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-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.

1. 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 2. Section 3 describes the overall architecture of the system, followed by the pilot study presented in Section 4. Conclusions and Future work are discussed in the last section.

2. 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.

3. 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.

3.1 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].

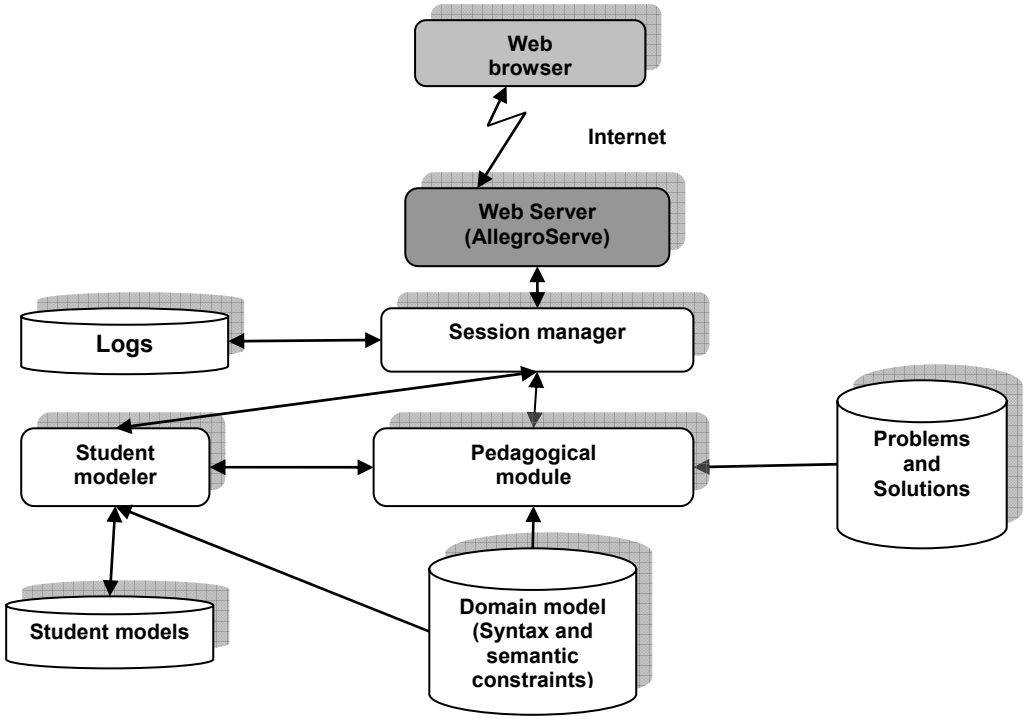

***Figure 1**: The architecture of the system*

3.2 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.

3.3 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

```
; 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*

and understood the problem. These mistakes would be minimised in COLLECT-*UML*, 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*

4. 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.

4.1 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 message in

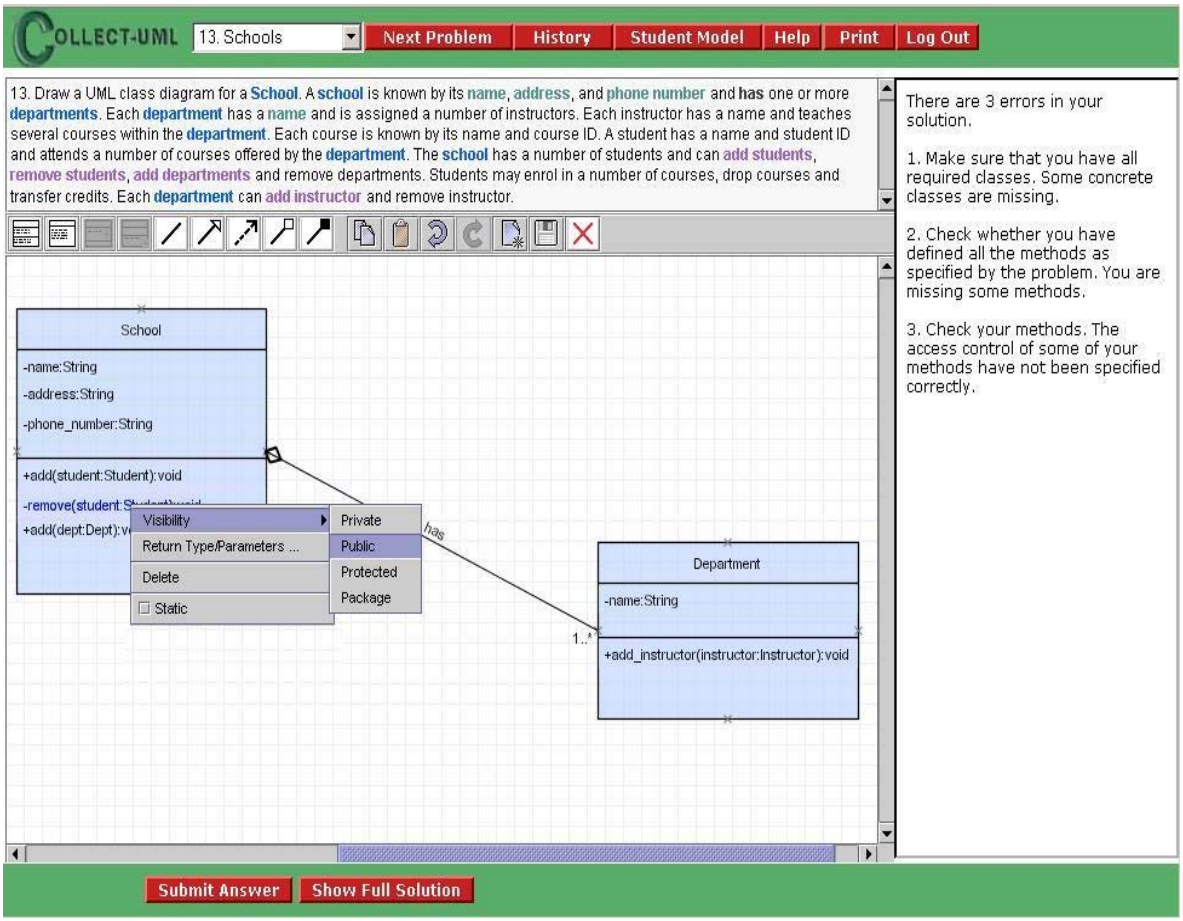

Figure 4: *The current version of* COLLECT-*UML interface*

this 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).

4.2 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.

5. 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.

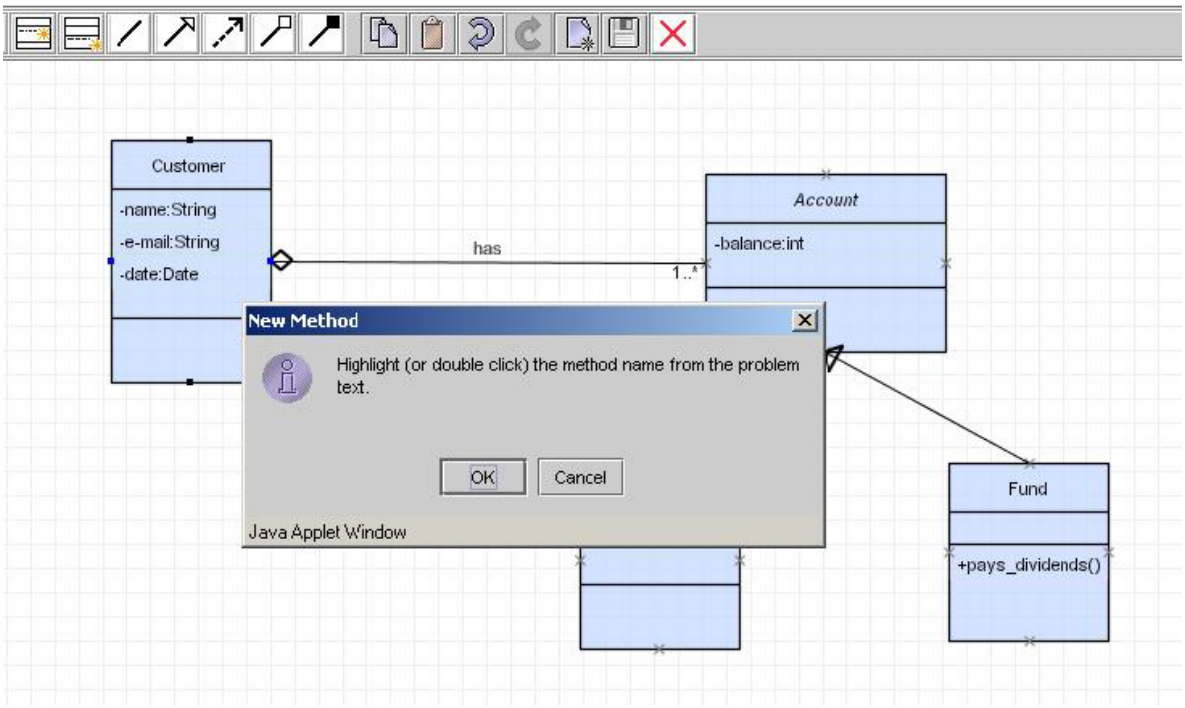

Figure 5: *An information dialog popping up to guide the users as to what they would need to do next*

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. *Protocol Analysis: Verbal Reports as Data*. The MIT Press, Cambridge, MA, 1984.

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

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

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

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

[9] Ohlsson, S. *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. *Knowledge acquisition for adaptive collaborative learning environments*. AAAI Fall Symposium: Learning How to Do Things, (Cape Cod, MA, 2000).

[11] 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, 2004.

[12] 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), (Biarritz, France, 2002), pp.377-387.