Software Engineering course. The evaluation studies of this research will provide a measure of the effectiveness of using CBM technique in intelligent computer supported collaborative learning environments.

```
(77
   "Check your methods. You have method(s) of type non-static,
    when it should be static."
   (and (match IS METHODS (?* "@" ?tag ?name ?c_tag ?type_IS
                           ?access_IS ?static_IS ?*))
        (match SS METHODS (?* "@" ?tag ?name2 ?c_tag ?type_SS
                           ?access_SS ?static_SS ?*))
        (test IS ("yes" ?static_IS)))
   (not-p (test SS ("no" ?static_SS)))
   "methods"
   (?tag))
```

**Fig. 3.** An example of a semantic constraint

## References


1. Constantino-Gonzalez, M., and Suthers, D. Coaching Collaboration in a Computer Mediated Learning Environment. (CSCL 2002), (NJ, USA, 2002), pp.583-584.
2. Dillenbourg, P. What do you mean by "Collaborative Learning". In Dillenbourg, P. (Eds.), Collaborative Learning: Cognitive and Computational Approaches, Amsterdam: Elsevier Science. pp.1-19, 1999.
3. Roschelle, J. and Teasley, S.D. The construction of shared knowledge in collaborative problem solving. In O'Malley, C.E. (Eds), Computer-Supported Collaborative Learning. Berlin, pp. 69-197, 1995.
4. Jerman, P., Soller, A. and Muhlenbrock, M. From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning. In Dillenbourg, P., Eurelings, A and Hakkarainen, K. (Eds.), European Perspectives on CSCL (CSCL 2001), (Maastricht, Netherlands, 2001), pp.324-331.
5. Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B. Constraint-based Tutors: a Success Story. 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2001), (Budapest, 2001), pp.931-940.
6. Plaisant, C., Rose, A., Rubloff, G., Salter, R. and Shneiderman, B. The design of history mechanisms and their use in collaborative educational simulations. (CSCL 1999), (California, USA, 1999), pp.348-359.
7. Tedesco, P. and Self, J. A. Using meta-cognitive conflicts in a collaborative problem solving environment. (ITS 2000), (Montreal, Canada, 2000), pp.232-241.
8. Soller, A. and Lesgold, *A*. Knowledge acquisition for adaptive collaborative learning environments. AAAI Fall Symposium: Learning How to Do Things, Cape Cod, MA, 2000.
9. McManus, M. and Aiken, R. Monitoring computer-based problem solving. International Journal of Artificial Intelligence in Education, 6(4). pp.307-336, 1995.
10. Barros, B. and Verdejo, M.F. Analysing student interaction processes in order to improve collaboration: The DEGREE approach. International Journal of Artificial Intelligence in Education, 11. pp.221-241, 2000.
11. Rosatelli, M., Self, J., and Thirty, M. LeCs: A collaborative case study system. (ITS 2000), (Montreal, Canada, 2000), pp.242-251.
12. Constantino-Gonzalez, M., and Suthers, D. A coached collaborative learning environment for Entity-Relationship modelling. 5th International Conference on ITSs (ITS 2000), (Montreal, Canada, 2000), pp.324-333.
13. Suebnukarn, S. and Haddawy, P. A Collaborative Intelligent Tutoring System for Medical Problem-Based Learning. International Conference on Intelligent User Interfaces, (Madeira, Portugal, 2004).
14. Mitrovic, A. An intelligent SQl tutor on the Web. International Journal of Artificial Intelligence in Education, 13 (2-4). pp.173-197, 2003.

# ICCE 2005 Paper

The following paper [Baghaei, Mitrovic and Irwin, 2005] was published and presented at the ICCE 2005 conference in Singapore.

# A Constraint-Based Tutor for Learning Object-Oriented Analysis and Design using UML

**Nilufar BAGHAEI, Antonija MITROVIC and Warwick IRWIN**
*Department of Computer Science and Software Engineering*
*University of Canterbury, Private Bag 4800, New Zealand*
*Phone number: ++64 (3) 3642987 Ext. 7756*
{N.Baghaei, T.Mitrovic, W.Irwin}@cosc.canterbury.ac.nz

**Abstract.** COLLECT-$\mathcal{UML}$ is an intelligent tutoring system that teaches Object-Oriented design using Unified Modelling Language (UML). UML is one of the most popular techniques used in the design and development of Object-Oriented systems nowadays. The Constraint-Based Modelling (CBM) has been used successfully in several systems and they have proved to be extremely effective in evaluations performed in real classrooms. In this paper, we present our experiences in implementing another constraint-based tutor, in the area of Object-Oriented design. We present the system's architecture and functionality and describe the results of a preliminary study with postgraduate students who interacted with the system as part of a think-aloud study. Participants felt that using the system helped them improve their UML knowledge. A full evaluation study is planned for May 2005, which aims to evaluate the interface and the effect of using the system on students' learning.

**Keywords:** ITS, Constraint-Based Modelling, Object-Oriented design, UML

## Introduction

Previous work has shown that the Constraint-Based Modelling (CBM) [9] is extremely efficient, and it overcomes many problems that other student modeling approaches suffer from [6]. CBM has been successfully applied in a number of domains. These tutors, called constraint-based tutors [6] have been developed in domains such as SQL (the database query language) [5, 8], database modelling [11, 12], data normalization [7], and punctuation [4]. All three tutors in the database domain were developed as problem solving environments for tertiary students. Students solve problems presented to them with the assistance of feedback from the system. The punctuation tutor was developed with the goal of improving the capitalisation and punctuation skills of 10-11 year old school children.

This paper presents our experiences in implementing a constraint-based tutor in the area of object-oriented design. UML modelling is one of the most popular techniques used in the design and development of object-oriented systems nowadays. UML was selected as an appropriate task for this research due mainly to its open-ended nature, and its complexity for novice designers.

Although the traditional method of learning UML in a classroom environment may be sufficient as an introduction to the concepts of object-oriented design, students cannot gain expertise in the domain by attending lectures only. Therefore, the existence of a computerized tutor, which would support students in gaining such design skills, would be highly useful.

We start with a brief overview of related work in Section 1. Section 2 describes the overall architecture of the system, followed by the pilot study presented in Section 3. Conclusions and Future work are discussed in the last section.

## 1. Related Work

Having found a lot of tutorials, textbooks and resources on UML, we are not aware of any attempt at developing an ITS for UML modelling. However, there has been an attempt [10] at developing a collaborative learning environment for object-oriented design problems using Object Modeling Technique (OMT).

The system 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 which 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 determines when and why the students are having trouble learning the new concepts they share with each other.

The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities. In this regard, even though the system is effective as a collaboration tool, it would not be an effective teaching system for a group of novices with the same level of expertise, as it could be common for a group of students to agree on the same flawed argument.

## 2. COLLECT-*UML*: A Knowledge-Based UML Modelling Tutor

COLLECT-*UML* is a web-based problem-solving environment, in which students are required to construct UML class diagrams that satisfy a given set of requirements. It assists students during problem solving and guides them towards a correct solution by providing feedback. The feedback is tailored towards each student depending on his/her knowledge. COLLECT-*UML* is designed as a complement to classroom teaching and when providing assistance, it assumes that the students are already familiar with the fundamentals of object-oriented software design.

### *2.1 Architecture*

The system has a *distributed architecture* [8], where the tutoring functionally is distributed between the client and the server (Figure 1). The application server consists of a student modeler, which creates and maintains student models for all users, a domain module and a pedagogical module. The user interface is Java-based (discussed in Section 3.3) and performs some teaching functions including immediate feedback for some problem-solving steps. The system is implemented in Allegro Common Lisp, which provides a development environment with an integrated Web Server (AllegroServe) [1].

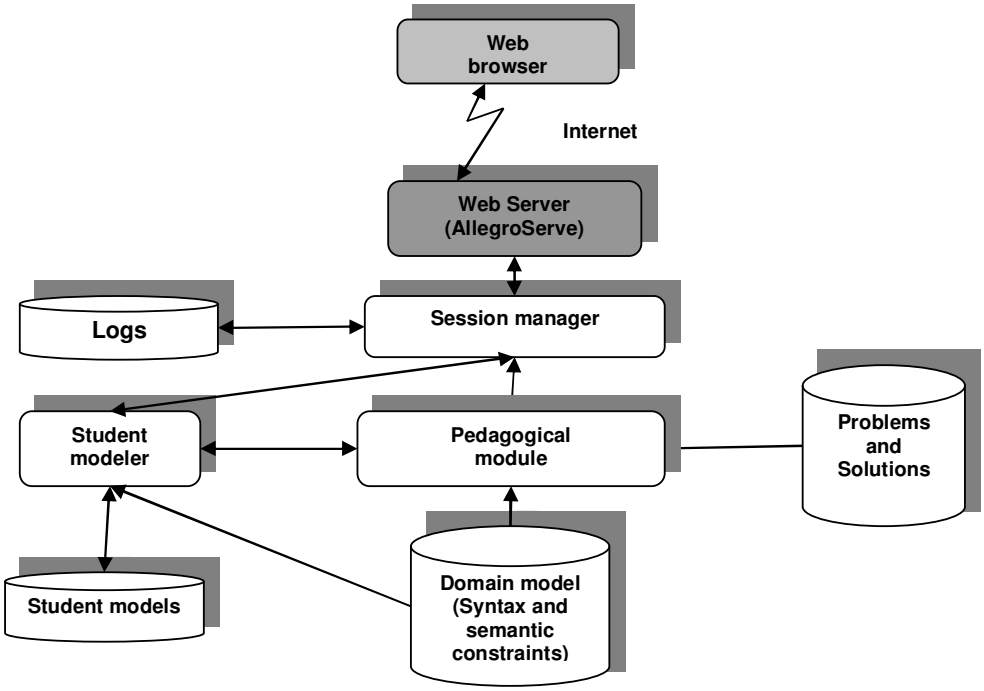



***Figure 1**: The architecture of the system*

*2.2 Domain constraints*

The system is able to diagnose students' solutions by using its domain knowledge represented as a set of syntax and semantic constraints (88 semantic and 45 syntax constraints). Each constraint consists of a relevance condition, a satisfaction condition and a feedback message. The feedback messages are presented when the constraint is violated. It is common in any design problem to have two or more correct solutions for a problem, especially if the problem is complex. The system contains only one correct solution for each problem. However, the system is capable of recognizing alternative correct solutions, as there are constraints that check for equivalent constructs in the student's and ideal solutions. Figure 2 gives examples of syntax and semantic constraints for the UML domain.

*2.3 Interface*

The interface is an HTML page, containing a Java Applet, which was implemented using API specification for the Java 2 Platform, version 1.4.2 (Figure 4). The current version of the applet contains 4 packages, 75 Java classes and 6853 lines of code.

In order to draw a UML diagram, the user selects the appropriate shape from the drawing toolbar and then positions the cursor on the desired place within the drawing area. Shapes can be resized by selecting them first, and then dragging the blue handles (shown as rectangles when the component is selected). The shape will remain red until the student selects a name for it from the problem text. A name can be selected either by double clicking on a word from the problem text, or by highlighting a phrase. The highlighted words are coloured depending on the type of the component. The feature is advantageous from a pedagogical point of view, as the student must follow the problem text closely. Many of the errors in students' solutions occur because they have not comprehensively read and understood the problem. These mistakes would be minimised in COLLECT-UML,

```
; Semantic

(117
  "Make sure each subclass has at least one specific (local) attribute or
   method."
  (and (match SS SUBCLASSES (?* "@" ?subtag ?*))
       (match IS SUBCLASSES (?* "@" ?subtag ?*))
       (or-p (match IS ATTRIBUTES (?* "@" ?attr_tag ?attr_name ?subtag
                                   ?*))
             (match IS METHODS (?* "@" ?method_tag ?method_name ?subtag
                                ?*))))
  (or-p (match SS ATTRIBUTES (?* "@" ?attr_tag1 ?attr_name1 ?subtag ?*))
        (match SS METHODS (?* "@" ?method_tag1 ?method_name1 ?subtag ?*)))
 "specialisation/generalisation"
 (?subtag))

; Syntax

(100
  "Check your inheritances. Two classes can not inherit from each other."
  (and (match SS SUBCLASSES (?* "@" ?subtag ?supertag ?*))
       (not-p (test SS ("null" ?subtag)))
       (not-p (test SS ("null" ?supertag))))
  (not-p (match SS SUBCLASSES (?* "@" ?subtag ?supertag ?* "@" ?supertag
                               ?subtag ?*)))
  "specialisation/generalisation"
  (?subtag ?supertag))
```

***Figure 2:*** *Examples of syntax and semantic constraints*

as students are required to focus their attention on the problem text every time they add a new component. Highlighting is also useful from the point of view of the student modeller for evaluating solutions [12]. There is no standard that is enforced in naming classes, methods, attributes or relationships. Since the names of the components in the student solution (SS) may not match the names of construct in the ideal solution (IS), the task of finding correspondence between the constructs of the SS and IS is difficult. This problem is avoided by forcing the student to highlight the word or phrase that is modelled by each component in the UML diagram.

To create a new attribute or a method, the student needs to select the parent class first, and then either clicks on relevant toolbar icon or right-clicks on the class and chooses the *"New ..."* menu option. The properties of an existing component (class, attribute, method or relationship) can be changed by right-clicking on that component and choosing the relevant menu option. To connect two classes of the diagram, the student needs to select the appropriate type of relationship. A relationship will be shown in red if it is not properly attached to other classes. Clicking on *"Student Model"* button will display an overview of their knowledge.

Currently, the system contains 14 problems, which cover different aspects of Object-Oriented modeling, and their ideal solutions. The ideal solutions are UML class diagrams that fulfil all the problem requirements. Figure 3 shows a sample problem and the internal representation of its ideal solution, which consists of 6 clauses (i.e. RELATIONSHIPS, ATTRIBUTES, METHODS, CLASSES, SUPERCLASSES and SUBCLASSES). The problem text is represented internally with embedded tags that specify the mapping to the components in the ideal solution. The tags are not visible to the student since they are extracted before the problem is displayed.

The applet saves the solutions submitted by students into XML files, which are converted to internal representation using an XSLT style-sheet. The constraints are applied to the internal representation of the solutions and feedback is given to students, using the messages attached to the violated constraints.

```
(5                                ; problem number
  5                               ; difficulty
  "5. 5. An <E1> owner </E1> <R1> owns </R1> one or more <E2> vehicle </E2>s.
   Each <E2> vehicle </E2> has a <E2A1> gross weight </E2A1>. Each <E1> owner
   </E1> has a <E1A1> number </E1A1> (unique) and a <E1A2> name </E1A2> and
   <E1A3> register </E1A3> a number of <E1> vehicle </E1>s. Each <E1> vehicle
   </E1> can be either <E3> bike </E3> or <E4> car </E4>. For each <E3> bike
   </E3>, the software records the <E3A1> serial number </E3A1> and for each
   <E4> car </E4>, the <E4A1> license plate </E4A1> and the <E4A2> color
   </E4A2> are recorded."

(("RELATIONSHIPS" "@ R1 association E2 E1 1..* 1 null null owns @ R99
       inheritance E2 E4 null null null null null @ R99 inheritance E2 E3 null
       null null null null ")
  ("ATTRIBUTES" "@ E1A2 name E1 String private no @ E1A1 number E1 String
       private no @ E2A1 weight E2 float private no @ E3A1 serial_number E3
       String private no @ E4A1 license_plate E4 String private no @ E4A2
       color E4 Color private no ")
  ("METHODS" "@ E1A3 registers E1 void public no 1 vehicles List null null
       null null ")
  ("CLASSES" "@ E1 Owner concrete @ E2 Vehicle concrete @ E3 Bike concrete @
       E4 Car concrete ")
  ("SUPERCLASSES" "@ E2 E3 @ E2 E4 ")
  ("SUBCLASSES" "@ E3 E2 @ E4 E2 "))
    "5.jpg"
    "Vehicles")
```

***Figure 3:*** *A sample problem and its ideal solution*

## 3. Pilot Study

A pilot study was conducted as a think-aloud protocol in March 2005. The study aimed to discover users' perceptions about various aspects of the system, mainly the quality of feedback messages and the interface.

The participants were 12 postgraduate students enrolled in an Intelligent Tutoring Systems course at the University of Canterbury. Even though the target population for COLLECT-$\mathcal{UML}$ is undergraduates who are learning UML software design, it was not possible to gain access to this population at the time of the pilot study, because the UML class diagrams had not been covered in the lectures. The participants had completed 50% of the ITS course lectures, and were expected to have a good understanding of ITS. All participants except two were familiar with UML modelling.

The study was carried out in the form of a think-aloud protocol [2]. 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 [3]. Each participant was asked to verbalise his/her thoughts while performing a UML modelling task using COLLECT-$\mathcal{UML}$. Participants were able to skip the problems without completing them and to return to previous problems.

Data was collected from video footages of think-aloud sessions, informal discussions after the session and researcher's observations.

*3.1 Students' impressions on interface*

The majority of the participants felt that the interface was nicely designed and the drawing area was big enough for them to work on the problems given. Three participants felt that some of the hints provided by the system were not helpful enough for them to correct their mistakes. For example, one participant had a class called *Shape* and an attribute (belonged to that class) called *Origin*. The attribute type had been specified as *int*, when the ideal solution expected them to have defined the attribute of type *Point*. The feedback in this

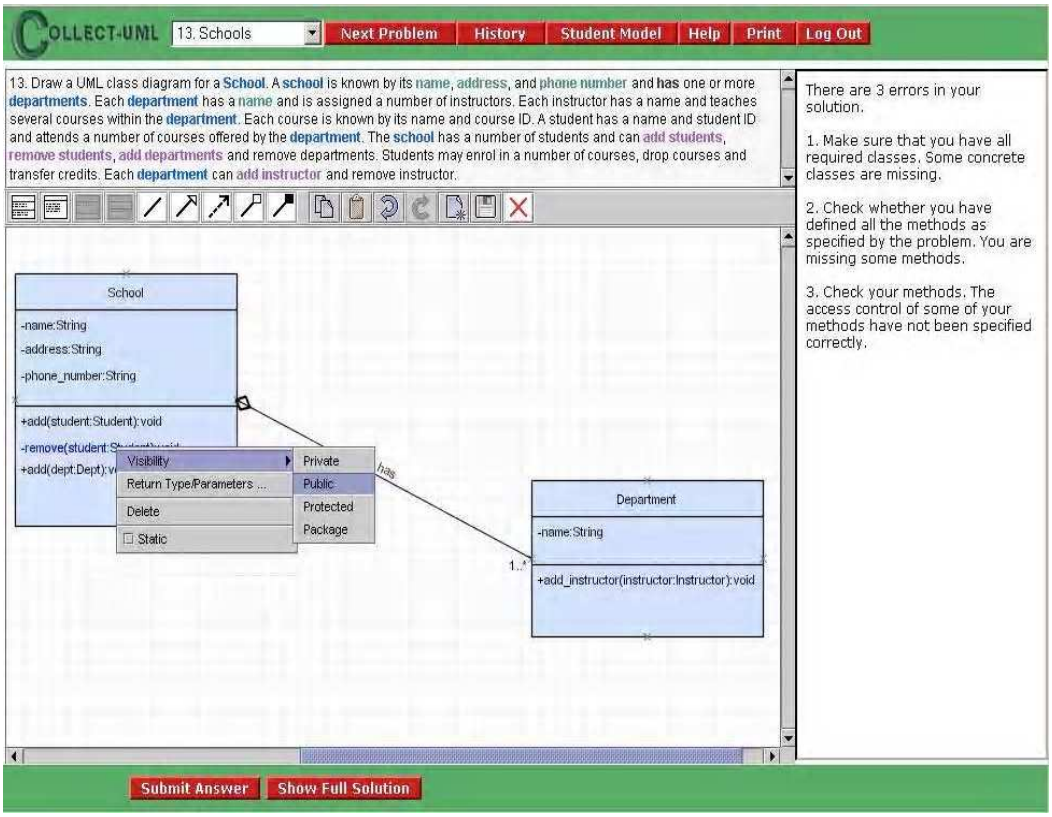



***Figure 4:*** *The current version of* COLLECT-$\mathcal{UML}$ *interface*

case was: "*Check your attributes. The types of some of your attributes have not been specified correctly*". Since the participant had defined several attributes, she was not sure which one the system was referring to. A modification was later made to the interface to highlight the errors in red.

For creating new attributes and methods, participants tended to use the tool bar icons more than right click menu. A few participants felt that the help document provided by the system was too long and some of them were not keen to refer to it, even when they needed help with something. Two participants also expressed their desire to have access to glossary and a tutorial on how to use the system. These features will be added to the system in the future.

In order to name a new component (class, attribute, method or relationship), the students were required to highlight (or double-click) the name from the problem text. Although some of the students found this somewhat restrictive, they became more comfortable with the interface once they had a chance to experiment with it. The students, who were more interested in typing in the names rather than highlighting the text, showed some interest after one of the researchers explained the reason behind restricting them.

The interface was modified to incorporate most of the suggestions mentioned above. The wording of one of the problems was changed after one participant commented that he found the problem text confusing.

The initial version of the interface was restricting the users in a way that they needed to first click on the *New attribute/method* toolbar button and then highlight the attribute/method name from the problem text. Some participants suggested that it would be more convenient if the system would allow them to create new attribute/methods by first letting them highlight its name from the problem text. The interface was then modified to incorporate both functionalities; i.e. the users can first highlight (or double-click) the attribute/method name from the problem text and then click on the *New Attribute/Method* toolbar button or vice versa.

It was observed during one of the sessions that one participant created a class, and chose the *New Method* menu option from the right-click pop-up menu. He was not sure what to do next, while the interface had disabled the toolbar and was expecting the user to highlight the name from the problem text. It was then decided to add some pop-up windows that would give information to novice users as to what they would need to do next. It displays the dialog window, each time the user wants to create a new class, relationship, attribute, or method (Figure 5). The system checks to see whether there are any attributes/methods specified already. If the user is about to create the first attribute/method for each class, the information window pops-up, otherwise, the system skips that level and prompts the user to highlight (or double-click) the name from the problem text, without showing the dialog information (as it is expected that the users would be familiar with what they would need to do then).

### *3.2 Students' impressions on feedback*

The majority of the participants felt that the feedback messages helped them to understand the domain concepts that they found difficult. For this study, we restricted the number of feedback messages up to 5 messages at a time.

The constraints were implemented so that they would only check for necessary constructs that the students were supposed to have included in their UML diagrams (i.e. classes, attributes, methods and relationships). Therefore, the participants were allowed to define extra methods for example, if they thought there were needed. This was a feature several participants particularly liked about the system.

## 4. Conclusions and Future Work

This paper presented COLLECT-UML, an ITS for UML modelling. An analysis carried out to investigate how participants interact with the single-user version of the system. Participants felt that using the system helped them improve their UML knowledge. Some of them experienced a number of difficulties interacting with the system. The video footage was useful in identifying the bugs in the system. All the bugs identified were fixed to make the system more robust.

A full evaluation study is planned for May 2005. The study aims to evaluate the interface and the effect of using the system on students' learning. It will involve second-year University students enrolled in an undergraduate Software Engineering course. The data recorded in the student model will be analyzed to see how much students learn during their interaction with the system.

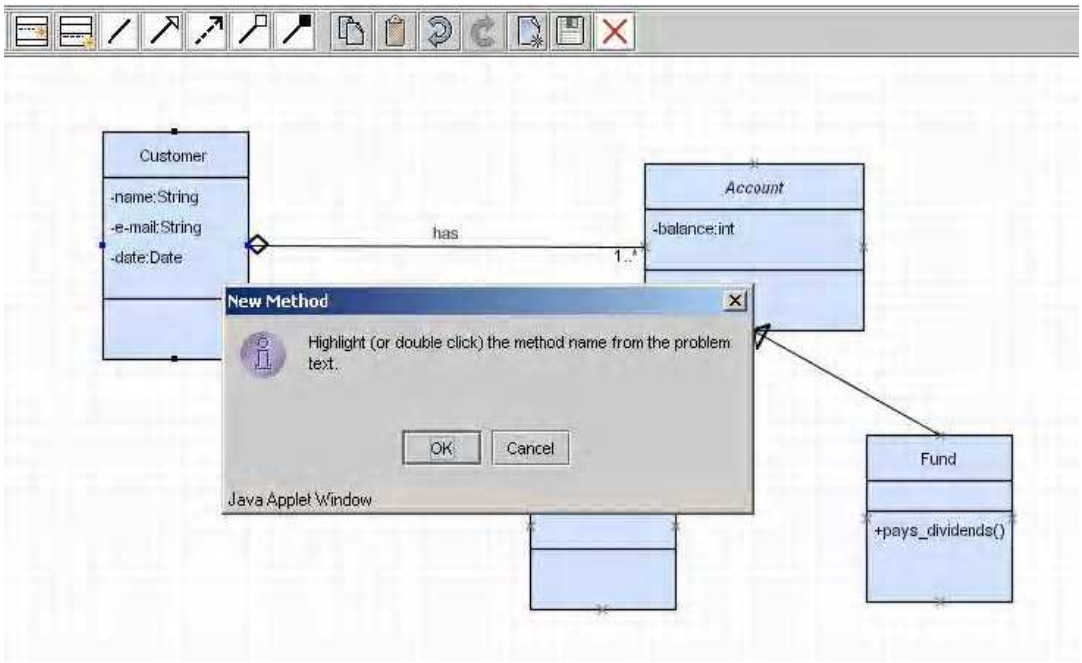



***Figure 5:*** *An information dialog popping up to guide the users as to what they would need to do next*

The most important goal of future work is to extend the system to support collaborative learning addressing both collaborative issues and task-oriented issues. CBM has been used to effectively present knowledge in several tutors supporting individual learning. The comprehensive evaluation studies of the multi-user version of the system will provide a measure of the effectiveness of using CBM technique in intelligent computer supported collaborative learning environments.

## References


[1] AllegroServe - a Web Application Server, http://www.franz.com/

[2] Ericsson, K. A. and Simon, H. A. (1984) Protocol Analysis: Verbal Reports as Data. *The MIT Press*, Cambridge, MA.

[3] Nielsen, J. (1993) Usability Engineering. *Academic Press Inc.*, San Diego, CA.

[4] Mayo, M., and Mitrovic, A. (2001) Optimising ITS behaviour with Bayesian networks and decision theory, *International Journal of Artificial Intelligence in Education*, 12 (2). pp.124-153.

[5] Mitrovic, A. (1998) Learning SQL with a Computerised Tutor. 29th *ACM SIGCSE Technical Symposium*, (Atlanta, 1998), pp.307-311.

[6] Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B. (2001) Constraint-based Tutors: a Success Story. *14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2001),* (Budapest, 2001), pp.931-940.

[7] Mitrovic, A. (2002) NORMIT, a Web-enabled tutor for database normalization. *ICCE 2002*, (Auckland, New Zealand), pp.1276-1280.

[8] Mitrovic, A. (2003) An intelligent SQl tutor on the Web. *International Journal of Artificial Intelligence in Education,* 13 (2-4). pp.173-197.

[9] Ohlsson, S. (1994) Constraint-based Student Modelling. *Student Modelling: the Key to Individualized Knowledge-based Instruction*, (Berlin, 1994), Springer-Verlag, pp.167-189.

[10] Soller, A. and Lesgold, A. (2000) Knowledge acquisition for adaptive collaborative learning environments. *AAAI Fall Symposium: Learning How to Do Things*, (Cape Cod, MA).

[11] Suraweera, P. and Mitrovic, A. (2004) An Intelligent Tutoring System for Entity Relationship Modelling. *International Journal of Artificial Intelligent in Education*, 14 (3-4). pp.375-417.

[12] Suraweera, P., and Mitrovic, A. (2002) KERMIT: a Constraint-based Tutor for Database Modeling. In: Cerri, S., Gouarderes, G. and Paraguacu, F. (eds.) *6th International Conference on Intelligent Tutoring Systems (ITS 2002)*, (Biarritz, France), pp.377-387.

# TICL Paper

The following paper [Baghaei, Mitrovic and Irwin, 2006] was published in the Technology, Instruction, Cognition and Learning (TICL) journal.

# Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams

Nilufar Baghaei, Antonija Mitrovic and Warwick Irwin
*Department of Computer Science and Software Engineering*
*University of Canterbury, Private Bag 4800, New Zealand*
*{n.baghaei, tanja, w.irwin}@cosc.canterbury.ac.nz*

**Abstract.** We present COLLECT-UML, a constraint-based tutoring system 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. Constraint-Based Modelling (CBM) has been used successfully in several tutoring systems, which have proven to be extremely effective in evaluations performed in real classrooms. In this paper, we present problem-solving support available in COLLECT-UML. The system observes students' actions and adapts to their knowledge and learning abilities. We describe the system's architecture and functionality. The effectiveness of the system has been evaluated in two studies with students taking ITS and software engineering courses. Objective data shows that students' performance increases significantly while interacting with the system, and that they do learn the domain concepts. The students have enjoyed the system's adaptivity and found it a valuable asset to their learning.

Keywords: problem-solving support, constraint-based modelling, UML class diagrams, ITS evaluation

## INTRODUCTION

Constraint-based tutors are Intelligent Tutoring Systems (ITS) which 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 & 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 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; Mitrovic & Ohlsson, 1999; Mitrovic, 2003], database modelling [Suraweera & Mitrovic, 2002; 2004], data normalization [Mitrovic 2002, 2005], punctuation [Mayo & Mitrovic, 2001] and English vocabulary [Martin & Mitrovic, 2003]. All three database tutors were developed as problem solving environments for tertiary students [Mitrovic et al., 2004]. Students solve problems presented to them with the assistance of feedback from the system. The tutors for punctuation and English vocabulary were developed for 9-12 year old school children.

This paper presents our experiences in implementing a constraint-based tutor in the area of object-oriented (OO) analysis and design using the Unified Modelling Language (UML). The chosen task is very complex, 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 modelling, 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.

Although many tutorials, textbooks and other resources on UML are available, we are not aware of any attempt at developing an ITS for UML modelling. However, there has been an attempt [Soller & Lesgold, 2000] at developing a collaborative learning environment for OO design problems using Object Modeling Technique (OMT) – a precursor of UML. The system 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 OO 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 which 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 determines when and why the students are having trouble learning the new concepts they share with each other. The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities. In this regard, even though the system is effective as a collaboration tool, it would probably not be an effective teaching system for a group of novices with the same level of expertise, as it could be common for a group of students to agree on the same flawed argument.

We start by describing the chosen instructional domain in Section 2. Section 3 describes the overall architecture of the system. COLLECT-UML supports problem-solving in two ways. The interface provides information about the domain of instruction, and its design is heavily influenced by the chosen domain. Section 4 discusses how the interface supports the learner while solving problems. Secondly, problem-solving is supported via the feedback that the system provides, which is discussed in Section 5. Section 6 presents the results of two evaluation studies performed. Conclusions are given in the last section.

## DIFFICULTIES OF LEARNING OBJECT-ORIENTED MODELLING

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. When OO first entered the mainstream of software engineering, it served (only) as a programming language paradigm. Subsequently, its influence broadened to provide a paradigm for the design of software, known as Object-Oriented Design (OOD), and broadened yet further to encompass Object-Oriented Analysis (OOA). In OO analysis, the same OO principles for structuring systems are used when performing requirements analysis to represent the concepts, behaviours and relationships found in some problem domain.

OO systems consist of *classes* (with *structure* and *behaviour)*, and *relationships* between them. Relationships have multiplicity, names and can be of different types (association, aggregation, composition, inheritance or dependency). In OOA and OOD, 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. Software engineering courses that teach OO analysis and design typically do so using UML.

UML consists of many types of diagrams, but *class diagrams* are the most fundamental for OO modelling, 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 modelling, with support for OO features such as inheritance and methods [Booch et al., 1999].

The research described in this paper 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 ITS described in this paper was designed to supplement the existing teaching programme by presenting additional problems and providing automated tutoring.

Let us illustrate the process of designing a class diagram on a simple example. A student is given the following description of the target system:

*Design a class diagram for a School. A school is known by its name, address, and phone number and has one or more departments. Each department has a name and is assigned a number of instructors. Each instructor has a name and teaches several courses within the department. Each course is known by its name and course ID. A student has a name and student ID and attends a number of courses offered by the department. The school has a number of students and can add students, remove students, add departments and remove departments. Students may enrol in a number of courses, drop courses and transfer credits. Each department can add instructors and remove instructors.*

From the description, the classes *school*, *department*, *student*, *course*, and *instructor* can be identified. The student may start by drawing these classes first. For each class, attributes and methods are described. For example, each department contains a name, and methods to add and remove instructors. All the attributes and methods are explicitly mentioned in the requirements.

The student also needs to identify the relationships between these classes. For example, each school has one or more departments, and this is mentioned in the second sentence of the problem text. The student needs to decide which relationship type would be most appropriate to use. Once all the relationship types are identified, the student needs to determine the multiplicities and names of the relationships.

The UML class diagram for the *School* software system is illustrated in Figure 1. As can be seen from this simple case, there are many things that the student has to know and think about when developing a UML diagram. The student must understand both the basic building blocks available and the restrictions specified on them. In real situations, the text of the problem is likely to be much longer, often ambiguous and incomplete. The student must be able to reason about the requirements and use his/her own world knowledge to make reasonable assumptions. UML modelling is not a well-defined process, and the task is open ended. There is no algorithm to derive the UML class diagram for a given set of requirements. There is no single, best solution for a problem, and often there are several correct solutions for the same requirements. In our experience students typically have many problems learning how to construct good quality OO models.

Although the traditional method of learning UML modelling in a classroom environment may be sufficient as an introduction to the concepts of OO analysis and design, students cannot gain expertise in the domain by attending lectures only. Even if some effort is made to offer students individual help through tutorials, a single tutor must cater for the needs of the entire group of students, and it is inevitable that they obtain only limited personal assistance. Therefore, the existence of a computerized tutor, which would support students in acquiring such design skills, would be highly useful.

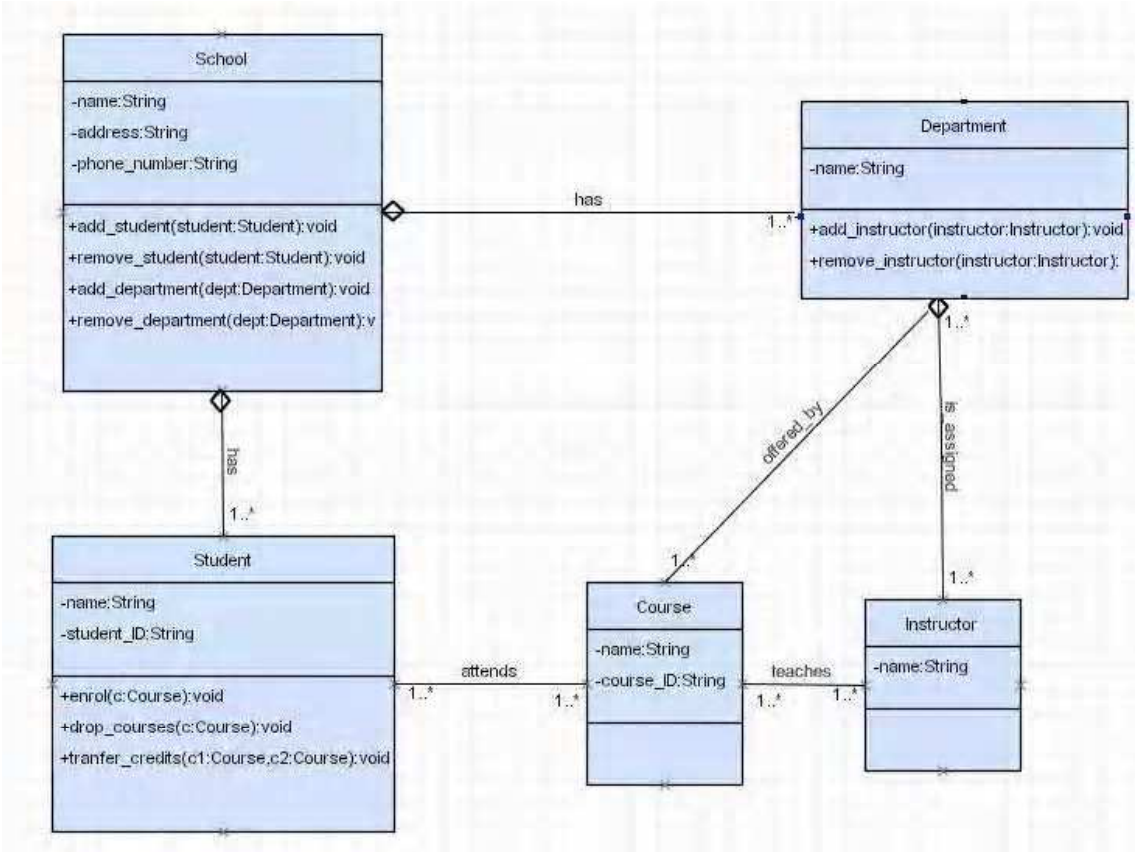



**Figure 1**. The UML Class diagram for *School*

## THE ARCHITECTURE OF COLLECT-UML

COLLECT-UML is a web-based problem-solving environment, in which students are required to construct UML class diagrams that satisfy a given set of requirements. 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 OO software design and UML. It assists students during problem solving and guides students towards a correct solution by providing feedback.

COLLECT-UML has a distributed architecture [Mitrovic, 2003], where the tutoring functionality is distributed between the client and the server. Because the task is very demanding and interactive, it was desirable to perform some pedagogical action on the client, in order to speed up interaction. The client intervenes in situations when the student makes simple syntax errors, such as submitting a diagram with missing component names. The system is implemented in WETAS [Martin & Mitrovic, 2002; 2003], a constraint-based authoring shell. WETAS itself is implemented in Allegro Common Lisp, which provides a development environment with an integrated Web Server [AllegroServe].

The system's components are illustrated in Figure 2. 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 modeller 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 modeller diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module, which generates appropriate feedback.

COLLECT-UML does not have a problem solver, as developing a general problem solver for UML modelling is extremely difficult. One of the major obstacles that would have to be overcome is natural language processing (NLP), as the problems in the domain are presented using natural language text. However, the NLP problem is far from being solved. Other complexities arise from the nature of the task. There are assumptions that need to be made during the development of UML diagrams. These assumptions are outside the problem description and are dependent on the semantics of the problem itself. Although this obstacle can be avoided by explicitly specifying these assumptions within the problem description, ascertaining these assumptions is an essential part of the process of constructing a solution and would over-simplify the problems.

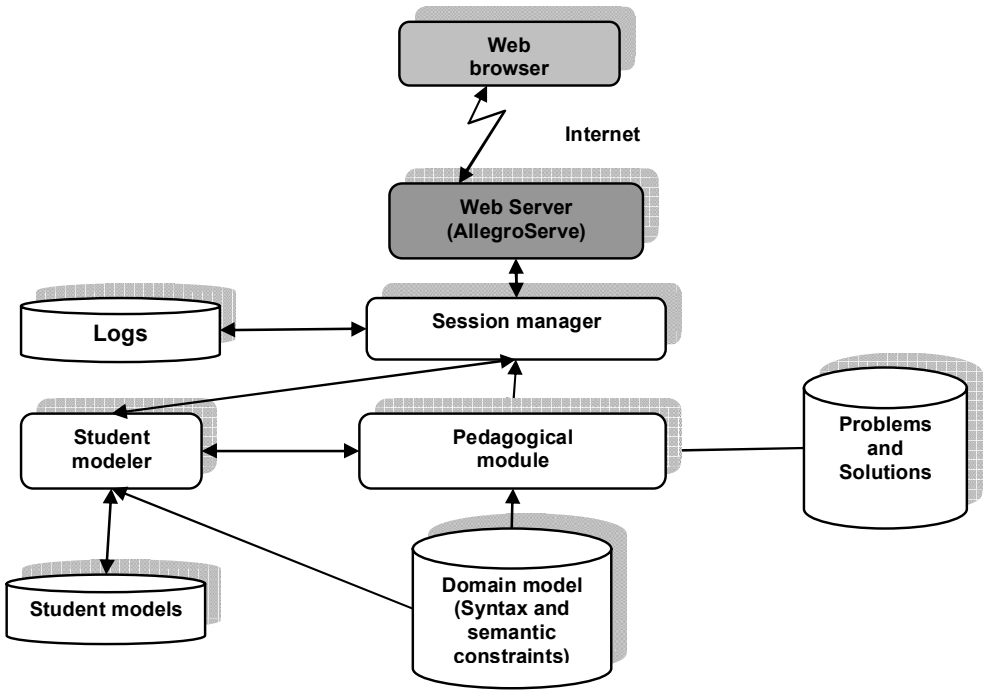



**Figure 2**. The architecture of the system

Although there is no problem solver, COLLECT-UML is capable of diagnosing students' solutions. The system contains an ideal solution for each problem, which is compared to the student's solution according to the system's domain model, represented as a set of constraints. Constraint-Based Modeling [Ohlsson 1994] is a student modeling approach that is not interested in the exact sequence of states in the problem space the student has traversed, but in what state he/she is in currently. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. The domain model is a collection of state descriptions of the form: *If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong.* If the relevance condition of a constraint is true (i.e. constraint is relevant to the student's solution being processed), the satisfaction condition should also be true. Otherwise the constraint is violated and the feedback message attached to that constraint is presented to the student.

The system's domain model contains 88 semantic and 45 syntax constraints that describe the basic principles of the domain. Semantic constraints are usually more complex than syntax constraints. In order to develop constraints, we studied material in textbooks, such as [Fowler, 2004], and also used our own experience in teaching UML and OO analysis and design. Figure 3 illustrates two constraints from the UML domain. Constraint 41 is a syntax constraint; it checks that there are some attributes or methods defined for each class in the student's solution (SS). The constraint contains a message which 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 classes, and also identifies the class to which the constraint was applied. Constraint 52 is a semantic constraint. Its relevance condition identifies a superclass and a subclass in the ideal solution, which have the same method defined. Then, the relevance condition looks for a matching class and a superclass in the student's solution, with the same method defined for the superclass. The student's solution is correct if there is a method with the same name defined in the subclass, which overrides the method defined in the superclass.

```
(41
 "Check your classes. Each class must have at least one attribute or method."
 ; Relevance condition
 (match SS CLASSES (?* "@" ?class_tag ?*))
 ; Satisfaction condition
 (or-p (match SS ATTRIBUTES (?* "@" ?tag1 ?attr_name ?c1ass_tag ?*))
       (match SS METHODS (?* "@" ?tag2 ?method_name ?class_tag ?*)))
 "classes"
 (?class_tag))

(52
 "Check your inheritance relationships. Some of your subclasses must override one or more methods
defined in the superclass. The ability of a subclass to override a method in its superclass allows a class to
inherit from a superclass whose behavior is similar, and then override methods as needed."
; Relevance condition
  (and (match IS SUPERCLASSES (?* "@" ?c1_tag ?*))
       (match IS SUBCLASSES (?* "@" ?c2_tag ?c1_tag ?*))
       (match IS METHODS (?* "@" ?m1_tag ?name ?c1_tag ?*))
       (match IS METHODS (?* "@" ?m1_tag ?name2 ?c2_tag ?*))
       (match SS SUPERCLASSES (?* "@" ?c1_tag ?*))
       (match SS SUBCLASSES (?* "@" ?c2_tag ?c1_tag ?*))
       (not-p (test SS ("null" ?c1_tag)))
       (not-p (test SS ("null" ?c2_tag)))
       (match SS METHODS (?* "@" ?m1_tag ?name3 ?c1_tag ?*)))
; Satisfaction condition
   (match SS METHODS (?* "@" ?m1_tag ?name4 ?c2_tag ?*))
   "methods"
   (?c1_tag ?c2_tag ?m1_tag))
```

**Figure 3**. Examples of constraints from COLLECT-UML

The short-term student model consists of a list of violated and a list of satisfied constraints for the current attempt. The long-term model records the history of usage for each constraint. This information is used to select problems of appropriate complexity for the student, and generate feedback.

## INTERFACE

Students interact with COLLECT-UML via its interface (Figure 4) 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 (Figure 5), 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. The applet was implemented using Java 1.4.2, and contains 4 packages, 75 Java classes and 6853 lines of code. Feedback is presented on the right, while the bottom part allows the student to select the feedback level, and submit solutions.

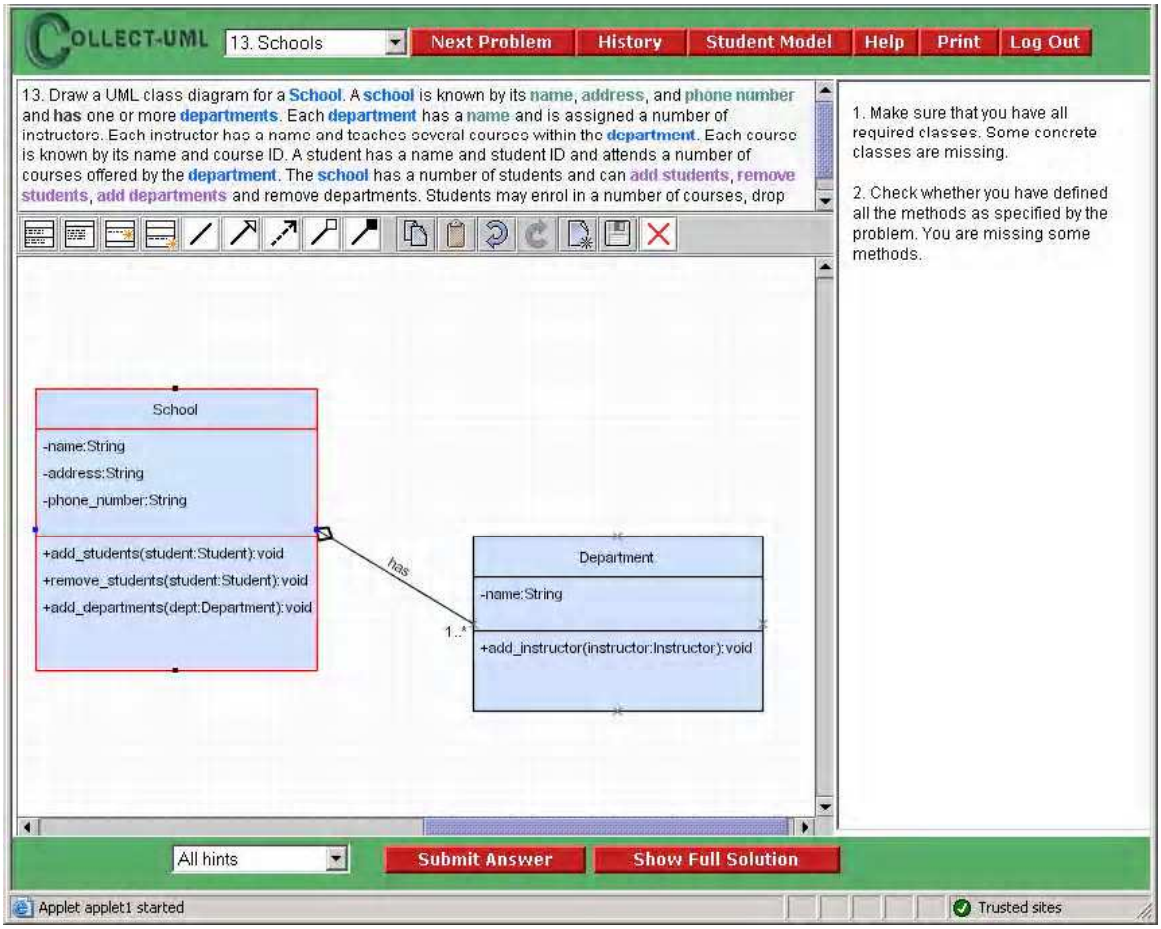



**Figure 4.** The interface of *COLLECT-UML*

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 can be seen from Figure 4, the applet contains a drawing bar with UML constructs. Students can therefore remind themselves of the basic building blocks to use when drawing UML diagrams. The symbols used for UML modelling are shown in Figure 6. 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 requires the student to name each newly added construct by using a word/phrase from the problem text as its name. A name can be selected by highlighting a phrase from the problem text. It is not possible to name a construct by typing. This is useful from the point of view of the student modeller for evaluating solutions [Suraweera & Mitrovic, 2002]. There is no standard that is enforced in naming classes, methods, attributes or relationships. Since the names of the components in the student solution may not match the names of construct in the ideal solution (IS), the task of finding correspondence

between the constructs of the SS and IS is difficult. This problem is avoided by forcing the student to use the names that come from the problem text directly.

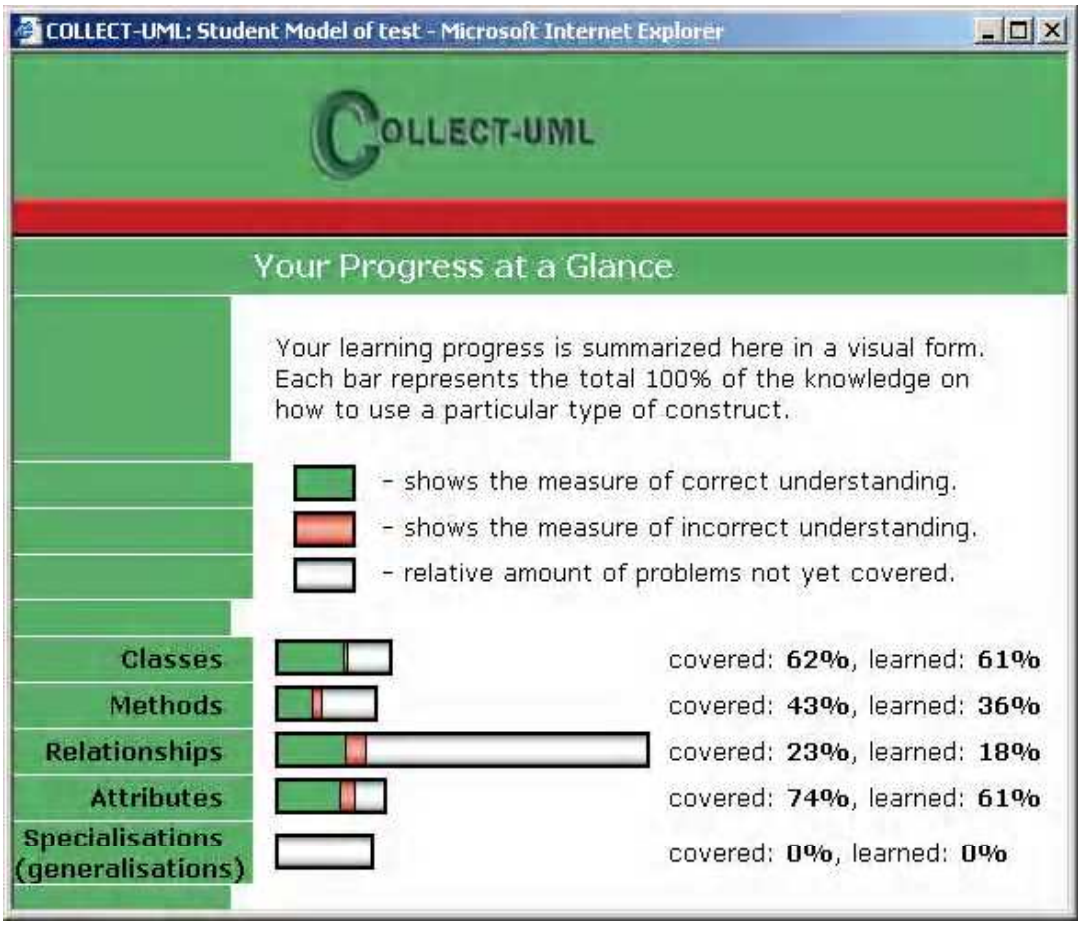



**Figure 5.** The open student model

This requirement enforces two of the most important practices in software design: using the end-users' language and reflecting on the requirements. By selecting names for various diagram components directly from the problem text, the student has to think about the requirements. The interface highlights the previously selected parts of the problem text that correspond to various types of UML constructs using different colours, making it easier for the student to review how much of the problem has been covered. Subjective evaluation of the system (described later in the paper) showed that several participants pointed out this feature, when asked what they liked in particular about COLLECT-UML.

Currently, the system contains 14 problems, which cover different aspects of UML modeling, and their ideal solutions. Figure 7 shows a sample problem and the internal representation of its ideal solution, which consists of 6 components (i.e. *RELATIONSHIPS, ATTRIBUTES, METHODS, CLASSES, SUPERCLASSES* and *SUBCLASSES*). The problem text is represented internally with embedded tags that specify the mapping to the constructs in the ideal solution. The tags are not visible to the student since they are extracted before the problem is displayed.

| Symbol | Component |
|---|---|
| | Concrete Class |
|  | Interface |
|  | Attribute |
|  | Method |
|  | Association |
|  | Inheritance |
|  | Dependency |
|  | Aggregation |
|  | Composition |

**Figure 6.** UML components supported by the system

The applet saves the solutions submitted by students as XML files, which are converted to internal representation using an XSLT style-sheet. The constraints are applied to the internal representation of the solutions and feedback is given to students, using messages attached to the violated constraints.

```
(13                ; problem number
10                 ; difficulty
"Draw a UML class diagram for a <E1> School </E1>. A <E1> school </E1> is known by its <E1A1>
name </E1A1>, <E1A2> address </E1A2>, and <E1A3> phone number </E1A3> and <R1> has </R1>
one or more <E2> departments </E2>. Each <E2> department </E2> has a <E2A1> name </E2A1> and
<R2> is assigned </R2> a number of <E3> instructors </E3>. Each <E3> instructor </E3> has a <E3A1>
name </E3A1> and <R3> teaches </R3> several <E4> courses </E4> within the <E2> department </E2>.
Each <E4> course </E4> is known by its <E4A1> name </E4A1> and <E4A2> course ID </E4A2>. A
<E5> student </E5> has a ..."

(("RELATIONSHIPS" "@ R1 aggregation E1 E2 null 1..* null null has @ R2
aggregation E2 E3 null 1..* null null is_assigned @ R3 association E4
E3 1..* 1..* null null teaches ...")
("ATTRIBUTES" "@ E1A1 name E1 String private no @ E1A2 address E1 String
private no @ E1A3 phone_number E1 String private no @ E3A1 name E3
String private no @ E4A1 name E4 String private no @ E4A2 course_ID E4
String private no @ E5A1 name E5 String private no ...")
("METHODS" "@ E1A4 add_student E1 void public no 1 student_ID String null
null null null @ E1A5 emove_student E1 void public no 1 Student_iD
String null null null null ...")
("CLASSES" "@ E1 School concrete @ E3 Instructor concrete @ E4 Course
concrete @ E5 Student concrete @ E2 Department concrete ")
("SUPERCLASSES" "")
("SUBCLASSES" ""))
"13.jpg"
"Schools")
```

**Figure 7.** A sample problem and its ideal solution

## FEEDBACK GENERATION

COLLECT-UML evaluates the student's solution once it is submitted, and provides feedback. During evaluation, the student modeller identifies the constraints that the student has violated. The feedback is offered at five levels of detail: *Simple Feedback*, *Error flag*, *Hint*, *All Hints* and *Full solution*. 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 such as "*Make sure that you have all required classes. Some concrete classes are missing.*" A list of feedback messages on all violated constraints is displayed at the *All hints level*. The UML class diagram of the complete solution is displayed when the user clicks 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 incremented with each submission until the feedback level reaches the *Hint* level. In other words, if the student 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 levels 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.

## EVALUATION

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

### Pilot Study

The pilot study was conducted as a think-aloud protocol in March 2005. The study aimed to discover students' perceptions about various aspects of the system, mainly the quality of feedback messages and the usability of the interface. The participants were 12 postgraduate students enrolled in an Intelligent Tutoring Systems course at the University of Canterbury. At the time of the study, the participants had completed 50% of the ITS course lectures, and were expected to have a good understanding of ITS. All participants except two were already familiar with UML modelling.

The study was carried out in the form of a think-aloud protocol [Ericsson & 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 verbalise his/her thoughts while performing a UML modelling task using COLLECT-UML. Participants were able to skip the problems without completing them and to return to previous problems. 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 the drawing area was big enough for them to work on the problems given. Three participants felt that some of the hints provided by the system were not helpful enough for them to correct their mistakes. The difficulty with the feedback came from the students not being able to interpret given messages. For example, the feedback message of constraint 41 (Figure 3) is "*Check your classes. Each class must have at least one attribute or method*." If the diagram contains many classes, the student might have difficulty identifying the class that the feedback message is relevant for. We have modified the system to highlight the part of the diagram related to the feedback message in red, making it easy for students to localize errors. Two participants also expressed their desire to have access to a glossary and a tutorial on how to use the system. These features will be added to the system in the future.

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. Pop-up dialog windows were added to help the users with naming the classes/methods/attributes once they were created.

The majority of the participants felt that the feedback messages helped them to understand the domain concepts that they found difficult. For this study, the feedback level was restricted to *All Hints* only. For the full evaluation study (described in the next section), the system was modified to include the five different levels of feedback, shown in Figure 4.

The constraints were implemented so that they would only check for necessary constructs that the students were supposed to have included in their UML diagrams (i.e. classes, attributes, methods and relationships). Therefore, the participants were allowed to define extra methods for example, if they thought they were needed. This was a feature several participants particularly liked about the system.

## Evaluation Study

The evaluation study was carried out at the University of Canterbury in May 2005, after COLLECT-UML was enhanced in the light of the findings from the pilot study. The study involved 38 volunteers from students enrolled in the Introduction to Software Engineering course offered by the Computer Science and Software Engineering department. This second year course teaches UML modelling as outlined by Fowler [2004]. The students learnt UML modelling concepts during two weeks of lectures and had some practice during two weeks of tutorials prior to the study.

The study was conducted in two streams of two-hour laboratory sessions. Each participant sat a pre-test, interacted with the system, and then sat a post-test and filled a user questionnaire. The pre-test and post-test (given in Appendices A and B) each contained four multiple-choice questions, followed by a question where the students were asked to design a simple UML class diagram. Both tests included questions of comparable difficulty, dealing with inheritance and association relationships.

Table 1 presents some general statistics about the study. The participants spent two hours interacting with the system, and solved half of the problems they attempted.

**Table 1:** Some statistics about the study

| | Average | s. d. |
|---|---|---|
| Time spent on problem solving (hours) | 1.52 | 0.43 |
| Attempted problems | 5.71 | 2.59 |
| Solved problems | 47% | 33% |
| Attempts per problem | 7.42 | 4.76 |
| Pre-test | 52% | 21% |
| Post-test | 76% | 17% |

*Learning*

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% (Table 1). The students' performance on the post-test was significantly better ($t = 2.71$, $p = 4.33E\text{-}08$).

We have also analyzed the log files, in order to identify how students learn the underlying domain concepts. Figure 8 illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all constraints and all participants (*All constraints*). The data points show a regular decrease, which is approximated by a power curve with a close fit of 0.93, thus showing that students do learn constraints over time. The probability of 0.19 for violating a constraint on the first occasion of application has decreased to 0.09 at its tenth occasion, displaying a 47% decrease in probability.

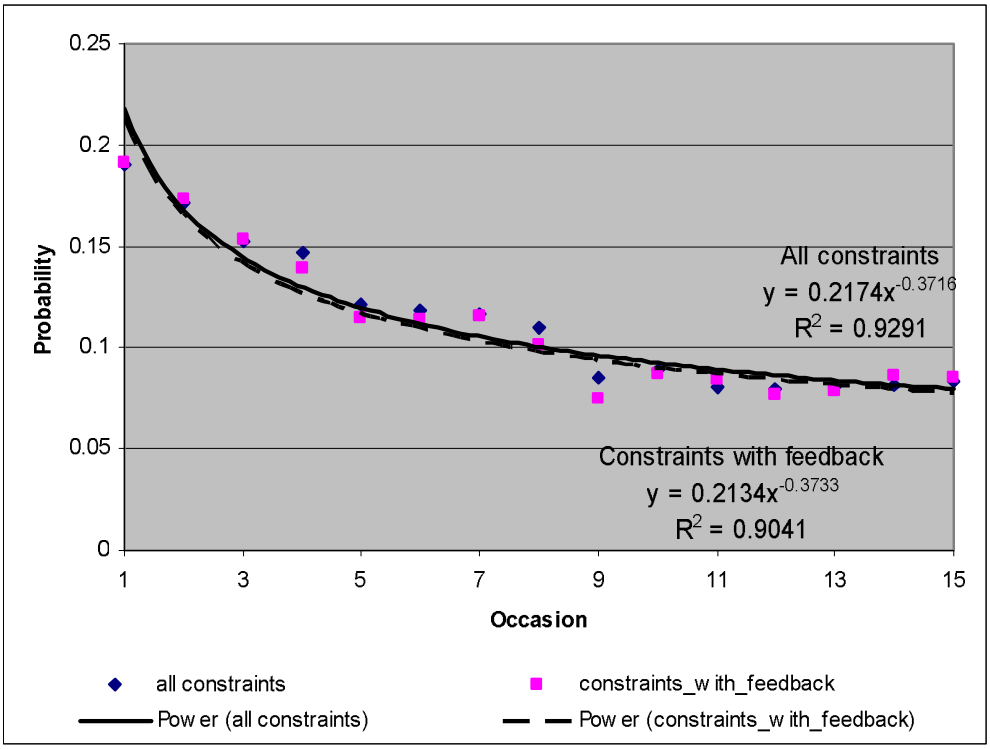



**Figure 8.** Probability of constraint violation

The other power line in Figure 8 labelled *Constraint with feedback* illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all participants, but only for constraints on which participants obtained specific feedback (i.e. when the participants asked for *Hint* or *All Hints* feedback levels). The student logs show that 53% of the participants asked for the *All Hints* feedback level. The *All constraints* learning curve has 2053 data points at the first occasion, while the *Constraints with feedback* curve has 40% less, because students often received feedback other than hints. The learning curve is again very regular, with a very high $R^2$ fit (0.9), and almost identical initial error probability (0.19) and learning rate (-0.37). We believe that the difference between these two curves is small because the participants often could recover from their errors by being shown where the error is (i.e. by being given the *Error Flag* feedback), or could correct slips by being told that there are problems in their solutions (*Simple feedback*).

We found out that 22 constraints 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 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 checks the correct use of dependencies between classes; 5) constraints that check inheritances in students' diagrams, making sure that there are no cycles, and finally 6) a constraint that makes sure each subclass is connected to a superclass.

There were also five constraints that were never satisfied, meaning that the participants did not learn the corresponding domain concepts during the session. The constraints in this group cover aggregation and composition, making sure that the student has used the correct UML construct to represent them. Also this group includes a constraint that checks that multiple inheritance is only specified for interfaces.

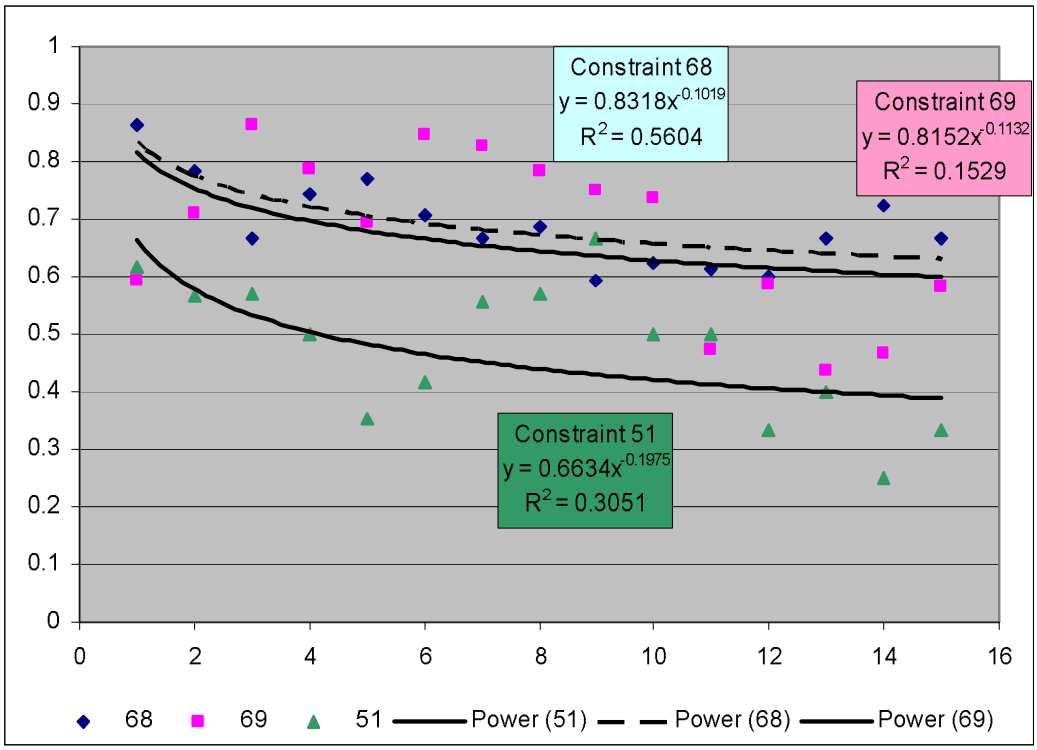



**Figure 9.** Learning curves for three difficult constraints

We have also looked at learning curves of individual constraints, trying to identify constraints that were especially difficult for our students. Figure 9 illustrates the learning curves for three constraints which were hard for participants. The learning rates for all three constraints are much lower that the ones in Figure 8, as well as the $R^2$ fit. Constraint 68, the most difficult of the three, checks whether the participant has specified the types of attributes correctly. Constraint 69, the second hardest, checks whether static attributes were specified as such. Finally, constraint 51 checks whether the correct parameters have been specified for methods. In all three 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 predominantly correct. Furthermore, we have noticed that some problem texts do not contain enough detail for the student to be able to complete the relevant parts of the solution, and therefore we believe that the probability of violating these constraints could be decreased by including more detail in the problem descriptions.

*Subjective analysis*

All the participants were given a questionnaire (Appendix C) at the end of their session to determine their perceptions of the system. Table 2 presents a summary of the responses. The students found the interface easy to learn and use. 60% of the participants were familiar with UML modelling from lectures and some work, and the rest had previous experience only from the lectures. Most of the participants (65%) responded they would recommend the system to other students.

The mean response when asked to rate how much they learnt by interacting with COLLECT-UML was 2.9, on the scale of 1 (nothing) to 5 (very much). As Table 1 shows, the students spent 1.52 hours on problem solving in average. Some participants indicated that they would have learnt a lot, if they had more time to interact with the system.

Students were offered individualised feedback on their solutions upon submission. The mean rating for the usefulness of feedback was 2.8. 67% of the participants had indicated that they would have liked to see more details in the feedback messages, especially the ones dealing with types of attributes and number of parameters for each method. These two common remarks point out that the problem texts do not contain enough information for students to make correct decisions related to these issues, as we have already noted from the analysis of individual constraints' learning curves. The problem texts will be modified in future, in order to provide such information. The comments we received on open questions also pointed out several features of the system, which can be improved.

**Table 2:** Mean responses from the user questionnaire for the evaluation study

| | **Average** | **s. d.** |
|---|---|---|
| Time to learn interface (min.) | 10 | 8 |
| Amount learnt | 2.9 | 0.9 |
| Enjoyment | 2.9 | 1 |
| Ease of using interface | 2.8 | 1 |
| Usefulness of feedback | 2.8 | 1 |

*Discussion*

The results show COLLECT-UML is an effective learning environment. The participants achieved significantly higher scores on the post-test, suggesting that they acquired more knowledge in UML modelling. The learning curves also prove that students do learn

constraints during problem solving. Subjective evaluation shows that most of the students felt spending more time with the system would have resulted in more learning 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 such as including short cut keys, including more details in some of the feedback messages and tool tip boxes, providing tutorials on how to use the system and including general explanations of the full solutions, when they are being displayed to the user.

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* in their personal time for practicing UML modelling.

## CONCLUSIONS

This paper discussed the design and implementation of COLLECT-*UML,* an ITS developed to assist students learning UML modeling. We presented the system's architecture and functionality, with emphasis on problem-solving support. COLLECT-*UML* supports problem solving through its interface, which provide domain-specific information and enforces good practices in the domain. The system also provides feedback on students' solutions. COLLECT-*UML*'s effectiveness in teaching UML class diagrams was evaluated in the two classroom experiments. The results of both subjective and objective analysis proved that COLLECT-*UML* is an effective educational tool. The participants performed significantly better on a post-test after short sessions with the system, and reported that the system was relatively easy to use. The reported studies evaluated the system as a whole; in the future studies, we will focus on a single feature of the system, such as feedback or adaptation.

The goal of future work is to extend the system to support collaborative learning, addressing both collaborative issues and task-oriented issues. The enhancement process will include implementation of the shared workspace, modification of the pedagogical module to support groups of users and designing and implementing the group-modeling component, which will generate feedback messages related to effective collaboration. CBM has been used to effectively present knowledge in several ITSs supporting individual learning. The comprehensive evaluation studies of the multi-user version of the system will provide a measure of the effectiveness of using the CBM technique in intelligent computer-supported collaborative learning environments.

## ACKNOWLEDGEMENTS

The work presented here was supported by the University of Canterbury PhD scholarship awarded to the first author. We thank Konstantin Zakharov for helping with the statistical analyses and Pramudi Suraweera for advice during the earlier stages of constraint development. This research could not have been done without the support of other past and present members of ICTG.

## REFERENCES


AllegroServe - a Web Application Server. Retrieved 31.5.2005 from http://www.franz.com/

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.

Ericsson, K. A., Simon, H. A. (1984) *Protocol Analysis: Verbal Reports as Data*. Cambridge: MIT Press.

Fowler, M. (2004) *UML Distilled: a Brief Guide to the Standard Object Modelling Language*. Reading: Addison-Wesley, 3rd edition.

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) *Proc. Int. Conf. Computers in Education*, pp. 183-187.

Martin, B., Mitrovic, A. (2003) Domain Modeling: Art or Science? In: U. Hoppe, F. Verdejo & J. Kay (ed) *Proc. 11th Int. Conference on Artificial Intelligence in Education*, IOS Press, 183-190.

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

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. *Proc. ICCE 2002*, pp.1276-1280.

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*, IOS Press, pp. 499-506.

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., Mayo, M., Suraweera, P., Martin, B. (2001) Constraint-based Tutors: a Success Story. *Proc. 14th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Berlin: Springer-Verlag LNAI 2070, pp.931-940.

Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A. (2004) DB-suite: Experiences with Three Intelligent, Web-based Database Tutors. *Journal of Interactive Learning Research*, 15(4), 409-432.

Nielsen, J. (1993) *Usability Engineering*. San Diego, CA: Academic Press Inc.

Ohlsson, S. (1994) Constraint-based Student Modelling. In: J. Greer and G. McCalla (eds) *Student Modelling: the Key to Individualized Knowledge-based Instruction*, Berlin: Springer-Verlag, pp.167-189.

Soller, A., Lesgold, A. (2000) Knowledge Acquisition for Adaptive Collaborative Learning Environments. *AAAI Fall Symposium: Learning How to Do Things*.

Sommerville, I. (2004) *Software Engineering*. Pearson/Addison-Wesley, 7th ed.

Suraweera, P., Mitrovic, A. (2002) KERMIT: a Constraint-based Tutor for Database Modeling. In: Cerri, S., Gouarderes, G. and Paraguacu, F. (eds.) *Proc. 6th Int. Conf. Intelligent Tutoring Systems*, 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.

## APPENDIX A: Pre-Test

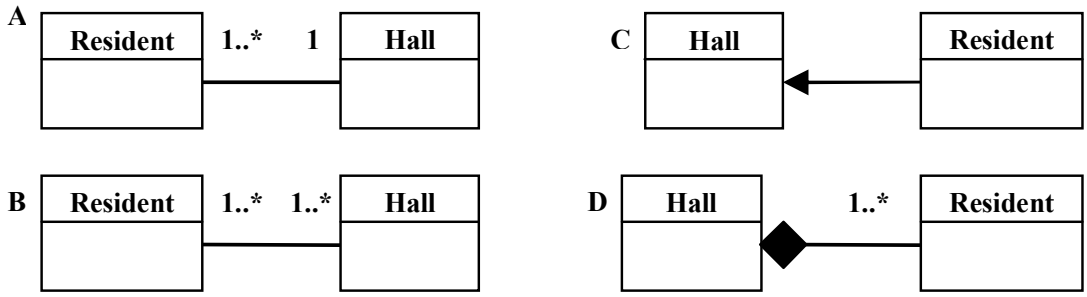
**1.** System analysts often examine textual requirements descriptions for domain model information. *Nouns* suggest:

**A.** *Classes* **D.** *Relationships*
**B.** *Attributes* **E**. *A and B*
**C.** *Methods* **F.** *C and D*

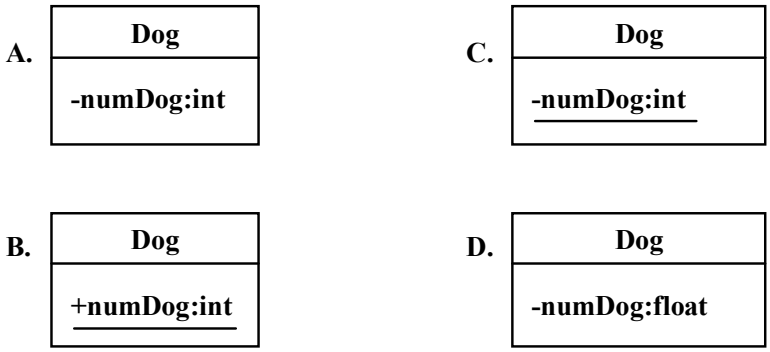
**2.** Which type of UML relationship would be used when one object merely *invokes methods* of another object?

**A.** *Inheritance* **D.** *Aggregation*
**B.** *Dependency* **E.** *All of the above*
**C.** *Association* **F.** *None of the above*

**3.** Select the most appropriate option that best describes the given situation: "**Residents live in a student hall**"




**4.** Which diagram best describes "**numDogs**" attribute? **numDogs** contains a count of the number of Dog instances. This count is accessed only within the class Dog.




**5.** Draw a UML class diagram to represent order payments. An order has a number and a price. There are two payment options: credit card and cheque. For each payment option, we store the payment date. For credit card option, the card number and the expiry date are recorded. For cheque option, the cheque number is stored.

## APPENDIX B: Post-Test

**1.** System analysts often examine textual requirements descriptions for domain model information. *Verbs* suggest:

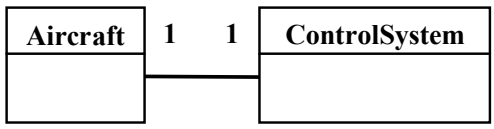
**A.** *Classes*
**B.** *Attributes*
**C.** *Methods*
**D.** *Relationships*
**E**. *All of the above*
**F.** *None of the above*

**2.** Which type of UML relationship cannot have a relationship name?

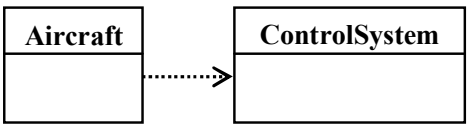
**A.** *Composition*
**B.** *Association*
**C.** *Inheritance*
**D.** *Aggregation*
**E.** *All of the above*
**F.** *None of the above*

**3.** Select the most appropriate option that best describes the given situation: "**an aircraft has a control system.**"

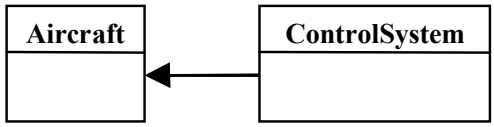
**A**



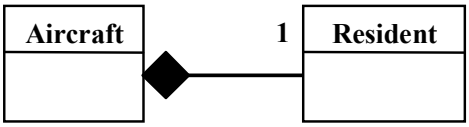

**C**




**B**




**D**




**4.** In object-oriented software, attributes usually have a visibility of ..... and methods have a visibility of .....

**A.** Public, Protected
**B.** Private, Protected
**C.** Private, Public
**D.** Public, Private

**5.** Draw a UML class diagram to represent customers. A customer has a name and an address and places one or more orders. Each order has a number and a date it was received. A customer can be either personal or corporate. For personal customers, the credit card number is recorded and for corporate customers, the credit card rating and limit are stored.

## APPENDIX C: Questionnaire

Thank you for using COLLECT-*UML*. Your feedback will be crucial for further improvements of the system and we would be most grateful, if you could take time to fill in this questionnaire. The questionnaire is anonymous, and you will not be identified as an informant. You may at any time withdraw your participation, including withdrawal of any information you have provided. By completing this questionnaire, however, it will be understood that you have consented to participate in the project and that you consent to publication of the results of the project with the understanding that anonymity will be preserved**.**

1. What is your previous experience with UML modelling? (Please circle one)
   A - Only lectures  B – Lectures plus some work  C – Extensive use

2. How much time did you need to learn about the system's functions? (Please circle one)

| | |
|---|---|
| (a) | Substantial time (most of the session) |
| (b) | 30 minutes |
| (c) | 10 minutes |
| (d) | Less than 5 minutes |

3. How much did you learn about UML modelling from using the system? (Please circle one)

| Nothing | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

   Please comment.

4. Did you enjoy learning with COLLECT-*UML*? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

5. Would you recommend COLLECT-*UML* to other students? (Please circle one)
   A – No  B- Don't know  C - Yes

6. Did you find the interface easy to use? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

7. Did you find the feedback from COLLECT-*UML* useful? (Please circle one and add a comment)

| Not at all | | | | Very much |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

8. Would you prefer more details in feedback? (Please circle one and comment)
   A – No  B- Don't know  C - Yes

9. Did you encounter any software problems or system crashes? If yes, please specify
10. What did you like in particular about COLLECT-*UML*?
11. Is there anything you found frustrating about the system?
12. Do you have any suggestions for improving COLLECT-*UML*?

# ITS 2006 Paper

The following paper [Baghaei and Mitrovic, 2006] was published and presented at the ITS 2006 conference in Jhongli, Taiwan.

# A Constraint-based Collaborative Environment for Learning UML Class Diagrams

**Nilufar BAGHAEI and Antonija MITROVIC**

*Intelligent Computer Tutoring Group*
*Department of Computer Science and Software Engineering*
*University of Canterbury, Private Bag 4800, New Zealand*
*{n.baghaei, tanja}@cosc.canterbury.ac.nz*

**Abstract.** COLLECT-$\mathcal{UML}$ is a constraint-based ITS that teaches object-oriented design using Unified Modelling Language (UML). UML is easily the most popular object-oriented modelling technology in current practice. 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 show that students' performance increased significantly. In this paper, we present our experiences in extending the system to provide support for collaboration. We present the architecture, interface and support for collaboration in the new, multi-user system. A full evaluation study has been planned, the goal of which is to evaluate the effect of using the system on students' learning and collaboration.

## 1. Introduction

E-learning is becoming an increasingly popular educational paradigm as more individuals who are working or are geographically isolated seek higher education. As such students do not meet face to face with their peers and teachers, the support for collaboration becomes extremely important [8]. Effective collaborative learning includes both learning to effectively collaborate, and collaborate effectively to learn, and therefore a collaborative system must be able to address collaboration issues as well as task-oriented issues [17].

In the last decade, many researchers have contributed to the development of CSCL and advantages of collaborative learning over individualised learning have been identified [14]. Some particular benefits of collaborative problem-solving include: encouraging students to verbalise 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 articulate their reasoning, and elaborate and reflect upon their knowledge [24, 27]. These benefits, however, are only achieved by active and well-functioning learning teams [15]. Numerous systems for collaborative learning have been developed; however, the concept of supporting peer-to-peer interaction in CSCL systems is still in its infancy. Various strategies for computationally supporting online collaborative learning have

been proposed and used, while more studies are needed that test the utility of these techniques [17].

This paper describes an Intelligent Tutoring System (ITS) that uses Constraint-Based Modeling (CBM) approach to support both problem-solving and collaborative learning. CBM has been used successfully in several tutors supporting individual learning [20]. We have developed COLLECT-UML [2, 3], a single-user version of a constraint-based ITS, that teaches UML class diagrams. In this paper, we describe extensions to this tutor, which support multiple students solving problems collaboratively. We start with a brief overview of related work in Section 2. Section 3 then presents COLLECT-UML and the evaluation study conducted with second-year university students taking a course in Introduction to Software Engineering. Section 4 describes the design and implementation of the collaborative interface as well as the system's architecture. Section 5 presents the collaborative model, which has been implemented as a set of meta-constraints. Conclusions are given in the last section.

## 2. Related Work

Three categories of CSCL systems can be distinguished in the context of the collaboration support [1, 17]. The first category includes systems that reflect actions; the basic level of support a system may offer involves making the students aware of the participants' actions. 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. Sharlock II [21]), while others internally compare the current state of interaction to a model of ideal interaction, but do not reveal this information to the users (e.g. EPSILON [25]). In the latter case, this information is either intended to be used later by a coaching agent, or analysed by researchers in order to understand the interaction [17]. Finally, the third class of systems offer advice 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 distinguished by the nature of the information in their models, and whether they provide feedback on strictly collaboration issues or both social and task-oriented issues. Examples of the systems focusing on the social aspects include Group Leader Tutor [19] and DEGREE [6], and an example of the systems addressing both social and task-oriented aspects of group learning is COLER [7].

Although many tutorials, textbooks and other resources on UML are available, we are not aware of any attempt at developing a CSCL environment for UML modelling. However, there has been an attempt [25] at developing a collaborative learning environment for OO design problems using Object Modeling Technique (OMT) – a precursor of UML. The system 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 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 which 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 determines when and why the students are having trouble learning new concepts they share with each other. The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities. In this regard, even though the system is effective as a collaboration tool, it would probably not be an effective teaching system for a group of novices with the same level of expertise, as it could be common for a group of students to agree on the same flawed argument.

CBM has been used successfully in several tutors supporting individual learning. The main contribution of this research is the use of CBM technique to support collaborative learning. 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). CBM is also used to model student and group knowledge.

## 3. COLLECT-UML: Single-User Version

COLLECT-UML is a 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 a correct solution by providing feedback. The feedback is tailored towards each student depending on his/her knowledge. COLLECT-UML 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. For details on system's architecture, functionality and the interface refer to [2, 3]; here we present only the basic features of the system.

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 modeller 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 modeller diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module, which generates appropriate feedback.

COLLECT-UML contains an ideal solution for each problem, which is compared to the student's solution according to the system's domain model, represented as a set of constraints [22]. The system's domain model contains 133 constraints that describe the basic principles of the domain. In order to develop constraints, we studied material in textbooks, such as [12], and also used our own experience in teaching UML and OO analysis and design.

Figure 1 illustrates a constraint from the UML domain. The relevance condition identifies a relationship of type aggregation in the ideal solution, and then checks whether the student's solution contains the same type of relationship, or a relationship of a different kind with the same name. The student's solution is correct if the satisfaction condition is met, when the matching relationship is of the same type (i.e. aggregation). The constraint also contains a message which 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 relationships, and also identifies the relationship and the classes to which the constraint was applied.

```
(78
 "Check the type of your relationships. You need to use aggregations between some of
your classes."
   (and (match IS RELATIONSHIPS (?* "@" ?rel_tag "aggregation" ?c1_tag ?c2_tag ?*))
     (or-p (match SS RELATIONSHIPS (?* "@" ?rel_tag ?type ?c1_tag ?c2_tag ?*))
         (match SS RELATIONSHIPS (?* "@" ?rel_tag ?type ?c2_tag ?c1_tag ?*))))
 (test SS ("aggregation" ?type))
 "relationships"
 (?rel_tag ?c1_tag ?c2_tag))
```

**Figure 1.** An example constraint

We performed an evaluation study [3] in May 2005 with 38 students enrolled in a Software Engineering course. The students learnt UML modelling concepts during two weeks of lectures/tutorials. The study was conducted in two streams of two-hour laboratory sessions. Each participant sat a pre-test, interacted with the system, and then sat a post-test and filled a user questionnaire. 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. Table 1 presents some general statistics about the study. The average mark on the post-test was significantly higher than the pre-test mark ($t = 2.71$, $p = 4.33E\text{-}08$). The students spent on average 90 minutes interacting with the system.

**Table 1.** Some statistics about the study

| | **Average** | **s. d.** |
|---|---|---|
| Attempted problems | 5.71 | 2.59 |
| Solved problems | 47% | 33% |
| Attempts per problem | 7.42 | 4.76 |
| Pre-test | 52% | 21% |
| Post-test | 76% | 17% |

We also analyzed the log files, in order to identify how students learn the underlying domain concepts. Figure 2 illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all constraints and all participants. The data points show a regular decrease, which is approximated by a power curve with a close fit of 0.93, thus showing that students do learn constraints over time. The probability of violating a constraint on the first occasion of application is halved by the tenth occasion, showing the effects of learning.

Students were offered individualised feedback on their solutions upon submission. The mean rating for the usefulness of feedback was 2.8. 67% of the participants had indicated that they would have liked to see more details in the feedback messages. The comments we received on open questions pointed out several features of the system, which can be improved.

The results showed that COLLECT-$\mathcal{UML}$ is an effective learning environment. The participants achieved significantly higher scores on the post-test, suggesting that they acquired more knowledge in UML modelling. The learning curves also prove that students do learn constraints during problem solving. Subjective evaluation shows that most of the students felt spending more time with the system would have resulted in more learning and that they found the system to be easy to use.

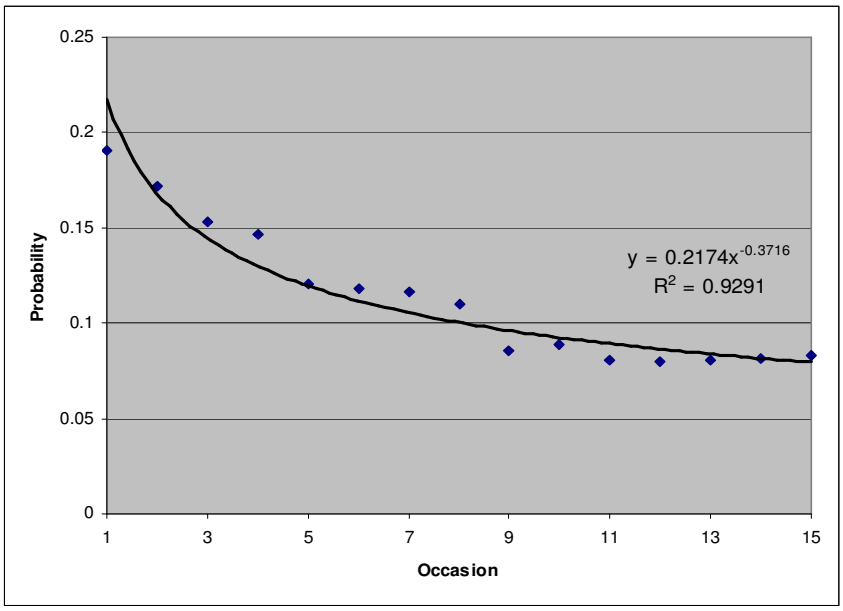



**Figure 2.** Probability of constraint violation

## 4. COLLECT-UML: Multi-User Version

The collaborative version of COLLECT-UML is designed for sessions in which students first solve problems individually and then join into small groups to create group solutions. The system's architecture is illustrated in Figure 3. The application server consists of a session manager that manages sessions and student logs, a student modeller that creates and maintains student models for individual users, a domain model (i.e. the constraint set), a pedagogical module and a group modeller. The system is implemented in Allegro Common Lisp.

The interface is shown in Figure 4. The problem description pane presents a design problem. Students construct their individual solutions in the private workspace (right), and use the shared workspace (left) to collaborate while communicating via the chat window (bottom). The private workspace enables students to try their own solutions and think about the problem before start discussing it in the group. The group area is initially disabled. When all of the students indicate readiness to work in the group by clicking on *Join the Group* button, the shared workspace is activated. The students select the components' names from the problem text. 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. Additionally, this panel shows the name of the student who has the control of this area and the students waiting for a turn.

A recent study [23] defines relevant characteristics of good collaboration and the authors have considered 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 limited communication channel. An implication of providing such 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 key over to the other partners. The advantage, however, is that it maintains clear semantics of a participant's actions and roles in the shared workspace [10]. 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 [11].

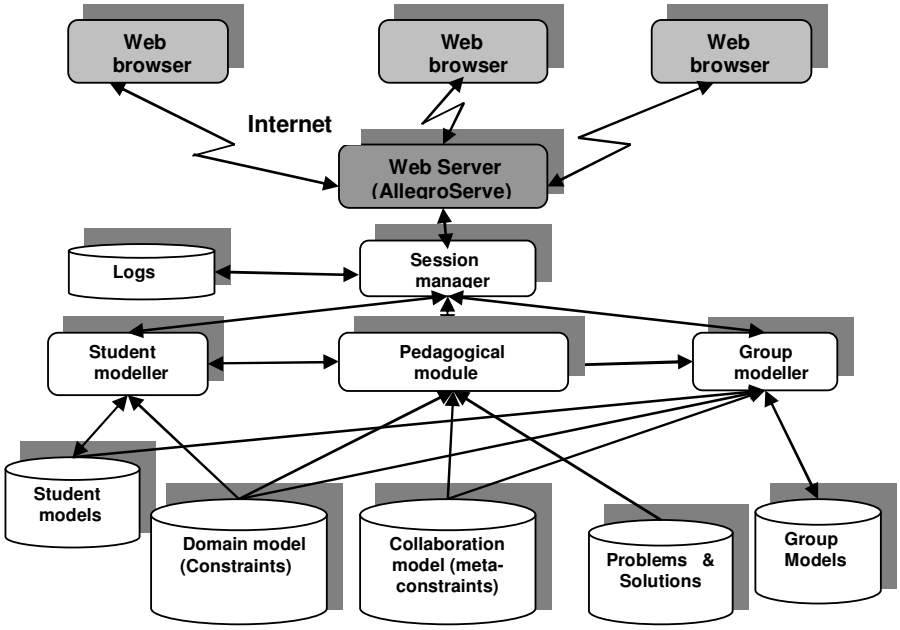



**Figure 3.** The architecture of COLLECT-$\mathcal{UML}$

The chat area enables students to express their opinions using sentence openers. The student needs to select one of the sentence openers before being able to express his/her opinion. The contents of selected sentence openers are displayed in the chat area along with any optional justifications. Sentence openers structure students' conversation and eliminate off-task discussions. A structured chat interface with specific sentence openers can promote more focus on reflection and the fundamental concepts at stake [5]. Although this kind of dialogue requires more effort from the student than using plain chat or email, as the student needs to categorize their own contributions, research shows that the quality of the dialogue can be higher [16]. In addition, structuring the dialogue makes it easier to analyze computationally [10].

Sentence openers provide a natural way for users to identify the intention of their conversational contribution without fully understanding the significance of the underlying communicative acts [19]. Results from various projects indicate that structured dialogues support students to stay on task and increase reflection [13]. However, requiring learners to select a sentence opener 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. According to Lazonder et al. [18], sentence openers should be derived from naturally occurring online text-based free dialogues, while Soller [24] states that it is critical to provide the widest and most appropriate range of sentence

openers. Some experiments [4] show that in interfaces containing both structured and free chat tools, the former are used more frequently.

The group moderator can submit the solution, by clicking on the *Submit Answer* button on the shared workspace. The system gives collaboration-based advice based on the content of the chat area, students' participation on the shared diagram and the differences between students' individual solutions and the group solution being constructed. The task-based advice is given to the whole group based on the quality of the shared diagram.

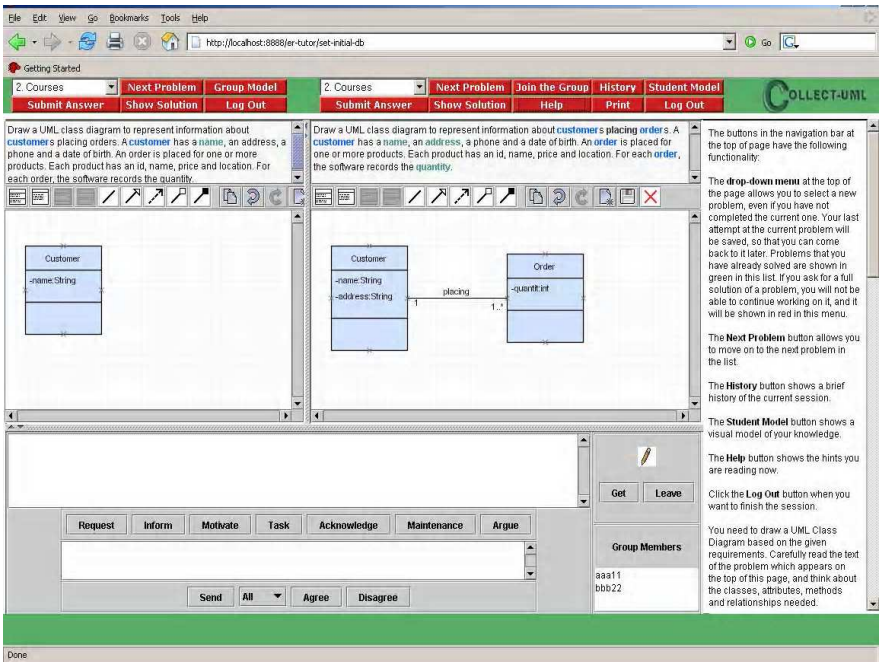



**Figure 4.** COLLECT-$\mathcal{UML}$ interface

The *Next Problem*, *Submit Answer*, *Show Solution* and *Log Out* buttons at the top of the shared diagram are controlled by the group moderator only, while the *Group Model* button can be accessed by all the members. 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 members.

## 5. Modeling Collaboration

The ultimate goal of COLLECT-$\mathcal{UML}$ is to support collaboration by modelling collaborative skills. The system is able to promote effective interaction by

diagnosing students' actions in the chat area and group diagram using a set of 22 meta-constraints, which represent an ideal model of collaboration. These constraints have the same structure as domain constraint, each containing 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 existing literature on characteristics of an effective collaboration, such as [9, 23, 24, 26]. Figure 5 illustrates two examples of meta-constraints. Constraint 221 encourages student participation in problem-solving. This constraint makes sure that the student contributes associations from his/her individual solution to the group solution. On the other hand, there are constraints that check whether the student participates in the dialogue. Constraint 237 checks whether the student has specified any justification for their agreement/disagreement with the group solution.

A history of all contributions made by each user to the shared diagram as well as the messages posted to the chat area is maintained on the server, and the meta-constraints are evaluated against this history. Feedback is given on contributions which involve adding/deleting/updating components in the shared diagram, as well as contributions made to the chat area.

```
(221
  "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."
  (and (match SS RELATIONSHIPS (?* "@" ?rel_tag "association" ?c1_tag ?c2_tag ?*))
       (match GS CLASSES (?* "@" ?c1_tag ?*))
       (match GS CLASSES (?* "@" ?c2_tag ?*)))
  (or-p (match GS RELATIONSHIPS (?* "@" ?rel_tag "association" ?c1_tag ?c2_tag ?*))
        (match GS RELATIONSHIPS (?* "@" ?rel_tag "association" ?c2_tag ?c1_tag ?*)))
  "relationships"
  (?rel_tag ?c1_tag ?c2_tag))

 (237
  "You may wish to explain to other members why you agree or disagree  with a solution."
  (and (match SC DESC (?* "@" ?tag ?text ?*))
       (or-p (test SC ("agree" ?tag))
             (test SC ("disagree" ?tag))))
  (not-p (test SC ("" ?text)))
  "descriptions"
  nil)
```

**Figure 5.** Examples of meta-constraints

## 6. Conclusions and Future Work

This paper presented the single-user version of COLLECT-UML, and the results of the evaluation study performed. The results of both subjective and objective analysis proved that COLLECT-UML is an effective educational tool. The participants performed significantly better on a post-test after short sessions with the system, and reported that the system was relatively easy to use.

We then presented the multi-user version of the same intelligent tutoring system. We have extended COLLECT-$\mathcal{UML}$' interface, and developed meta-constraints, which provide feedback on collaborative activities. The goal of future work is to complete the implementation of the multi-user version and conduct a full evaluation study with second-year University students enrolled in an undergraduate software engineering course. The study is planned for April 2006. Participants will be divided into three groups. The experimental condition will receive feedback on the domain model as well as their collaborative activities. The students will also be provided with a script addressing the characteristics of a good collaboration and the phases they are expected to go through, at the beginning of the session. The second group will receive feedback on their solutions only. These students will be provided with the same script at the beginning of the session, but will not receive feedback on collaboration. The control group will only receive feedback on the domain level. There will not be any type of support on the collaboration process available to this group. Our hypothesis is that all groups will increase their problem-solving skills, but that only the experimental group will improve collaboration skills. All participants will be assessed on their understanding of what characterises good collaboration at the end of the session by answering questions in the post-test. Their interaction in the shared diagram and chat area will also be analysed.

CBM has been used to effectively represent domain knowledge in several ITSs supporting individual learning. The contribution of the project presented in this paper is the use of CBM to model collaboration skills, not only domain knowledge. Comprehensive evaluation of the multi-user version of COLLECT-$\mathcal{UML}$ will provide a measure of the effectiveness of using the CBM technique in intelligent computer-supported collaborative learning environments.

## References


1. Baghaei, N. and Mitrovic, A. (2005) *COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based tutor.* In Khosla, R., Howlett, R. and Jain L. (eds.) KES 2005, pp.458-464.
2. Baghaei, N., Mitrovic, A. and Irwin, W. (2005) *A Constraint-Based Tutor for Learning Object-Oriented Analysis and Design using UML.* In Looi, C., Jonassen, D. and Ikeda M. (eds.) ICCE 2005, pp.11-18.
3. Baghaei, N., Mitrovic, A. and Irwin, W. (2006) *Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams*, Technology, Instruction, Cognition and Learning Journal, 4(1-2) (in print).
4. Baker, M. J. and Lund, K. (1997) *Promoting reflective interactions in a computer-supported collaborative learning environment*. Journal of Computer Assisted Learning, 13(3), 175-193.
5. Baker, M., de Vries, E., Lund, K. and Quignard, M. (2001) *Computer-mediated epistemic interactions for co-constructing scientific notions: Lessons learned form a five-year research program.* In Dillenbourg, P., Eurelings, A. and Hakkarainnen, K. (eds.) European Perspectives on CSCL (CSCL 2001), pp.89-96.
6. Barros, B. and Verdejo, M.F. (2000) *Analysing student interaction processes in order to improve collaboration:The DEGREE approach.* Artificial Intelligence in Education, 11, 221-241.
7. Constantino-Gonzalez, M., and Suthers, D. (2000) *A coached collaborative learning environment for Entity-Relationship modelling*. In Gauthier, G., Frasson, C. and VanLehn, K. (eds.) 5th Int. Conf. Intelligent Tutoring Systems, pp.324-333.
8. Constantino-Gonzalez, M., and Suthers, D. (2002) *Coaching Collaboration in a*

*Computer Mediated Learning Environment*. In Stahl, G. (eds.) CSCL 2002, pp.583-584.

9. Constantino-Gonzalez, M. A., Suthers, D. and 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), 261-297.
10. Dimitracopoulou, A. (2005) *Designing Collaborative Learning Systems: Current Trends & Future Research Agenda.* In Koschmann, T., Suthers, D. and Chan T.W. (eds.) 6th Int. Computer Supported Collaborative Learning Conf. CSCL 2005.
11. Feidas, C., Komis, V. and Avouris, N. (2001) *Design of collaboration-support tools for group problem solving*. In Avouris, N. and Fakotakis, N. (eds.) Advances in Human-Computer Interaction, pp.263-268.
12. Fowler, M. (2004) *UML Distilled: a Brief Guide to the Standard Object Modelling Language*. Reading: Addison-Wesley, 3rd edition.
13. Gogoulou, A., Gouli, E., Grigoriadou, M. and Samarakou, M. (2005) *ACT: A Web – based Adaptive Communication Tool.* In Koschmann, T., Suthers, D. and Chan T.W. (eds.) 6th Int. Computer Supported Collaborative Learning Conf., pp.180-189.
14. Inaba, A. and Mizoguchi, R. (2004) *Learners' Roles and Predictable Educational Benefits in Collaborative Learning; An Ontological Approach to Support Design and Analysis of CSCL.* In Lester, J., Vicari, R. M. and Paraguacu, F. (eds.) 7th Int. Conf. Intelligent Tutoring Systems, pp.285–294.
15. Jarboe, S. (1996) *Procedures for enhancing group decision making*. In Hirokawa B. and Poole M. (eds.) Communication and Group Decision Making, pp.345-383.
16. Jermann, P., Soller, A. and Lesgold, A. (2004) *Computer software support for CSCL.* In Dillenbourg P., Strijbos J.W., Kirschner, P.A. and Martens R.L. (eds.) Computer-supported collaborative learning: Vol 3. What we know about CSCL ... and implementing it in higher education, pp.141-166.
17. Jerman, P., Soller, A. and Muhlenbrock, M. (2001) *From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning*. In Dillenbourg, P., Eurelings, A and Hakkarainen, K. (eds.) European Perspectives on CSCL (CSCL 2001), pp.324-331.
18. Lazonder, A., Wilhelm, P. and Ootes S. (2003) *Using sentence openers to foster student interaction in computer-mediated learning environments*. Computers & Education, 41, 291-308.
19. McManus, M. and Aiken, R. (1995) *Monitoring computer-based problem solving*. Artificial Intelligence in Education, 6(4), 307-336.
20. Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B. (2001) *Constraint-based Tutors: a Success Story*. 14th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp.931-940.
21. Ogata, H., Matsuura, K., and 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.
22. Ohlsson, S. (1994) *Constraint-based Student Modelling.* Student Modelling: the Key to Individualized Knowledge-based Instruction, Springer-Verlag, pp.167-189.
23. Rummel, N. and 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.
24. Soller, A. (2001) *Supporting Social Interaction in an Intelligent Collaborative Learning System.* Artificial Intelligence in Education, 12, 40-62.
25. Soller, A. and Lesgold, *A*. (2000) *Knowledge acquisition for adaptive collaborative learning environments.* AAAI Fall Symposium: Learning How to Do Things, Cape Cod, MA.
26. Vizcaino, A. (2005) *A Simulated Student Can Improve Collaborative Learning.* Artificial Intelligence in Education, 15, 3-40.
27. Webb, N. M., Troper, J. D. and Fall, R. (1995) *Constructive activity and learning in collaborative small groups*. Journal of Educational Psychology, 87, 406-423.

# UM 2007 Paper

The following paper [Baghaei and Mitrovic, 2007] was published and presented at the UM 2007 conference in Corfu, Greece.

# From Modelling Domain Knowledge to Metacognitive Skills: Extending a Constraint-based Tutoring System to Support Collaboration

Nilufar Baghaei and Antonija Mitrovic

Department of Computer Science and Software Engineering
University of Canterbury, Private Bag 4800, Christchurch, New Zealand
nilufar.baghaei@gmail.com, tanja@cosc.canterbury.ac.nz

**Abstract.** Constraint-based tutors have been shown to increase individual learning in real classroom studies, but would become even more effective if they provided support for collaboration. COLLECT-UML is a constraint-based intelligent tutoring system that teaches object-oriented analysis and design using Unified Modelling Language. 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. 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 problem-solving. The effectiveness of the system was evaluated in a study conducted at the University of Canterbury in May 2006. In addition to improved problem-solving skills, the participants both acquired declarative knowledge about good collaboration and did collaborate more effectively. The results, therefore, show that Constraint-Based Modelling is an effective technique for modelling and supporting collaboration skills.

## 1 Introduction

Constraint-based tutors are Intelligent Tutoring Systems (ITS) which use Constraint-Based Modelling (CBM) [15] to represent 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 [4], 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), database modelling, data normalization [13], punctuation [11] and English vocabulary [10].

All constraint-based tutors developed so far support individual learning. This paper describes extending COLLECT-UML [1, 3], a constraint-based ITS, to support the acquisition of collaboration skills. COLLECT-UML teaches Object-Oriented (OO) analysis and design using Unified Modelling Language (UML). 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 Section 2. The architecture of COLLECT-UML and its interface are discussed in Section 3. Section 4 describes the collaborative model, which has been implemented as a set of meta-constraints. In Section 5, we present the results of an evaluation study conducted recently. Conclusions are given in the last section.

## 2 Related Work

In the last decade, many researchers have contributed to the development of computer-supported collaborative learning (CSCL) and advantages of collaborative learning over individualised learning have been identified. Some particular benefits of collaborative problem-solving include: encouraging students to verbalise 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 [17]. These benefits, however, are only achieved by well-functioning learning teams [8]. Various strategies for computationally supporting online collaborative learning have been proposed and used, but more studies are needed that test the utility of these techniques [9].

CSCL systems can be classified into three categories based on their collaboration support [9]. The first category includes systems that reflect actions; this basic level of support makes students aware of each others' actions. 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 [14]), while others internally compare the current state of interaction to a model of ideal interaction, but do not reveal this information to the users (e.g. EPSILON [18]). In the latter case, this information is either intended to be used later by a coaching agent, or analysed by researchers in order to understand the interaction [9]. Finally, the third class of systems offer advice on collaboration. The coach in these systems plays a role similar to that of a teacher. The systems can be distinguished by the nature of the information in their models, and whether they provide feedback on strictly collaboration issues or both social and task-oriented issues. An example of the systems focusing on the social aspects is Group Leader Tutor [12], while COLER [5] addresses both social and task-oriented aspects of group learning.

Although many tutorials, textbooks and other resources on UML are available, we are not aware of any attempt at developing a CSCL environment for UML modelling. However, there has been an attempt [18] at developing a collaborative learning environment for OO design problems using Object Modeling Technique (OMT), a precursor of UML. The system 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 problems. The system dynamically assesses a group's interaction, and determines when and why the students are having trouble learning new concepts they share with each other. The system does not evaluate the OMT diagrams and an instructor or intelligent coach's assistance is needed in mediating group knowledge sharing activities. In this regard, even though the system is effective as a collaboration tool, it would probably not be

an effective teaching system for a group of novices with the same level of expertise, as the students may agree on the same flawed argument.

## 3 COLLECT-UML

COLLECT-UML is a problem-solving environment implemented in Allegro Common Lisp, 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 constraint-based tutoring system which supported students working individually. Being a Web-enabled system, its interface is delivered via a Web browser. The system consists of a session manager that manages sessions and student logs, a student modeller that maintains student models, the constraint set and a pedagogical module. We performed an evaluation study in a real classroom, and

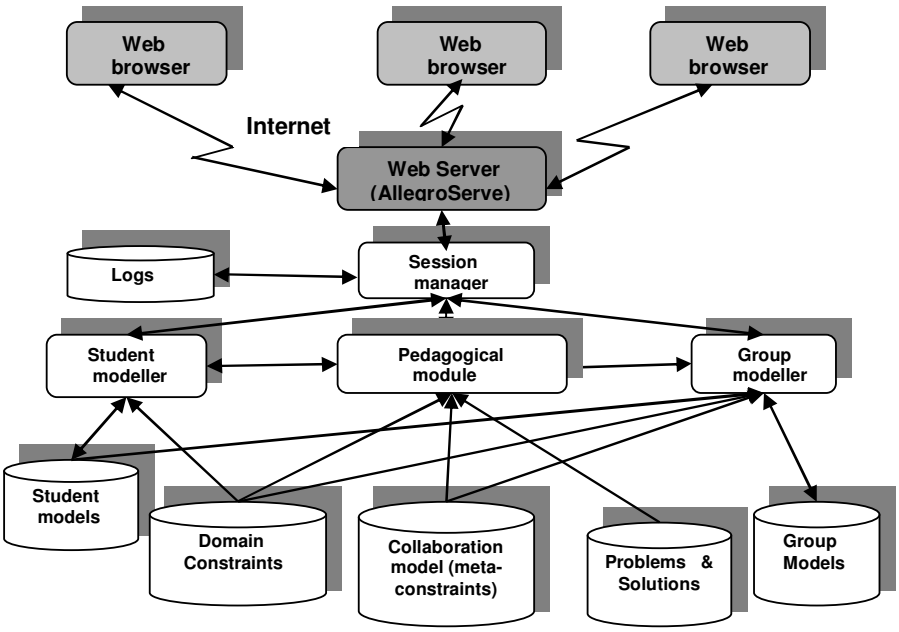



**Fig. 1.** The architecture of COLLECT-UML

the results showed that students' performance increased significantly. For details on the functionality and the evaluation studies of this version please refer to [1, 3].

The architecture of the collaborative version of the system (Figure 1) introduces the group modeller, a new component responsible for creating and maintaining group models. The pedagogical module uses both the student model and the group 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 group model records the history of group usage for each domain constraint.

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 [15]. The system's domain model contains a set of 133 constraints defining the basic domain principles, a set of problems and their solutions [3]. In order to develop constraints, we studied material in textbooks, such as [7], and also used our own experience in teaching UML and OO analysis and design. Figure 2 illustrates a constraint from the UML domain, which checks whether the student has defined all the methods necessary for the current problem. The relevance condition identifies a method in the ideal solution (IS) and then checks whether the class it belongs to also exists in the student's solution (SS). The student's solution is correct if the satisfaction condition is met, when the matching method also exists in the student's solution. The constraint also contains a message which would be given to the student if the constraint is violated.

```
(54
  "Check whether you have defined all the methods as specified
   by the problem. You are missing some methods."
  (and (match IS METHODS (?* "@" ?tag ?name ?class_tag ?*))
       (match SS CLASSES (?* "@" ?class_tag ?*)))
  (match SS METHODS (?* "@" ?tag ?name2 ?class_tag ?*))
  "methods"
  (?class_tag))
```

**Fig. 2.** Example of a domain constraint

The student interface is shown in Figure 3. The problem text describes a situation 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 drawing new components in the group diagram. The private and shared workspaces can be resized. The students need to select the component names from the problem text by highlighting or double-clicking on the words/phrases. 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*, and shows the name of the student who has the control of this area. The chat area enables students to express their opinions by selecting one of the sentence openers, and typing their statement.

While all group members can contribute to the chat area and 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: *Simple Feedback*, *Error flag*, *Hint* and *All Hints*. In addition, the group moderator has the option of asking for the complete solution, by clicking on *Show Full Solution* button. The collaboration-based advice is given to individual students based on the content of the chat area (i.e. sentence openers the students used), the student's contributions to the shared diagram and the differences between student's individual solution and the group solution being constructed. The system scales to a large number of participants and to large problem spaces. For more details on the interface and justification of using sentence openers, private workspace and turn taking, please refer to [2].

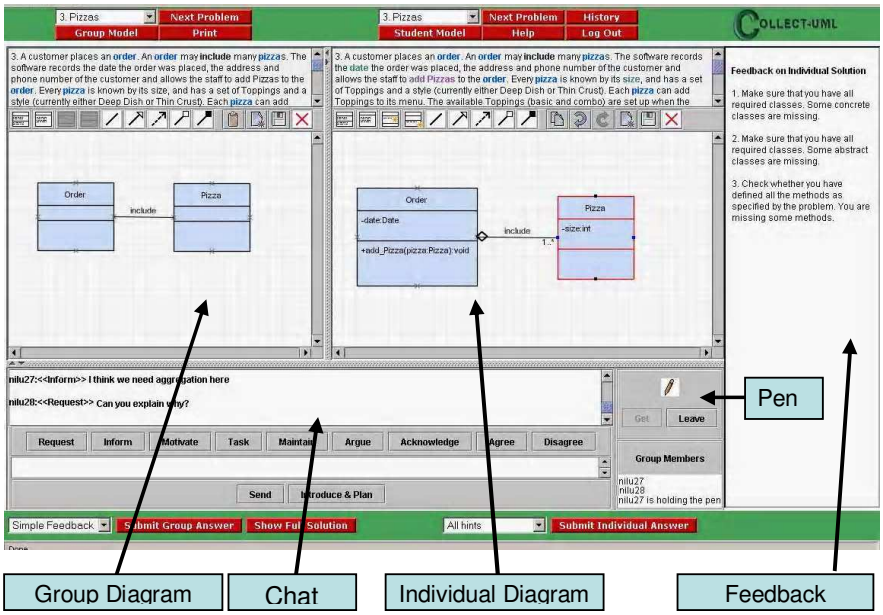



**Fig. 3.** COLLECT-$\mathcal{UML}$ Interface

## 4 Modelling Collaboration

Research on learning has demonstrated the usefulness of collaboration for improving student's problem-solving skills. However, 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 practised to be acquired. The goal of our research is to support collaboration by modelling collaborative skills. COLLECT-$\mathcal{UML}$ 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 constraint 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 [5, 16, 17, 19], and also used our own experience in collaborative work. The collaborative teaching strategy is based on the socio-cognitive conflict theory [6]. According to this theory, social interaction is constructive only if it creates a confrontation between students' divergent solutions. The meta-constraints are divided into two 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.), and constraints that monitor students' contributions to the chat area and the use of sentence openers.

Figure 4 illustrates two meta-constraints. The relevance condition of constraint 223 focuses on aggregation relationships that exist in the student's individual solution between certain classes, when the same classes also exist in the group solution (GS). For this constraint to be satisfied, the corresponding relationships 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 them to discuss those relationships with the group, or add them to the group solution. 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 sentence openers. If not, it encourages them to provide more explanation as part of their contribution.

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 id of the student who made that change, is sent to the server. Each chat message will also be sent to the server in the XML format. Histories of all contributions made to the shared diagram as well as the messages posted to the chat area are stored on the server. The meta-constraints are evaluated against these histories, and feedback is given on contributions which involve adding/deleting/updating components in the shared diagram, as well as contributions made to the chat area.

## 5 Evaluation

An evaluation study was carried out at the University of Canterbury in May 2006. The study involved 48 volunteers enrolled in an introductory Software Engineering course. The students learnt UML modelling concepts during two weeks of lectures and had some practice during two weeks of tutorials prior to the study. The study was conducted in two streams of two-hour laboratory sessions over two weeks. In the first week, the students filled out a pre-test and interacted with the single-user version. 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.

```
(223
  "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."
  (and (match SS RELATIONSHIPS (?* "@" ?rel_tag "aggregation"
        ?c1_tag ?c2_tag ?*))
       (match GS CLASSES (?* "@" ?c1_tag ?*))
       (match GS CLASSES (?* "@" ?c2_tag ?*)))
  (or-p (match GS RELATIONSHIPS (?* "@" ?rel_tag "aggregation"
        ?c1_tag ?c2_tag ?*))
        (match GS RELATIONSHIPS (?* "@" ?rel_tag "aggregation"
        ?c2_tag ?c1_tag ?*)))
   "relationships"
  (?rel_tag ?c1_tag ?c2_tag))

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

**Fig. 4.** Examples of meta-constraints

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 were also handed out. The idea of providing students with such a script and therefore supporting instructional learning came from a recent study conducted by Rummel and Spada [16]. The participants were also given a screenshot of the system highlighting the important features of the multi-user interface (Figure 3).

The students were randomly divided into pairs with a pre-specified moderator. The moderator for each pair was the student who had scored higher in the pre-test. The pairs worked on a relatively complex problem individually and joined the group discussion whenever they were ready – the group diagram was activated after 10 minutes. At the end of the session, each participant completed a post-test and 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 on their individual and collaborative activities.

The experimental group consisted of 26 students (13 pairs) who received feedback on their solution as well as their collaborative activities. The control group consisted of 22 students (11 pairs) who only received feedback on their solutions (no feedback on collaboration was provided in this case). All pairs received instructions on characteristics of good collaboration at the beginning of second week.

The total time spent interacting with the system was 1.4 hours for the control and 1.3 hours for the experimental group. 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 also

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 1. The numbers reported for the post-test do not include the collaboration question.

**Table 1.** Pre- and post-test scores

| | Control | | Experimental | |
|---|---|---|---|---|
| | **Average** | **s. d.** | **Average** | **s. d.** |
| Collaboration | 22% | 22% | 52% | 39% |
| Pre-test | 52% | 20% | 49% | 19% |
| Post-test | 76% | 25% | 73% | 25% |
| Gain score | 17% | 28% | 21% | 31% |

There was no significant difference on the pre-test results, meaning that the groups were comparable. The students' performance on the post-test was significantly better for both control group ($t = 2.11$, $p = 0.01$) and experimental group ($t = 2.06$, $p = 0.002$). The experimental group, who received feedback on their collaboration performed significantly better on the collaboration question ($t = 2.02$, $p = 0.003$), showing that they acquired more knowledge on effective collaboration. We also calculated the effect size for the question about collaboration. The common method to calculate it is to subtract the control group's mean score from the experimental group's mean score and divide by the standard deviation of the control group. Using this method, the effect size on student's collaboration knowledge is very high: $(Average\ collaboration_{exp} - Average\ collaboration_{control}) / s.d._{control} = 1.3$.

The experimental group students contributed more to the group diagram, with the difference between the average number of individual contribution for control and experimental group 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 (for each student).

We have also analyzed the students' individual log files, in order to identify how students learnt the underlying domain concepts in the second week. Figure 5

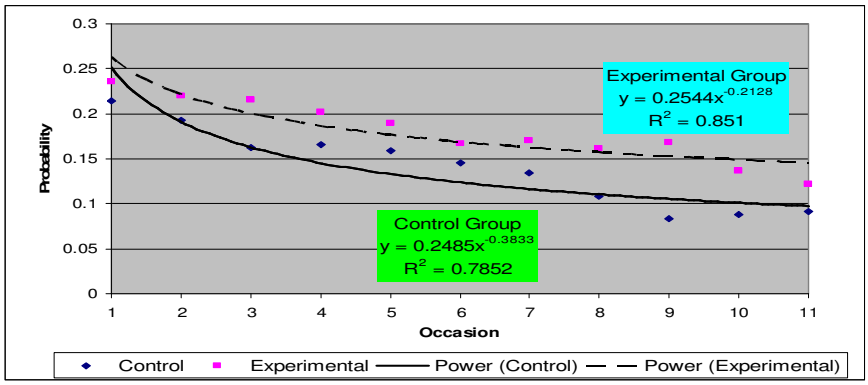



**Fig. 5**. Probability of domain constraint violation for individuals in control and experimental group

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 control and experimental groups respectively, thus showing that students do learn constraints over time. The probability of 0.21/0.23 for violating a constraint on the first occasion of application has decreased to 0.09/0.12 at its eleventh occasion, displaying a 61.9%/47.8% decrease in probability for the control/experimental group respectively. Figure 6 illustrates the learning curve for meta-constraints only (for the experimental group). There is also a regular decrease, 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.

The participants were given a questionnaire at the end of the session to determine their perceptions of the system. Most of the participants (61% of control and 78% of experimental group) responded they would recommend the system to other students. The students found the interface easy to learn and use and enjoyed working with a partner. 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.

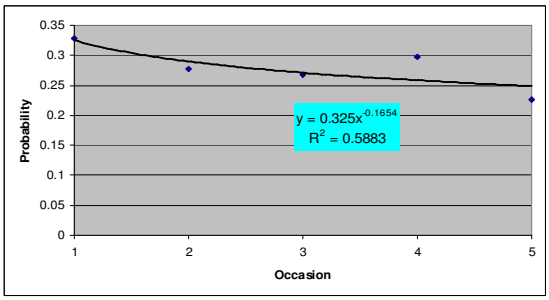



**Fig. 6.** Probability of meta-constraint violation

## 6 Conclusions

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. We described the process of extending COLLECT-$\mathcal{UML}$, an ITS for UML class diagrams, to support collaboration.

The system's effectiveness in teaching good collaboration and UML class diagrams was evaluated in a classroom experiment. The results of both subjective and objective analysis proved that COLLECT-$\mathcal{UML}$ is an effective educational tool. The experimental group students acquired more declarative knowledge on effective collaboration, as they scored significantly higher on the collaboration test. 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. All students improved their problem-solving skills: the participants from both control and experimental group performed significantly better on the post-test after short sessions with the system, showing that they acquired more knowledge on

UML modelling. Finally, the students enjoyed working with the system and found it a valuable asset to their learning.

The results, therefore, show that CBM is an effective technique for modelling and supporting collaboration in CSCL environments.

## References


1. Baghaei, N., Mitrovic, A., Irwin, W.: A Constraint-Based Tutor for Learning Object Oriented Analysis and Design using UML. Proc. ICCE 2005, (2005) 11-18
2. Baghaei, N., Mitrovic, A.: A Constraint-based Collaborative Environment for Learning UML Class Diagrams. Proc. ITS 2006, (2006) 176-186
3. Baghaei, N., Mitrovic, A., Irwin, W.: Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams, Technology, Instruction, Cognition and Learning Journal, 4(2) (2006) (in print)
4. Brusilovsky, P., Peylo, C.: Adaptive and Intelligent Web-based Educational Systems. Artificial Intelligence in Education, 13, (2003) 159-172
5. Constantino-Gonzalez, M. A., Suthers, D., Escamilla de los Santos, J.: Coaching web-based collaborative learning based on problem solution differences and participation. Int. J. Artificial Intelligence in Education, 13(2-4), (2003) 263-299
6. Doise, W., Mugny, G.: The social development of the intellect. Int. Series in Experimental Social Psychology, 10, Pergamon Press. (1984)
7. Fowler, M.: UML Distilled: a Brief Guide to the Standard Object Modelling Language. Reading: Addison-Wesley, 3rd edition. (2004)
8. Jarboe, S.: Procedures for enhancing group decision making. In: B. Hirokawa, M. Poole (eds.): Communication and Group Decision Making. (1996) 345-383
9. Jerman, P., Soller, A., Muhlenbrock, M.: From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning. In: P. Dillenbourg, A. Eurelings, K. Hakkarainen (eds.) CSCL 2001, (2001) 324-331
10. Martin, B., Mitrovic, A.: Domain Modelling: Art or Science? In: Hoppe, U., Verdejo, F., Kay J. (eds.), Proc. 11th Int. Conference on AIED, (2003) 183-190
11. Mayo, M., Mitrovic, A.: Optimising ITS behaviour with Bayesian networks and decision theory. Artificial Intelligence in Education, 12(2), (2001) 124-153
12. McManus, M., Aiken, R.: Monitoring computer-based problem solving. Int. Journal of Artificial Intelligence in Education, 6(4), (1995) 307-336
13. Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A.: DB-suite: Experiences with three Intelligent, Web-based Database Tutors. Journal of Interactive Learning Research, 15(4), (2004) 409-432
14. Ogata, H., Matsuura, K., Yano, Y.: Active Knowledge Awareness Map: Visualizing learners activities in a Web based CSCL environment. Int. Workshop on New Technologies in Collaborative Learning, (2000) 89-97
15. Ohlsson, S: Constraint-based Student Modelling. In: J. Greer, G. McCalla (eds.): Student Modelling: the Key to Individualized Knowledge-based Instruction, Berlin: Springer-Verlag (1994) 167-189
16. Rummel, N., Spada, H.: Learning to collaborate: An instructional approach to promoting collaborative problem-solving in computer-mediated settings. Journal of the Learning Sciences, 14(2), (2005) 201-241
17. Soller, A.: Supporting Social Interaction in an Intelligent Collaborative Learning System. International Journal of AIED, 12, (2001) 40-62
18. Soller, A., Lesgold, A.: Knowledge acquisition for adaptive collaborative learning environments. AAAI Fall Symposium: Learning How to Do Things. (2000)
19. Vizcaino, A.: A Simulated Student Can Improve Collaborative Learning. Int. Journal of Artificial Intelligence in Education, 15, (2005) 3-40

# ijCSCL Paper

The following paper [Baghaei, Mitrovic and Irwin, 2007] was published in the international journal of Computer Supported Collaborative Learning (ijCSCL). It is available on the SpringerLink website: http://www.springerlink.com/content/120055

Computer-Supported Collaborative Learning (2007) 2:159–190
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 /
Published online: 5 September 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

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 et al. 2003) 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.

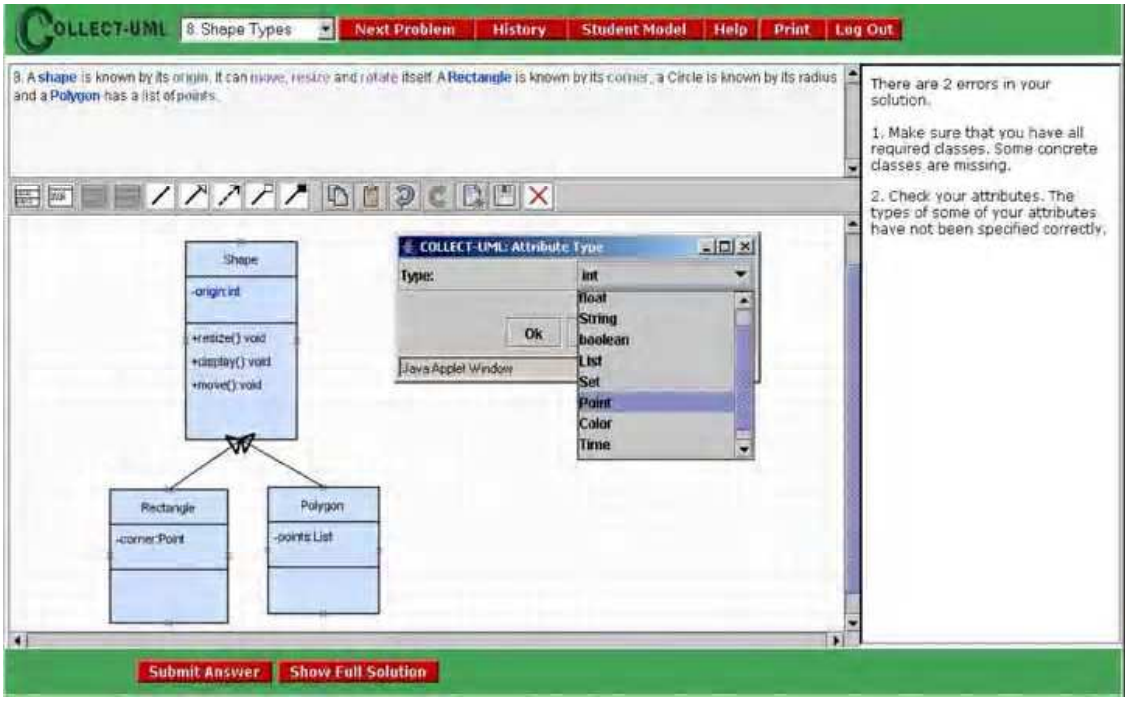



**Fig. 1** Single-user version of COLLECT-UML interface

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

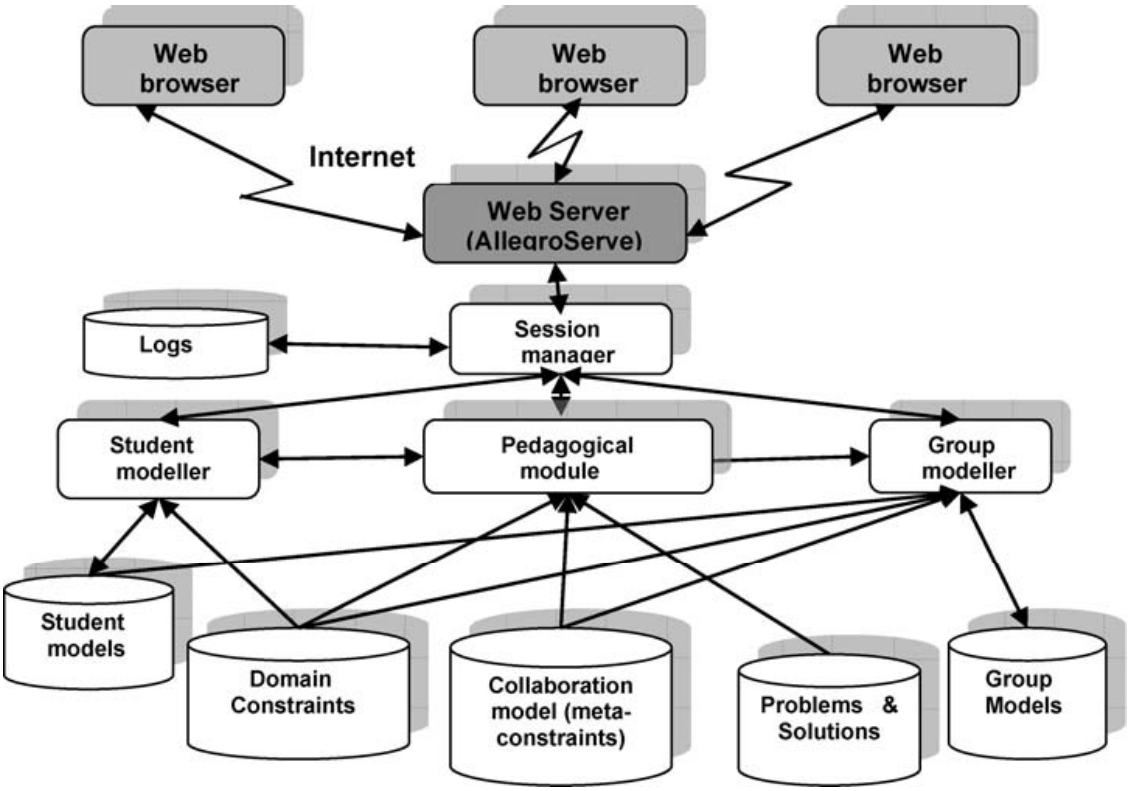



**Fig. 3** The architecture of COLLECT-UML

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

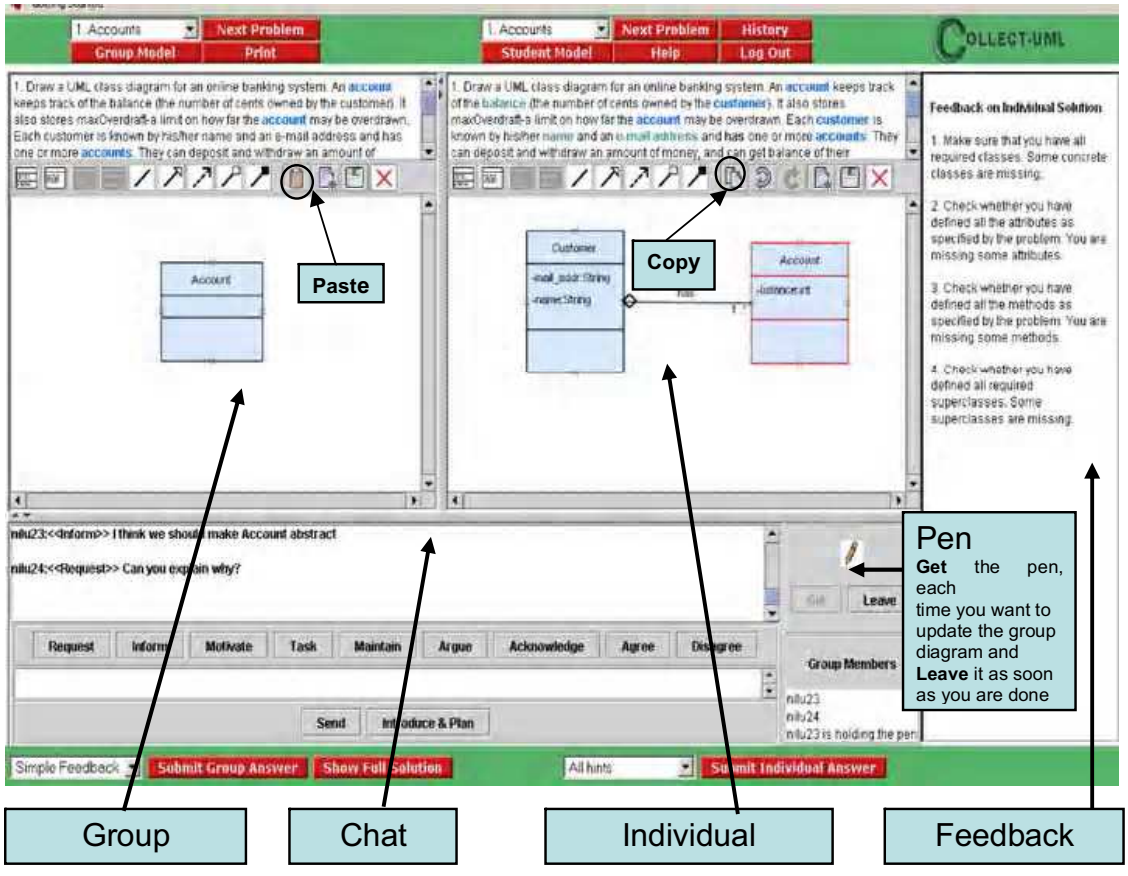



**Fig. 4** COLLECT-UML interface

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

| 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.

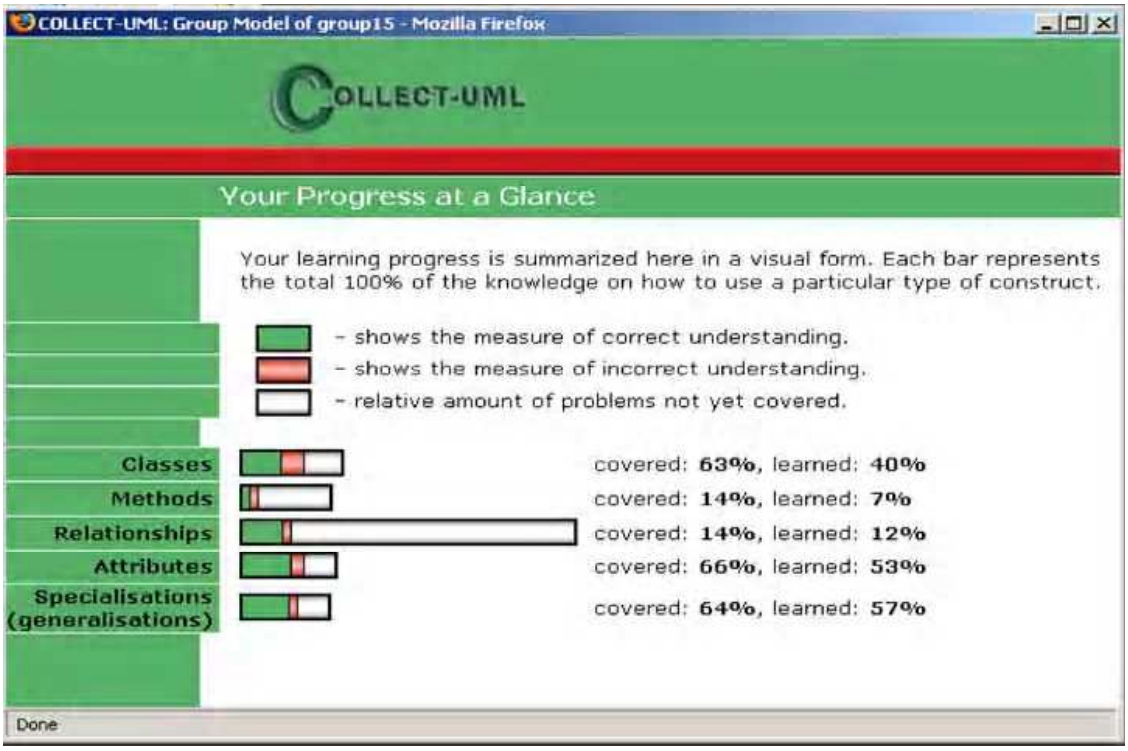



**Fig. 5** Open group model

## 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).

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

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 |

**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:

$$\left( \right) \Big/ s\,d_{\,control} \quad 1\,3$$

## 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 |

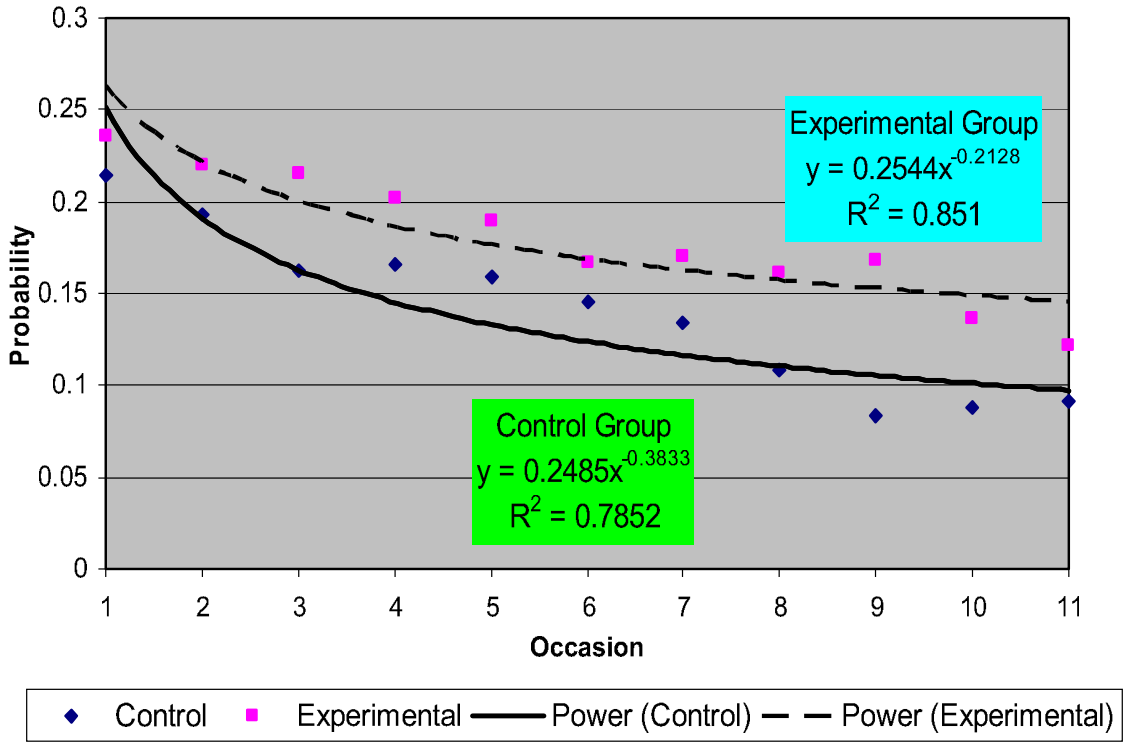



**Fig. 8** Probability of domain constraint violation for individuals in control and experimental groups

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.

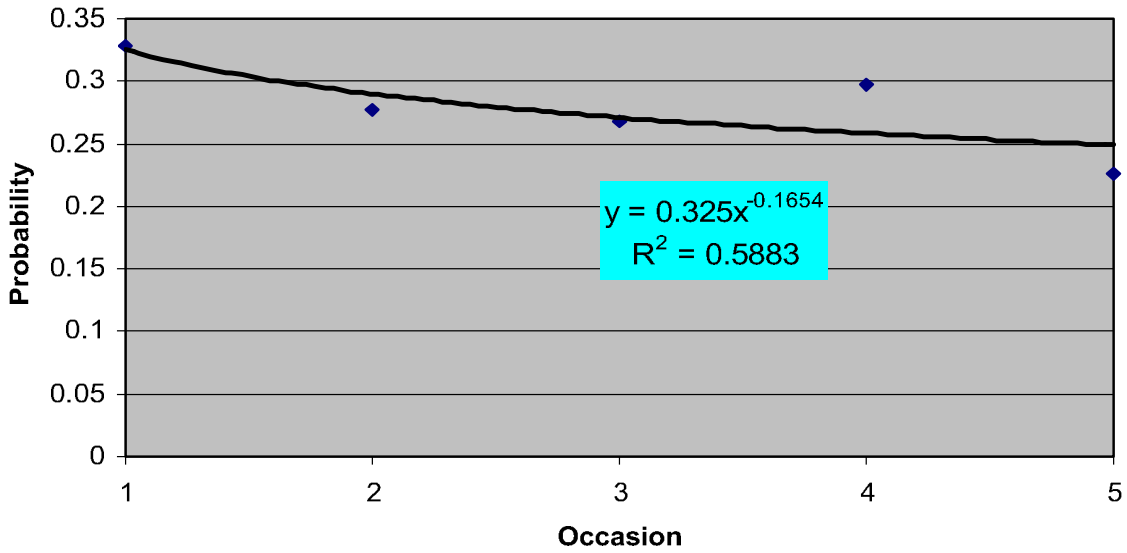



**Fig. 9** Probability of meta-constraint violation for the experimental group

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

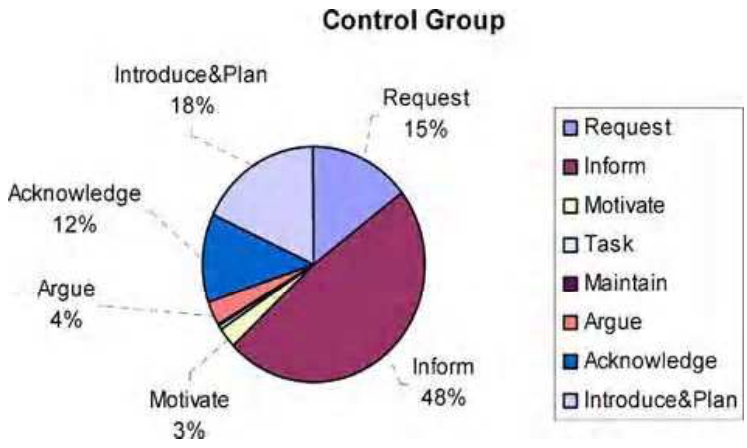



**Fig. 10** Use of communication categories by the control group

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.

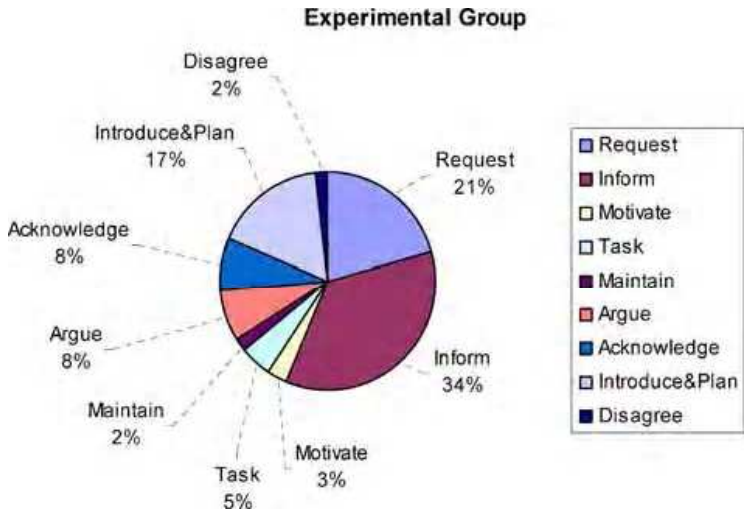



**Fig. 11** Use of communication categories by the experimental group

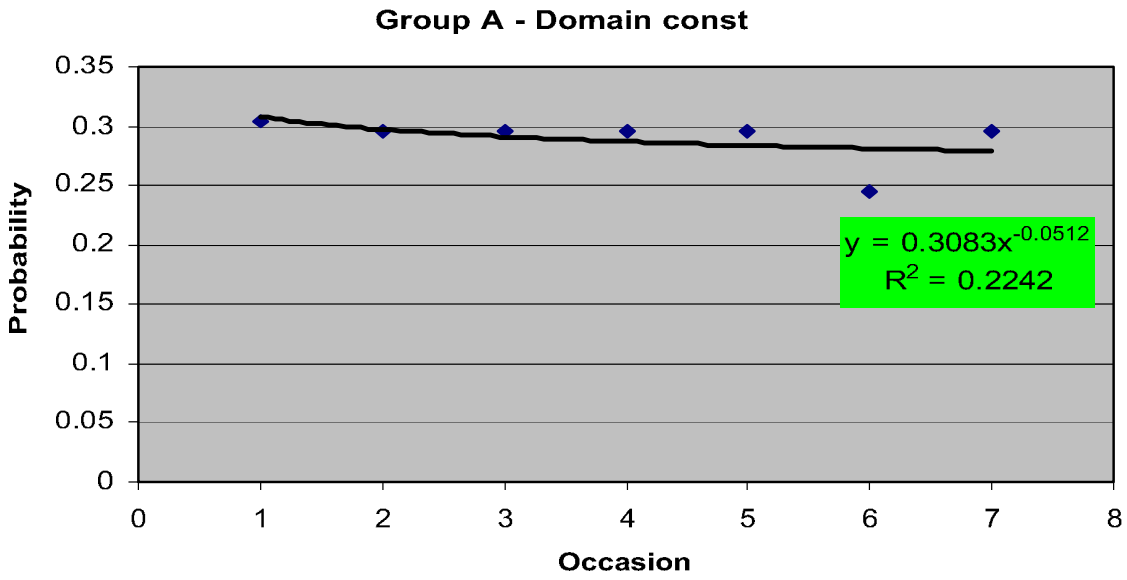



**Fig. 12** Probability of domain constraint violation for Pair A

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

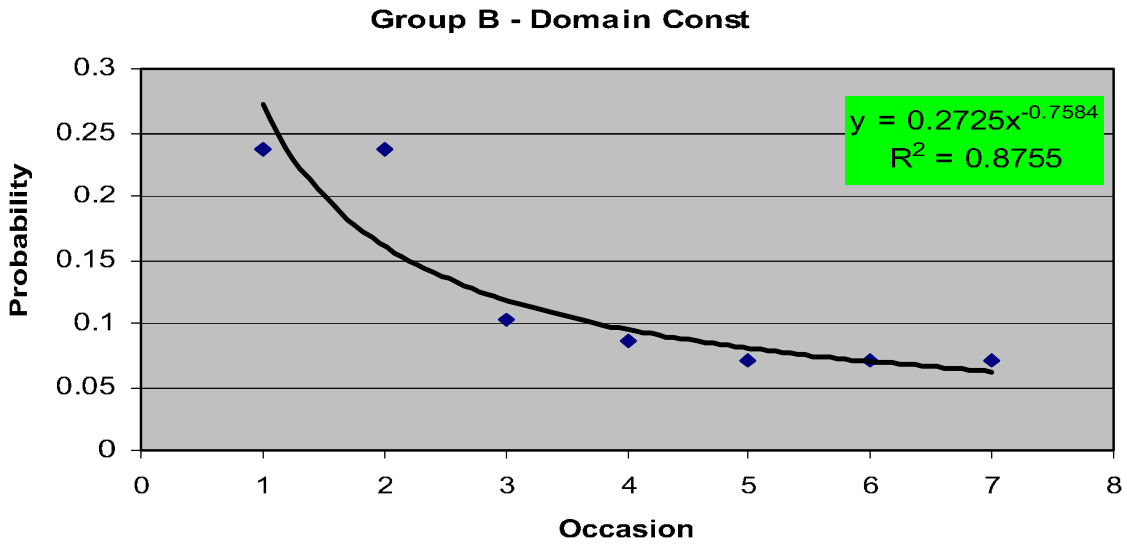



**Fig. 13** Probability of domain constraint violation for Pair B

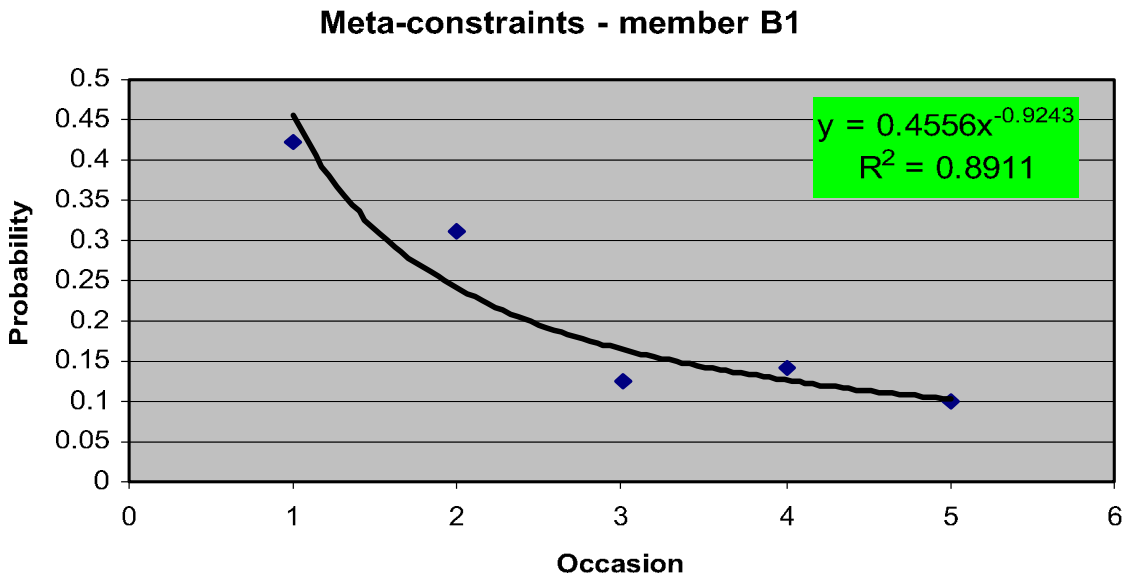



**Fig. 14** Probability of meta-constraint violation for Member B1 of Pair B

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.

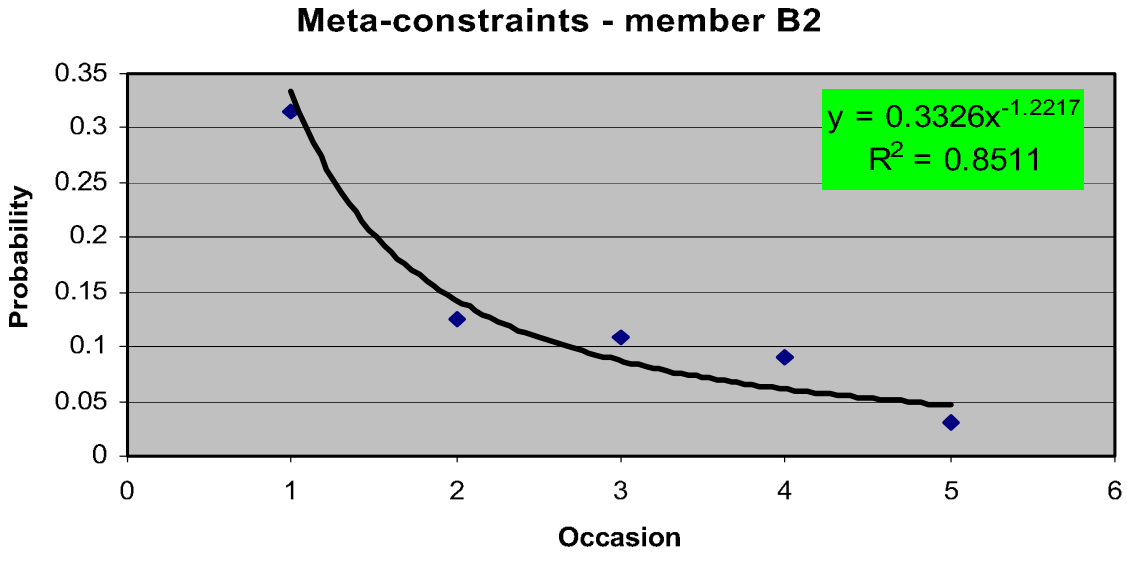



**Fig. 15** Probability of meta-constraint violation for Member B2 of Pair B

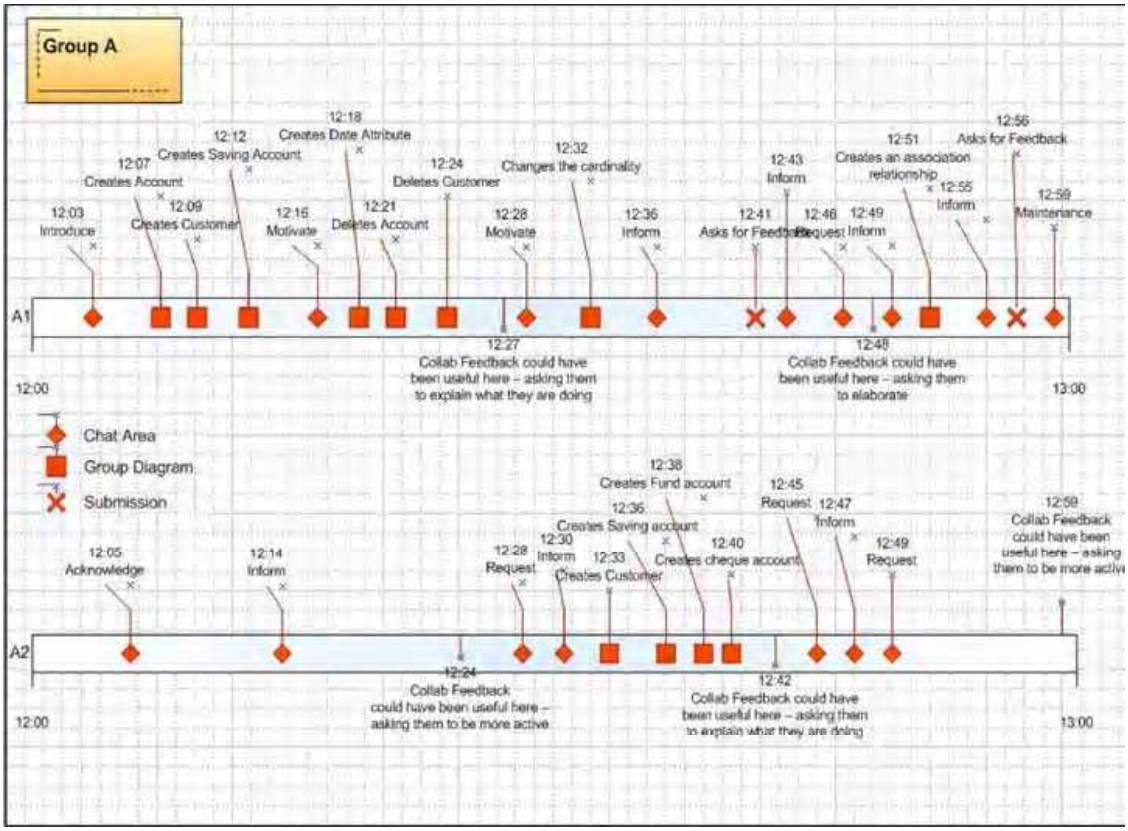



**Fig. 16** Part of the collaboration log of Pair A (control)

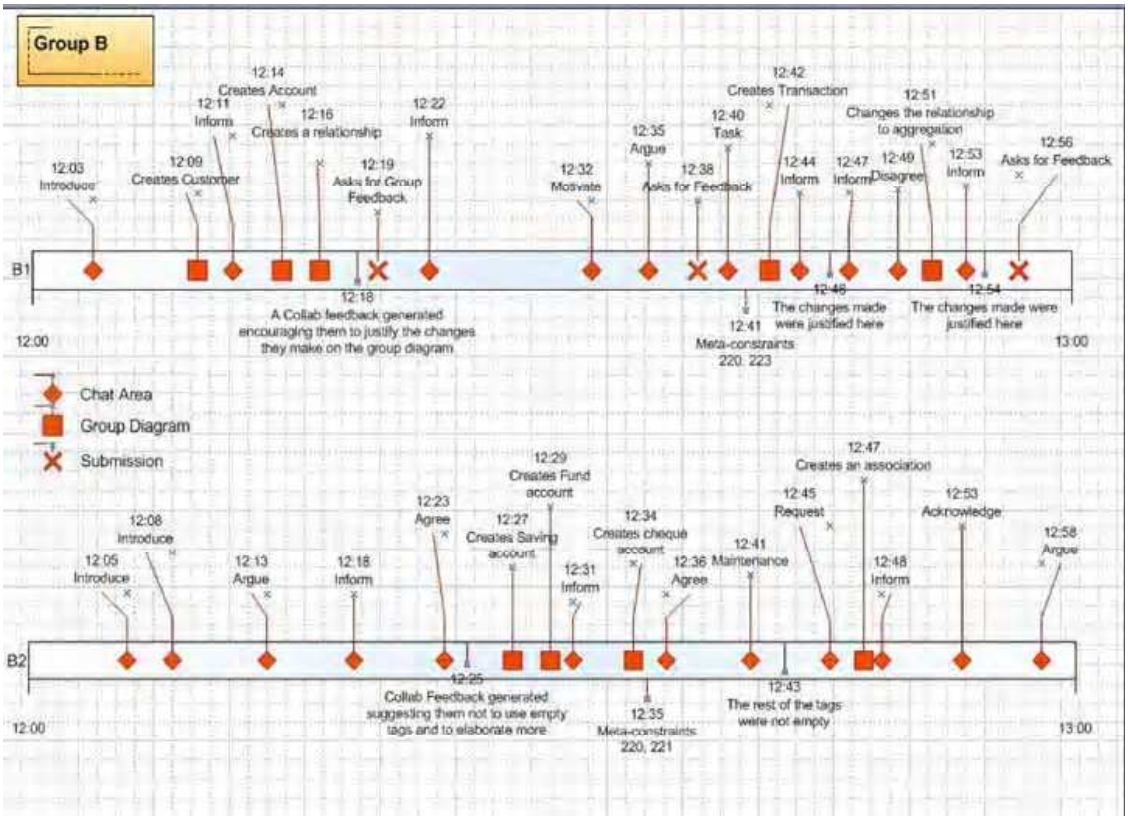



**Fig. 17** Part of the collaboration log of Pair B (experimental)

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

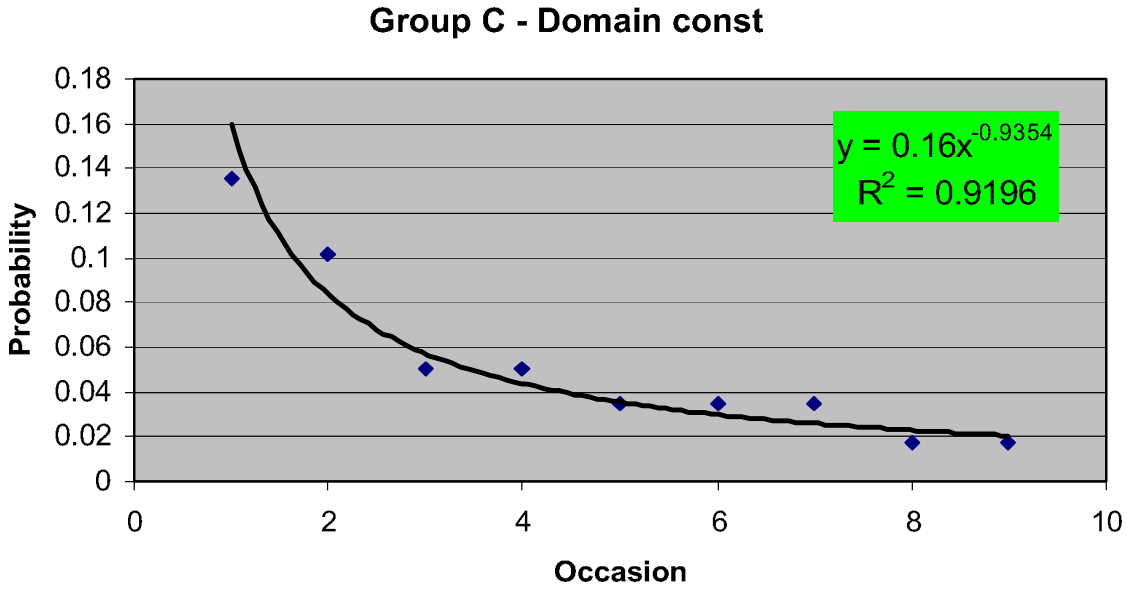



**Fig. 18** Probability of domain constraint violation for Pair C

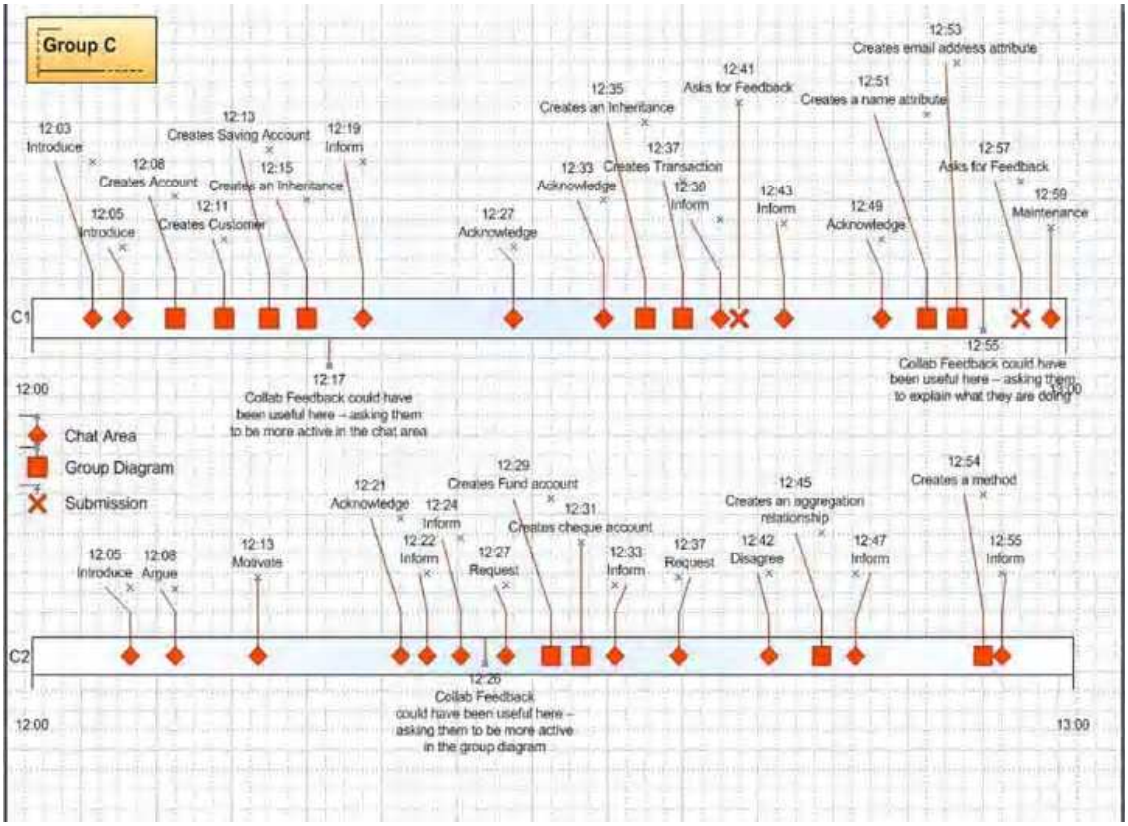



**Fig. 19** Part of the collaboration log of Pair C (control)

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.

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:

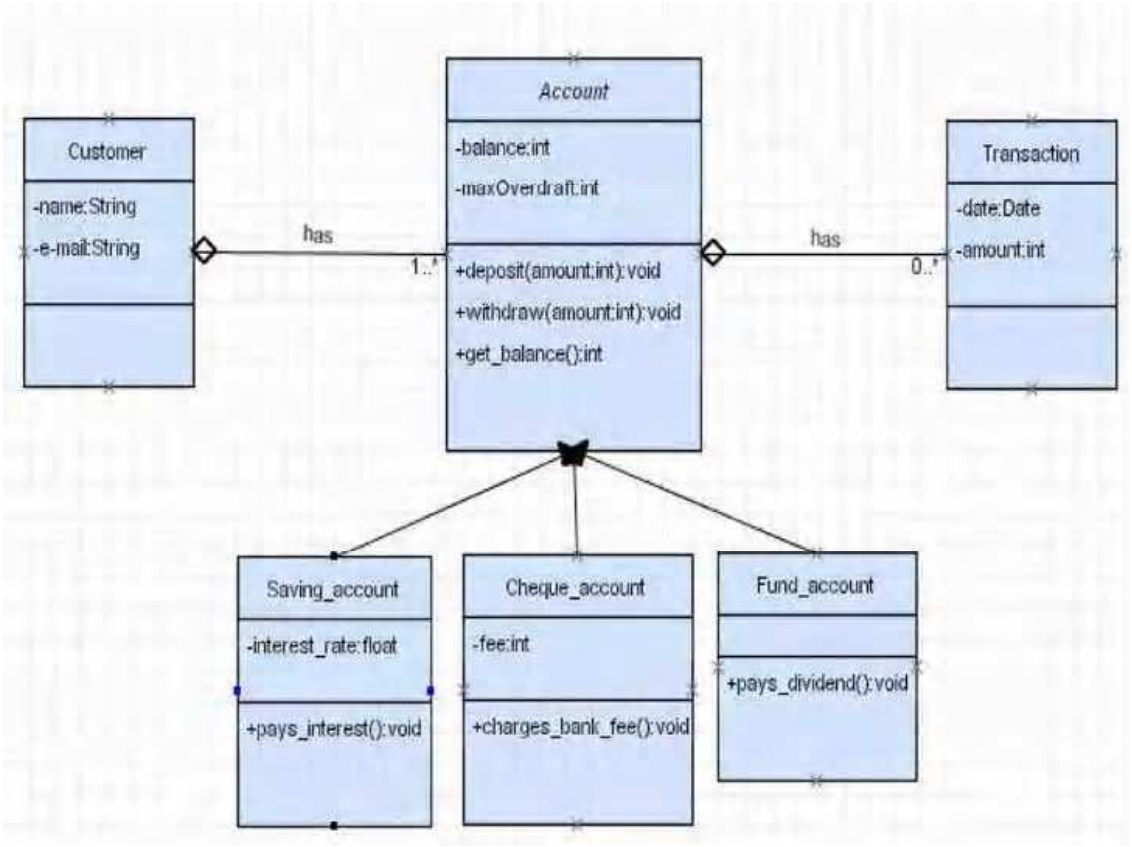
1. The experimental group students acquired more declarative knowledge on effective collaboration, as they scored significantly higher on the collaboration test, with the effect size of 1.3.
2. The collaboration skills of the experimental group students were better, as evidenced by these students being 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.




## References

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 intelligent tutoring system for UML. *Technology, Instruction, Cognition and Learning Journal, 4*(2), 113–137.

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.

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.

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).

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.

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., & 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.