NORMIT: a Web-Enabled Tutor for Database Normalization

Antonija Mitrovic
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
tanja@cosc.canterbury.ac.nz

Abstract

The paper describes the design and development of NORMIT, an Intelligent Tutoring Systems (ITS) that teaches database normalization to university students. NORMIT is a Web-enabled system, and we discuss its architecture and techniques used to deal with multiple students. We also discuss Constraint-Based Modeling (CBM), the underlying student and domain modelling approach. NORMIT is the first in the series of constraint-based tutors developed at ICTG that teaches a procedural task, and we comment on the suitability of CBM for such tasks. We also discuss the plans for the evaluation of the system and future work.

1. Introduction

Web-enabled educational systems are becoming the dominant type of systems available to students. Web-based systems offer several advantages in comparison to standalone systems. They minimize the problems of distributing software to users and hardware/software compatibility. New releases of systems are immediately available to everyone. More importantly, students are not constrained to use specific machines in their schools, and can access Web-enabled tutors from any location and at any time. The time/location independence is of enormous value for learning environments, as flexibility and accessibility are extremely important for learning.

The Intelligent Computer Tutoring Group (ICTG) has been involved with developing intelligent tutoring systems for a number of years. We have developed two stand-alone systems: KERMIT [12], an ITS that teaches the conceptual database modeling using the Entity-Relationship data model, and CAPIT [6], a system that teaches English punctuation and capitalization rules. We have also developed two Web-enabled systems: SQLT-Web, an ITS that teaches the SQL database language [8, 9], and LBITS, a tutor that develops the language skills of elementary school children. All these systems use Constraint-Based Modeling [10] to model the domain knowledge and the knowledge of their students. The instructional domains covered by these systems differ significantly. CAPIT and LBITS cover domains with a small number of rules. SQL is a declarative database language which students find very difficult to master. KERMIT teaches an open-ended design task, and is based on fuzzy knowledge. Therefore, we decided to develop a system that teaches a procedural task, to see how well our existing methodology for building ITSs will support a different kind of tasks. We also decided to develop a Web-enabled system due to reasons discussed earlier.

The paper is organized as follows. In the next section we discuss the process of database normalization and how it is supported in NORMIT. In Section 3, we present the architecture of the system, focusing on the components necessary for dealing with multiple students. The final section presents our plan for the evaluation of the system, and discusses the future work.

2. Learning database normalization in NORMIT

Database normalization is the process of refining a database schema in order to ensure that all tables in a relational database are of high quality [4]. Normalization is usually taught in introductory database courses in a series of lectures that define all the necessary concepts, and later practised on paper by looking at specific databases and applying the definitions. To the best of our knowledge, there are no ITSs that support student learning database normalization, and NORMIT is novel in that respect.

The student needs to log on to NORMIT first, and the first-time user gets a brief description of the system and database normalization in general. NORMIT is a problem-solving environment, and as such provides only limited information about the task itself. We have envisioned the system as a complement to traditional classroom instruction, so the emphasis is on problem solving, not on providing information. However, the system does provide help about the basic domain

concepts, when there is evidence that the student does not understand them, or has problem applying knowledge. The system also insists on using the appropriate domain vocabulary; "talking science" has been shown to increase learning and deep understanding of the domain. After logging in, the student needs to select the problem to work on. NORMIT lists all the pre-defined problems, so that the student may select one that looks interesting. In addition, the student may enter his/her own problem to work on.

The database normalization task is a procedural one: the student should go through a number of steps to analyze the quality of the database schema. We require the student to go through the following steps in NORMIT:

1. *Determine candidate keys*: the student needs to analyze the given table and functional dependencies in order to determine all candidate keys. A candidate key is an attribute or a set of attributes that has two properties: uniqueness (its value is unique within the table) and irreducibility (no attribute can be removed from the key so that each value of the key is still unique. Figure 1 illustrates this task: the student is currently working on a table consisting of 5 attributes, for which five functional dependencies are given. The student enters the candidate keys one at a time, and may ask the system to evaluate the solution at any time.
2. *Determine the closure of a set of attributes*: if the student is unsure whether a set of attributes makes a candidate key, he/she may compute the closure of that set under the given set of functional dependencies.
3. *Determine the prime attributes* (a prime attribute is an attribute that belongs to any candidate key).
4. *Simplify functional dependencies*: if any of the given functional dependencies has more than one attribute on the right-hand side, the student needs to turn it into as many dependencies as there are attributes on its right-hand side (this step is the application of the decomposition rule).
5. *Determine the normal form* the table is in. During this task, when necessary, the student will also be asked to specify functional dependencies that violate one or more normal forms.

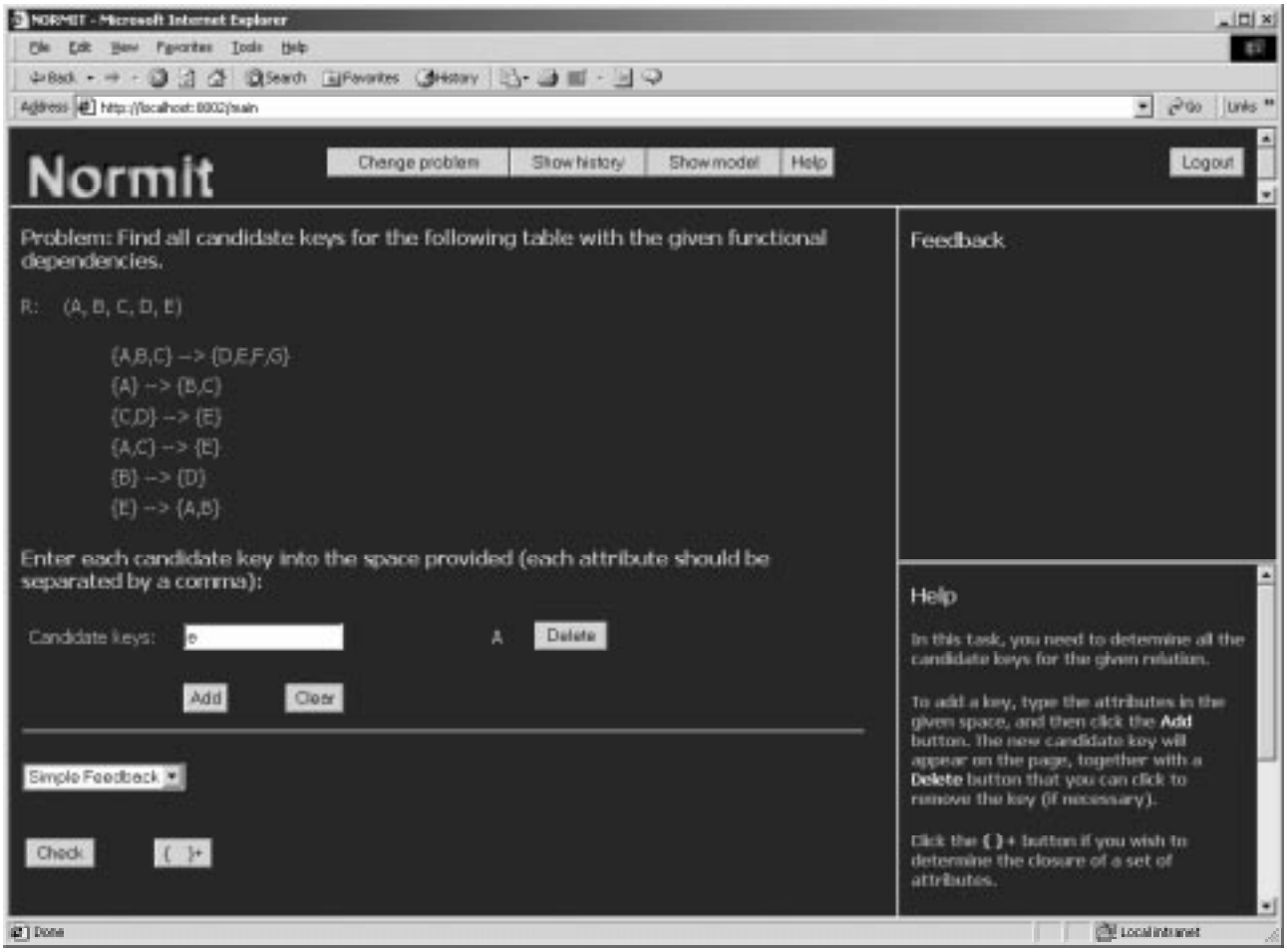

Fig. 1. A screenshot from NORMIT

6. If necessary, *decompose the table* so that all the final tables are in Boyce-Codd normal form.

The sequence of the steps is fixed: the student will only see a Web page corresponding to the current task. However, the student may ask for a new problem at any time during problem solving. In addition to that, he/she may review the history of the current session, or examine a global view of the student model. When the student submits the solution to the current step, the system analyses it and offers feedback. The first submission receives only a general feedback, specifying whether the solution is correct or not. If there are errors in the solution, the incorrect parts of the solution are shown in red. On the second submission, NORMIT provides a general description of the error, specifying what general domain principles have been violated. On the next submission, the system provides a more detailed message, by providing a hint as to how the student should change the solution. The correct solution is only available on student's request.

3. The architecture of NORMIT

Figure 2 illustrates the architecture of NORMIT. As can be seen, NORMIT is based on a centralized architecture, as many other existing Web-enabled ITSs (e.g. ELM-ART [3], AST [11] and SQLT-Web [7]). Centralized tutors perform all tutoring function on the server side, where all student models are also kept. Distributed systems (e.g. ADELE [5], AlgeBrain [2] or Belvedere [13]) also keep the student model on the central server, but some of the tutoring functions are performed on the client.

NORMIT is developed in Allegro Common Lisp (ACL) [1] and uses the AllegroServe Web server, which is an extensible server provided with ACL. 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 student who interacts

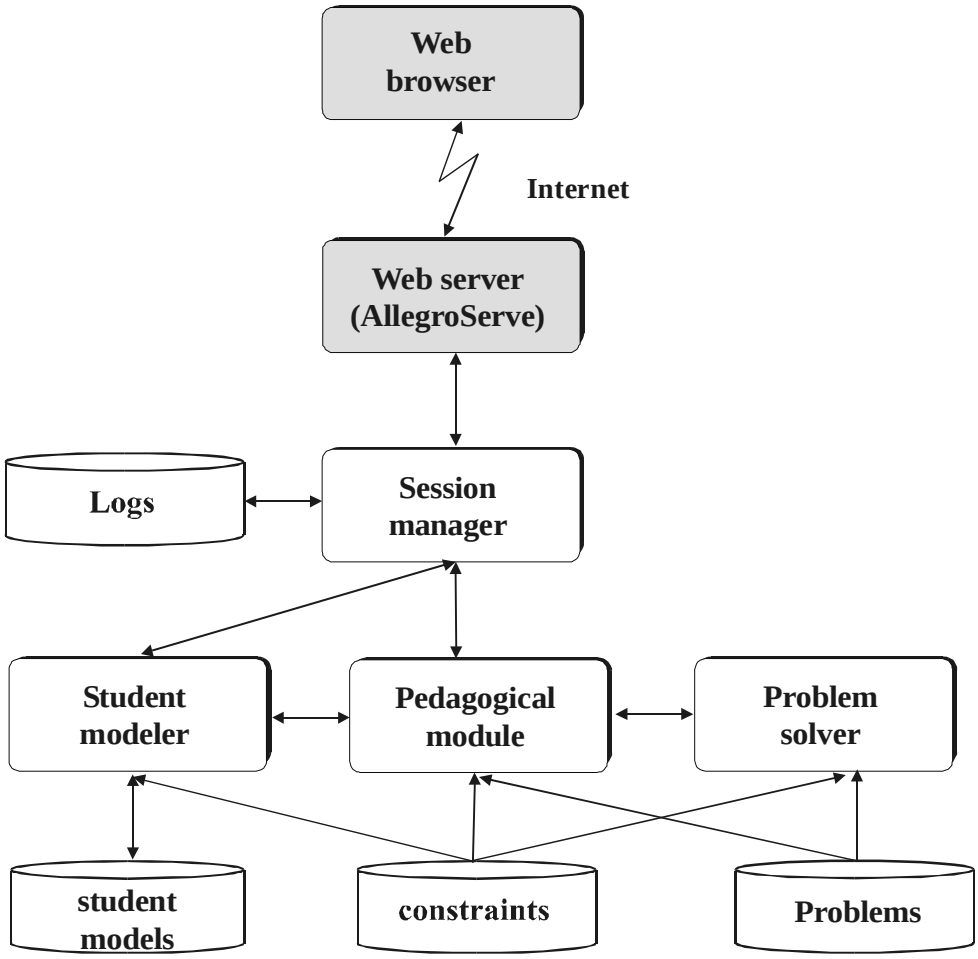

Fig. 2. The architecture of NORMIT

with the system for the first time. A Web-based tutor must be able to associate each request to the appropriate student model. Some Web-enabled systems use cookies or IP numbers to identify the student who made a request. Those two approaches were not suitable in our case. It was not possible to use the IP number, as several students might be using the same machine. We did not want to use cookies for identification purposes because cookies reside on a specific machine and would prevent the student from using the system from different machines. Instead, we identify students by their login name, which is embedded in a hidden tag of HTML forms and sent back to the server. If a student accesses a page by following a link instead of accessing it through a form, then user name is appended to the end of the URL.

It is also necessary to store student-specific data separately from data about other students. All processing is carried out within a single address space, and therefore there must be a uniform mechanism for identifying students and associating requests to corresponding student models. In order to achieve this, we use a hash table that maps the string representing a student name to their student object, which contains all details pertaining to the student.

Each action a student performs in the interface is first 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, which decides how to respond to it. If the submitted action is a solution to the current step, the pedagogical module sends it to the student modeller, which diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module. The pedagogical module then generates feedback. If the student has requested a new problem, the pedagogical module consults the student model in order to identify the knowledge elements the student has problems with, and selects one of the predefined problems that feature identified misconceptions.

The domain knowledge is represented as a set of constraints. Constraint-Based Modeling (CBM) is a student modeling approach proposed by Ohlsson [10], as a way of overcoming the intractable nature of student modeling. CBM starts from the observation that all correct solutions to a problem are similar in that they do not violate any of the basic principles of the domain. CBM is not interested in the exact sequence of states in the problem space the student has traversed, but in what state he/she is currently in. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. Constraints define equivalence classes of problem states. An equivalence class triggers the same instructional action; hence all states in an equivalence class are pedagogically equivalent. It is therefore possible to attach feedback messages directly to constraints. A violated constraint signals an error, which translates to incomplete/incorrect knowledge. The domain model is therefore 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."*

In other words, if the student solution falls into the state defined by the relevance condition, it must also be in the state defined by the satisfaction condition in order to be correct.

NORMIT currently contains 53 constraints, which are modular and problem-independent; they describe the basic principles of the domain, and do not involve any elements of problems directly. Some constraints check the syntax of the solution, while others check the semantics, by comparing the student's solution to the ideal solution, generated by the problem solver. The semantic constraints check whether the student has specified all the necessary parts of the solution. In order to identify constraints, we studied material in textbooks, such as [4], and also used our own experience in teaching database normalization. Figure 3 shows constraint 5, which specifies one condition the student's solution must satisfy when working on the closure task. The constraints are written in Lisp, and can contain built-in functions as well as specially developed functions. The first two lists of constraint 5 are its relevance and satisfaction condition. The relevance condition is a compound one: it firstly tests whether the current task the student is working is the closure task, and then it checks whether the student has specified the attribute set. Finally, it binds variable *a* to each attribute that appears in this set, thus forming a multiple binding list. The satisfaction part consists of a single test, which is applied to each binding of variable *a* separately. If the attribute appears in the closure, the constraint is satisfied. In the opposite case, the student will be given the appropriate feedback.

There are two feedback messages in the constraint, which are given to the student if his/her solution is incorrect. The first message is shorter, and tells the

```
(5
(and (equalp (current-task sol) 'closure)
     (not (null (attribute-set sol)))
     (bind-all ?a (attribute-set sol) bindings))
(member ?a (closure sol) :test 'equalp)
"Each attribute that is an element of the set of attributes
we want to compute the closure of must appear in the
closure."
"Remember the reflexivity rule? Each attributes
determines itself (A -> A).
The general form of the reflexivity rule is:
If X is a superset of Y, or X=Y, then X -> Y"
(?a "attribute-set"))
```

Fig. 3. An example constraint

student what needs to be done. If the student still cannot correct the solution after this message, NORMIT will present the second message, which explains the underlying domain principle that has been violated (in this case, it is the reflexivity rule). The final element of the constraint specifies the part of the solution that is incorrect (in this case, that is the attribute to which variable *a* is bound). This binding is used for highlighting the error.

4.Conclusions and future work

This paper presented the architecture and underlying philosophy of NORMIT, a Web-enabled ITS for teaching database normalization. NORMIT uses Constraint-based modelling to model domain knowledge and the knowledge of its students. However, unlike the previous tutors we developed, NORMIT is the first constraint-based tutor that teaches a procedural task. We have experienced no problems specifying constraints for such a task. The system contains a problem solver, capable of solving normalization problems. The knowledge base contains 53 constraints that check the syntax and semantics of students' solutions, enabling it to analyze all students' submissions. To analyze the semantics of solutions, NORMIT compares the student's solution to the ideal solution produced by the problem solver. The number of constraints is likely to be higher, as we are currently working on the decomposition task.

NORMIT is a Web-enabled system, with a centralized architecture. Student models are kept on the server, and all tutoring functions are also executed on the server. The amount of information that needs to be transferred from the browser to the server is not large, and we believe that such architecture is appropriate. NORMIT is developed in AllegroServe, an extensible Web server that allows the components of the system to be developed in Lisp. A special component of the system called the session manager ensures that a student's actions are associated with her/his student model, thus enabling the system to be used by multiple students simultaneously.

We plan to evaluate NORMIT in a real classroom in September 2002 at the University of Canterbury. The system will be used in an introductory database course, which has more than 170 enrolled students. We plan to compare the students' performance on a pre-test to their performance on a post-test, after using NORMIT. Information about all sessions will be recorded in logs, and we will analyze how students learn constraints, and also evaluate other types of support the system offers, such as the open student model and support for self-explanation.

Acknowledgements

The work presented here was supported by the Computer Science Department, University of Canterbury. We thank Li Chen for developing the interface.

References

1. Allegro Common Lisp, Franz Inc, 1998.
2. S. Alpert, M. Singley, P. Fairweather, Deploying Intelligent Tutors on the Web: an Architecture and an Example. *Int. J. Artificial Intelligence in Education*, 10, 1999, 183-197.
3. P. Brusilovsky, E. Schwarz, G. Weber, ELM-ART: an Intelligent Tutoring System on Wolrd Wide Web. In C. Frasson, G. Gauthier, A. Lesgold (eds), *Proc. 3rd Int. Conf. On Intelligent Tutoring Systems (ITS'96)*, Springer, LCNS 1086, 1996, 261-269.
4. R. Elmasri, S.B. Navathe, *Fundamentals of database systems*. Benjamin/Cummings, Redwood, 1994.
5. W.L. Johnson, E. Shaw, R. Ganeshan, Pedagogical Agents on the Web. Proc. ITS'98 Workshop on Intelligent Educational Systems on the Web, 1998.
6. M. Mayo, A. Mitrovic, Optimising ITS Behaviour with Bayesian Networks and Decision Theory'. *International Journal on Artificial Intelligence in Education*, 12(2), 2001, 124-153.
7. A. Mitrovic, K. Hausler, Porting SQL-Tutor to the Web. Proc. ITS'2000 workshop on *Adaptive and Intelligent Web-based Education Systems*, 2000, 37-44.
8. A. Mitrovic, B. Martin, M. Mayo, Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor. *User Modeling and User-Adapted Interaction*, 12(2-3), 2002, 243-279.
9. A. Mitrovic, S. Ohlsson, Evaluation of a constraint-based tutor for a database language, *Int. J. Artificial Intelligence in Education*, 10(3-4), 1999, 238-256.
10. S. Ohlsson, Constraint-based Student Modeling. In *Student Modeling: the Key to Individualized Knowledge--based Instruction*. Berlin: Springer-Verlag, 1994, 167-189.
11. M. Specht, G. Weber, S. Heitmeyer, V. Schoch, AST: Adaptive WWW-Courseware for Statistics. *Proc. Workshop on Adaptive Systems and User Modeling on the World Wide Web, UM-97*, 1997, 91-96.
12. P. Suraweera, A, Mitrovic, Designing an Intelligent Tutoring System for Database Modelling. In: M.J. Smith, G. Salvendy (eds) *Proc. 9th Int. Conf Human-Computer Interaction International (HCII 2001)*, New Orleans, vol. 2, 2001, 745-749.
13. D. Suthers, D. Jones, An Architecture for Intelligent Collaborative Educational Systems In: B. de Boulay, R. Mizoguchi (eds) *Artificial Intelligence in Education: Knowledge and Media in Learning Systems*. IOS, Amsterdam, 1997, 55-62.