# An Intelligent Tutoring System for Entity Relationship Modelling

Pramuditha Suraweera, Antonija Mitrovic

# An Intelligent Tutoring System for Entity Relationship Modelling

**Pramuditha Suraweera & Antonija Mitrovic,** *Intelligent Computer Tutoring Group, Computer Science Department*, *University of Canterbury, Private Bag 4800, Christchurch, New Zealand*
*{psu16,tanja}@cosc.canterbury.ac.nz*

**Abstract.** The paper presents KERMIT, a Knowledge-based Entity Relationship Modelling Intelligent Tutor. KERMIT is a problem-solving environment for the university-level students, in which they can practise conceptual database design using the Entity-Relationship data model. KERMIT uses Constraint-Based Modelling (CBM) to model the domain knowledge and generate student models. We have used CBM previously in tutors that teach SQL and English punctuation rules. The research presented in this paper is significant because we show that CBM can be used to support students learning design tasks, which are very different from domains we dealt with in earlier tutors. The paper describes the system's architecture and functionality. The system observes students' actions and adapts to their knowledge and learning abilities. KERMIT has been evaluated in the context of genuine teaching activities. We present the results of two evaluation studies with students taking database courses, which show that KERMIT is an effective system. The students have enjoyed the system's adaptability and found it a valuable asset to their learning.

## INTRODUCTION

Intelligent Tutoring Systems (ITS) have been proven to be very effective in domains that require extensive practice (Corbett et al., 1998; Koedinger et al., 1997; Mitrovic & Ohlsson, 1999). In this paper, we present KERMIT, a Knowledge-based Entity Relationship Modelling Intelligent Tutor. KERMIT (Suraweera & Mitrovic, 2001) is an ITS designed and implemented for teaching database modelling. It is developed as a problem-solving environment where the system presents a description of a scenario for which the student has to design a database.

This research is significant because it extends our study of Constraint-Based Modelling (CBM). In previous work we have shown that CBM is capable of supporting students' learning in two different domains: SQL, a declarative language, (Mitrovic & Ohlsson, 1999; Mitrovic et al., 2001) and punctuation and capitalization rules in English (Mayo & Mitrovic, 2001). Database design is a very different domain. As any other design domain, it involves open-ended tasks for which there are no well-formed problem solving algorithms. On the contrary, the learner is given an abstract definition of a good solution. In database modelling, a good solution is defined as an ER schema that matches the requirements, and satisfies all the integrity rules of the chosen data

model. As we show in section 2, these conditions are very vague. In this paper we show that CBM can be applied successfully to support design tasks too.

We start by briefly describing database design, and the Entity Relationship model used in KERMIT. Section 3 reviews related work. In section 4, we describe the overall architecture of the system, and present details of the user interface, knowledge base, student modeller and the pedagogical module.

The effectiveness and the students' perception of KERMIT were evaluated during two empirical evaluation studies. These two studies, presented in section 5, prove the effectiveness of the system for student's learning. Finally, we present the conclusions and the directions for future work in the last section.

## DIFFICULTIES OF LEARNING DATABASE MODELLING

Databases have become ubiquitous in today's information systems. Learning how to develop good quality databases is a core topic in Computer Science curriculum. Database design is a process of generating a model of a database using a specific data model. A model of a database (also known as a database schema) evolves through a series of phases. The initial phase of requirements analysis enables database designers to understand the application domain, and provides input for the conceptual design phase. In this phase, the designers reason about the application domain as described in the requirements, and use their world knowledge to discover important relationships between various data items of importance for the database. The conceptual schema of a database is a high-level description of the database and the integrities that data must satisfy. This high-level schema is later transformed into a logical schema (such as relational schema) which can be implemented on a Database Management System (DBMS), and finally into a physical schema, which contains details of data storage.

The quality of conceptual schemas is of critical importance for database systems. Most database courses teach conceptual database design using the Entity-Relationship (ER) model, a high-level data model originally proposed by Chen (1976). The ER model views the world as consisting of *entities*, and *relationships* between them. The entities may be physical or abstract objects, roles played by people, events, or anything else data should be stored about. Entities are described in terms of their important features, called *attributes* in the terminology of the ER model. Relationships represent various associations between entities, and also may have attributes.

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

> *You are to design a database to record details of artists and museums in which their paintings are displayed. For each painting, the database should store the size of the canvas, year painted, title and style. The nationality, date of birth and death of each artist must be recorded. For each museum, record details of its location and speciality, if it has one.*

From the description, it is obvious that artists, museums and paintings are of importance. Therefore, the student may start by drawing the entities first. Each entity is described in terms of

some attributes. For example, each painting would be described by its title, style and the size of canvas. All three attributes are explicitly mentioned in the requirements. For each artist, we need to know his/her name, nationality, date of birth and death. The artist's name, however, is not explicitly listed in the text. Finally, for each museum, we need to know its location and speciality.

The student also needs to identify the relationships between these three types of entities. Each painting is displayed in a museum, and this is mentioned in the first sentence of the problem text. However, the other necessary relationship is not mentioned explicitly: the relationship between the painting and the artist. The student needs world knowledge in order to identify this relationship.

Once when all concepts are identified, the student needs to determine the integrities of the model. Each entity type must have at least one key attribute, which uniquely identifies it. For the museum, that may be the museum name (which is not explicitly given in the text). Assuming there will be no two artists with the same name, the key attribute of ARTIST is his/her name. The key of PAINTING is its title, assuming each title is unique.

ER schema is usually presented in the graphical form, and the ER diagram for the museum database is illustrated in Figure 1. The ER model also contains two integrities defined on relationships. Each entity type that participates in a relationship can participate totally (shown by the double line on the diagram) or partially, and the number of instances participating in an instance of the relationship type is known as cardinality (shown by 1, N and M in Figure 1). The additional complexity is introduced by relationships possibly involving more than two entity types (higher degree relationships), and having simple, composite or multivalued attributes.

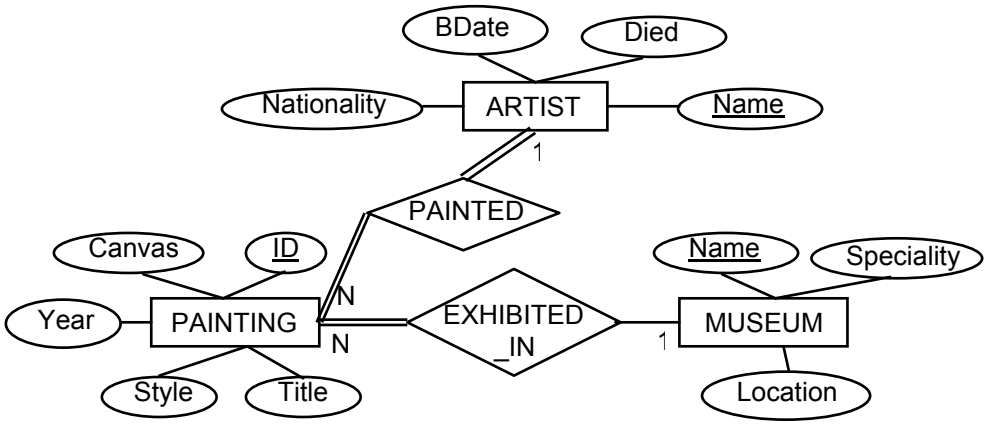



Fig. 1. The ER diagram for the Museum database

As can be seen from this simple case, there are many things that the student has to know and think about when developing an ER diagram. The student must understand the data model used, both the basic building blocks available and the integrity constraints specified on them. In real situations, the text of the problem would be much longer, often ambiguous and incomplete. To identify the integrities, the student must be able to reason about the requirements and use his/her own world knowledge to make valid assumptions. The ER modelling is not a well-defined process, and the task is open ended. There is no algorithm to use to derive the ER schema 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.

The authors have been involved in teaching database courses for a number of years, and in our experience students typically have many problems learning to design databases. Experiments conducted by Batra and colleagues (1990, 1994) have shown that although novices face little difficulty in modelling entities, they are challenged when modelling relationships. From our experiences, students find the concepts of weak entities and higher degree relationships difficult to grasp.

Although the traditional method of learning ER modelling in a classroom environment may be sufficient as an introduction to the concepts of database 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. The difficulties of targeting information at the appropriate level for each student's abilities also become apparent in group tutorials, with weaker students struggling and more able students not challenged sufficiently. Therefore, the existence of a computerized tutor, which would support students in acquiring database design skills, would be highly important.

## RELATED WORK: INTELLIGENT TUTORS FOR DB MODELLING

Educational systems with problem-solving environments for DB modelling can be used as educational tools for enhancing students' learning. Ideally these teaching systems would offer the student a vast array of practice problems with individualised assistance for solving them, allowing students to learn at their own pace. There have been very few research attempts at developing such teaching systems for DB modelling. This section outlines two such systems: ERM-VLE (Hall & Gordon, 1998) and COLER (Constantino-Gonzalez & Suthers, 2000; Constantino-Gonzalez et al., 2001).

### ERM-VLE: a virtual reality environment

ERM-VLE (Hall & Gordon, 1998) is a text-based virtual learning environment for ER modelling. In text-based virtual reality environments, users communicate with one another and with the virtual world exclusively through the medium of text. The objective of the learner in ERM-VLE is to model a database for a given problem by navigating the virtual world and manipulating objects. The virtual world consists of different types of rooms such as entity creation rooms and relationship creation rooms. The authors claim that the organisation of the environment reflects the task structure, and encourages a default order of navigation around the virtual world. The student issues commands such as *pick up*, *drop*, *name*, *evaluate*, *create* and *destroy* to manipulate objects. The effect of a command is determined by the location in which it was issued. For example, a student creates an entity whilst in the entity creation room. The evaluation command provides hints for modelling the ER schema.

The interface of ERM-VLE consists of several panes. The Scenario pane contains the requirements for the database being modelled. The 'Current ERM' pane provides a graphical representation of the ER model that the user is building. The graphical representation is

dynamically updated to reflect the activities of the student, but the student does not directly interact with the graphical representation. The student interacts with the virtual world solely by issuing textual commands. The 'ERM world' pane contains a record of past interactions between the student and the world.

The solution of each problem is embedded in the virtual world. Correspondences between the phases of the scenario and constructs of the ER model are stored in the solution. The learner is only allowed to establish the system's ideal correspondences. If the student attempts to establish an association that does not comply with the system's solution, the system intervenes and informs the student that the association is not allowed.

When the system was evaluated with a group of experienced DB designers and novices, the experienced designers felt that the structure of the virtual world had restricted them (Hall & Gordon, 1998). On the other hand, novices felt that they had increased their understanding of ER modelling. However, these comments cannot be treated as substantial evidence as to the effectiveness of the system since the system has not been evaluated properly.

ERM-VLE restricts the learner since he or she is forced to follow the identical solution path that is stored in the system. This method has a high tendency to encourage shallow learning as users are prevented from making errors and they are not given an explanation about their mistakes. Moreover, a text-based virtual reality environment is a highly unnatural environment in which to construct ER models. Students who learn to construct ER models using ERM-VLE would struggle to become accustomed to modelling databases outside the virtual environment.

**COLER: collaboratively building ER diagrams**

COLER (Constantino-Gonzalez & Suthers, 1999; Constantino-Gonzalez & Suthers, 2000; Constantino-Gonzalez et al., 2001) is a web-based collaborative learning environment for ER modelling. The main objectives of the system are to improve students' performance in ER modelling and to help them to develop collaborative and critical thinking skills. The system contains an intelligent coach that is aimed at enhancing the students' abilities in ER modelling. COLER is designed to enable interaction between students from different places via a networked environment to encourage collaboration.

COLER's interface contains a private workspace as well as a shared workspace. The student's individual solution is constructed in the private workspace, whereas the collaborative solution of the group of students is created in the shared workspace. The system contains a help feature that can be used to obtain information about ER modelling. The students are provided with a chat window through which they can communicate with other members of the group. Only a single member can edit the shared workspace at any time. Once any modifications are completed, another member of the group is given the opportunity to modify the shared workspace. The interface also contains an opinion panel, which shows the opinion of the group on the current issue. Each member has to vote on each opinion with either *agree*, *disagree* or *not sure*. The personal coach resident in the interface gives advice in the chat area based on the group dynamics: student participation and the group's ER model construction.

COLER is designed for students to initially solve the problem individually and then join a group to develop a group solution. The designers argue that this process helps to ensure that the students participate in discussions and that they have the necessary raw material for negotiating differences with other members of the group (Constantino-Gonzalez & Suthers, 2000;

Constantino-Gonzalez et al., 2001). The private workspace also allows the student to experiment with different solutions to a problem individually. Once a group of students agree to be involved in collaboratively solving a problem, the shared workspace is activated. After each change in the shared workspace, the students are required to express their opinions by voting.

COLER encourages and supervises collaboration, and we believe it has the potential in helping students to acquire collaboration skills. However, it does not evaluate the ER schemas produced, and cannot provide feedback regarding their correctness. In this regard, even though the system is effective as a collaboration tool, the system would not be an effective teaching system for a group of novices with the same level of expertise. From the authors' experience, it is very common for a group of students to agree on the same flawed argument. Accordingly, it is highly likely that groups of students unsupervised by an expert may learn flawed concepts of the domain. In order for COLER to be an effective teaching system, an expert should be present during the collaboration stage.

**Discussion**

Intelligent tutoring systems are developed with the goal of automating one-to-one human tutoring, which is the most effective mode of teaching. ITS offer greater flexibility in contrast to non-intelligent software tutors since they can adapt to each individual student. Empirical studies conducted to evaluate ITSs in other domains have shown vast improvements in student learning. Although ITSs have been proven to be effective in a number of domains, an effective ITS for DB modelling is yet to evolve.

ERM-VLE, the text based virtual reality environment for ER modelling, is a highly unnatural environment in which to construct ER models. Students would struggle to transfer their knowledge acquired using ERM-VLE to modelling databases for real life requirements. Furthermore, since the solutions are embedded in the virtual environment itself, students who have used the system have complained that it was too restrictive since they were forced to follow the ideal solution path. The method of forcing the user to follow the path of the system's solution has an increased risk of encouraging shallow learning.

The collaborative learning environment, COLER, is yet to undergo a comprehensive evaluation to test its effectiveness. COLER encourages and supervises collaboration with peers in collaboratively constructing an ER model. However, the system is not capable of evaluating the group solution and commenting on its correctness. The system assumes that the combined solution agreed upon by all the members of the group is correct. This assumption may not be valid for a group of novices with similar misconceptions about ER modelling. Consequently for COLER to be an effective teaching system, a human expert must participate in the modelling exercise.

## KERMIT: A KNOWLEDGE-BASED ER MODELLING TUTOR

KERMIT is an intelligent teaching system that assists students learning ER modelling. The system is designed as a complement to classroom teaching, and therefore we assume that the students are already familiar with the fundamentals of database theory. KERMIT is a problem-solving environment, in which students construct ER schemas that satisfy a given set of

requirements. The system assists students during problem solving and guides them towards the correct solution by providing feedback tailored towards each student depending on his or her knowledge.

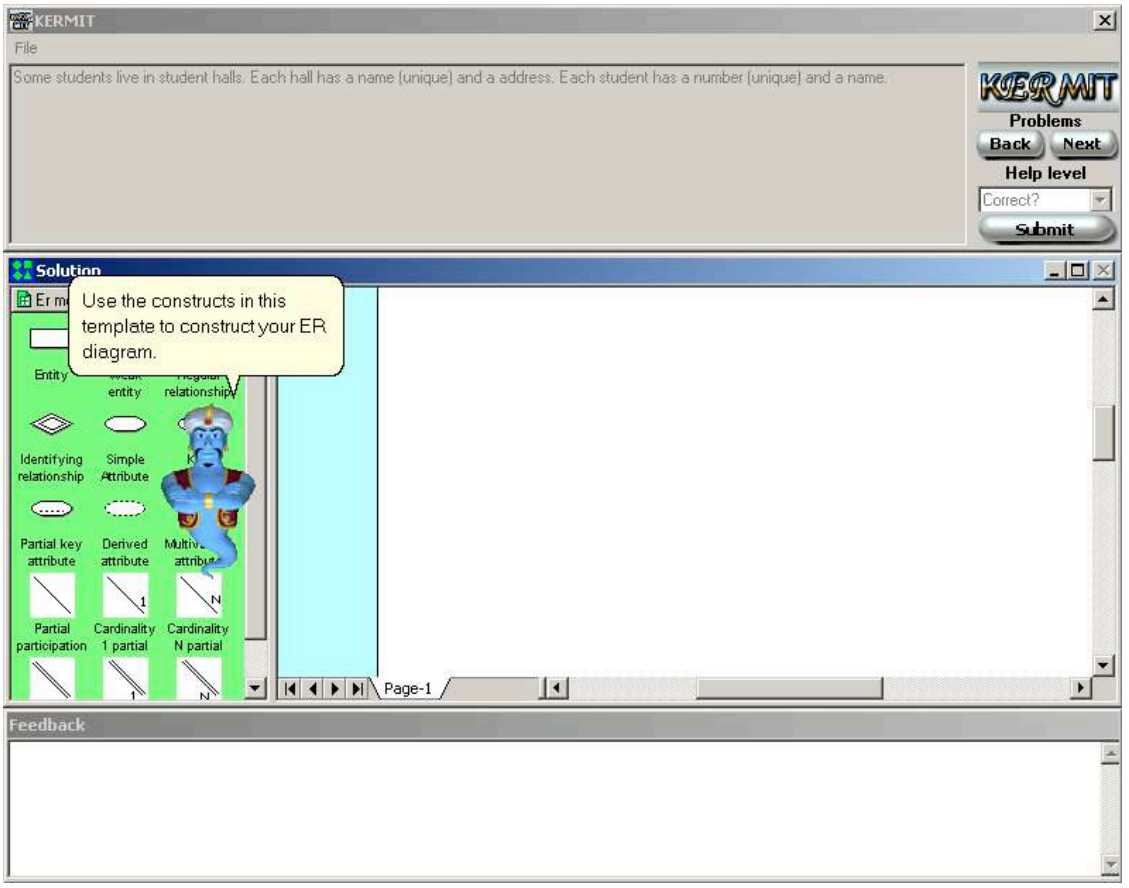



Fig. 2. A screenshot from the introduction to KERMIT

The system is designed for individual work. A student initially logs onto the system with an identifier. The system introduces its user interface, including its functionality, to first time users (Figure 2). During the problem solving stage, the student is given a textual description of the requirements of the database that should be modelled. The task is to use the ER modelling notation to construct an ER schema according to the given requirements. The ER model is constructed using the workspace integrated into KERMIT's interface. Once the student completes the problem or requires guidance from the system, their solution is evaluated by the system. Depending on the results of the evaluation, the system may either congratulate the student or offer hints on the student's errors. The student can request more detailed feedback messages depending on their needs. On completion of a problem, KERMIT selects a new problem that best suits the student's abilities. At the completion of a session with KERMIT, the student logs out.

KERMIT was developed using Microsoft Visual Basic to run on the Microsoft Windows platform. The teaching system was developed to support the Entity Relationship data model as defined by Elmasri and Navathe (Elmasri & Navathe, 2003). The following sections discuss KERMIT's design and implementation details.

**Architecture**

The main components of KERMIT are its user interface, pedagogical module and student modeller (Figure 3). Users interact with KERMIT's interface to construct ER schemas for the problems presented to them by the system. The pedagogical module drives the whole system by selecting the instructional messages to be presented to the student and selecting problems that best suit the particular student. The student modeller, which is implemented using constraint based modelling (Ohlsson, 1994), evaluates the student's solution against the system's knowledge base and records the student's knowledge of the domain in the form of a student model.

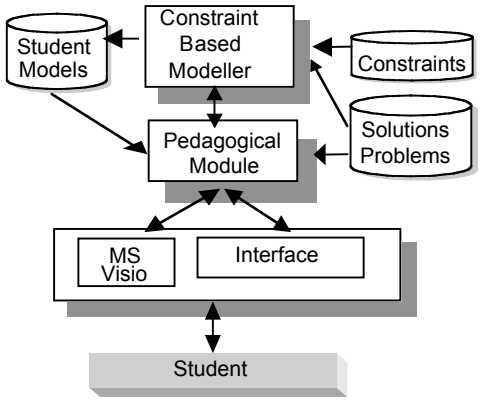



Fig. 3. Architecture of KERMIT

KERMIT does not have a domain module that is capable of solving the problems given to students. Developing a problem solver for ER modelling is an extremely difficult task. There are assumptions that need to be made during the composition of an ER schema. 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, ascertaining these assumptions is an essential part of the process of constructing a solution. Explicitly specifying the assumptions would over simplify the problems and result in students struggling to adjust to solving real world problems. Another complexity arises due to the fuzziness of the knowledge required in modelling a database. Consequently, developing a problem solver for database modelling would be difficult, if not entirely impossible.

Although there is no problem solver, KERMIT is able to diagnose students' solutions by using its domain knowledge represented as a set of constraints. The system contains an ideal solution for each of its problems, which is compared against the student's solution according to the system's knowledge base (see a later section for details on the knowledge base). The knowledge base, represented in a descriptive form, consists of constraints used for testing the student's solution for syntax errors and comparing it against the system's ideal solution.

KERMIT's knowledge base enables the system to identify student solutions that are identical to the system's ideal solution. More importantly, this knowledge also enables the system to identify alternative correct solutions, i.e. solutions that are correct but not identical to the system's solution.

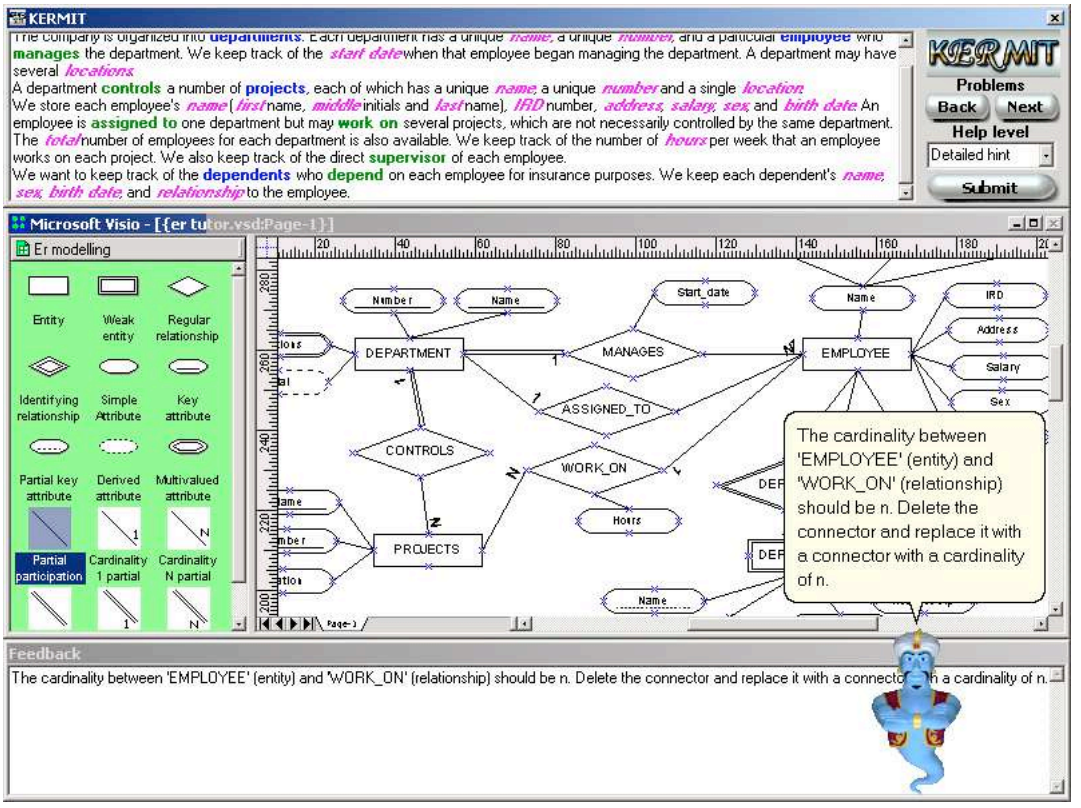



Fig. 4. User interface of KERMIT

**User interface**

Students interact with KERMIT via its user interface to view problems, construct ER diagrams, and view feedback. KERMIT's interface, illustrated in Figure 4, consists of four main components. The top window displays the textual description of the current problem. The middle window is the ER modelling workspace where students create ER diagrams. The lower window displays feedback from the system in textual form. The animated pedagogical agent (the Genie) that inhabits the learning environment presents feedback verbally incorporated with animations and speech bubbles.

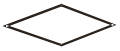
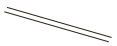
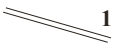
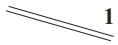
The top right corner of the interface contains the button for submitting solutions and obtaining feedback. The feedback content is dependent on the level of feedback. The student can either choose a specific level of feedback using a pull down list or can submit the solution again

to request more feedback. The next problem button can be used to request a new problem to work on and the system presents the student with a new problem that best suits the student's abilities.

*ER modelling workspace*

The workspace functions as an ER diagram-composing tool that assists students in constructing ER models. It is essential that the workspace is intuitive and flexible to use. It is also important that students do not feel restricted by the workspace and that its interface does not degrade their performance.

Table 1
Symbols available in KERMIT

| Symbol | Construct |
|---|---|
|  | **Regular entity** |
|  | **Weak entity** |
|  | **Regular relationship** |
|  | **Weak relationship** |
|  | **Simple attribute** |
|  | **Multivalued attribute** |
|  | **Key attribute** |
|  | **Partial key** |
|  | **Derived attribute** |
|  | **Simple connector, partial participation** |
|  | **Total participation connector** |
|  1 | **Partial participation with cardinality 1** |
|  N | **Partial participation with cardinality N** |
|  1 | **Total participation with cardinality 1** |
|  1 | **Total participation with cardinality N** |

Although the ER modelling workspace is not the main focus of our research, it is an important component of the system. During the design phase we explored the possibility of incorporating an effective commercial CASE tool, developed for database design. We selected *Microsoft Visio* (Visio), which is a diagram composing tool that allows users to create 2D diagrams by dragging and dropping objects from a template. We developed a set of objects for

ER modelling, and integrated *Visio* with KERMIT by using its comprehensive set of APIs that allow external programs to control *Visio* and to access its internal representation of the diagram.

The objects that the students can use to draw diagrams are shown on the left of the diagram, and are also given in Table 1. When creating an entity type, the student has to select either a regular or weak one. Similarly, when creating a relationship type, the student needs to use the appropriate object: diamond for a regular relationship, or a double diamond for an identifying relationship. There are four types of attributes to choose from: simple, multivalued, derived, key or partial key. To create a composite attribute, the student needs to create each component separately and then attach it to the attribute itself. Whenever a new object is created, the system asks for it to be named, by highlighting a phrase from the problem text. Figure 5 illustrates the state of the interface immediately after the student created a regular entity type. The student selects the name and the result of that action is shown in Figure 6.

Simple connector (single line) is used to connect attributes to other attributes, relationships or entities. To specify the constraints of relationships types, the student needs to select the appropriate connector from the four available ones.

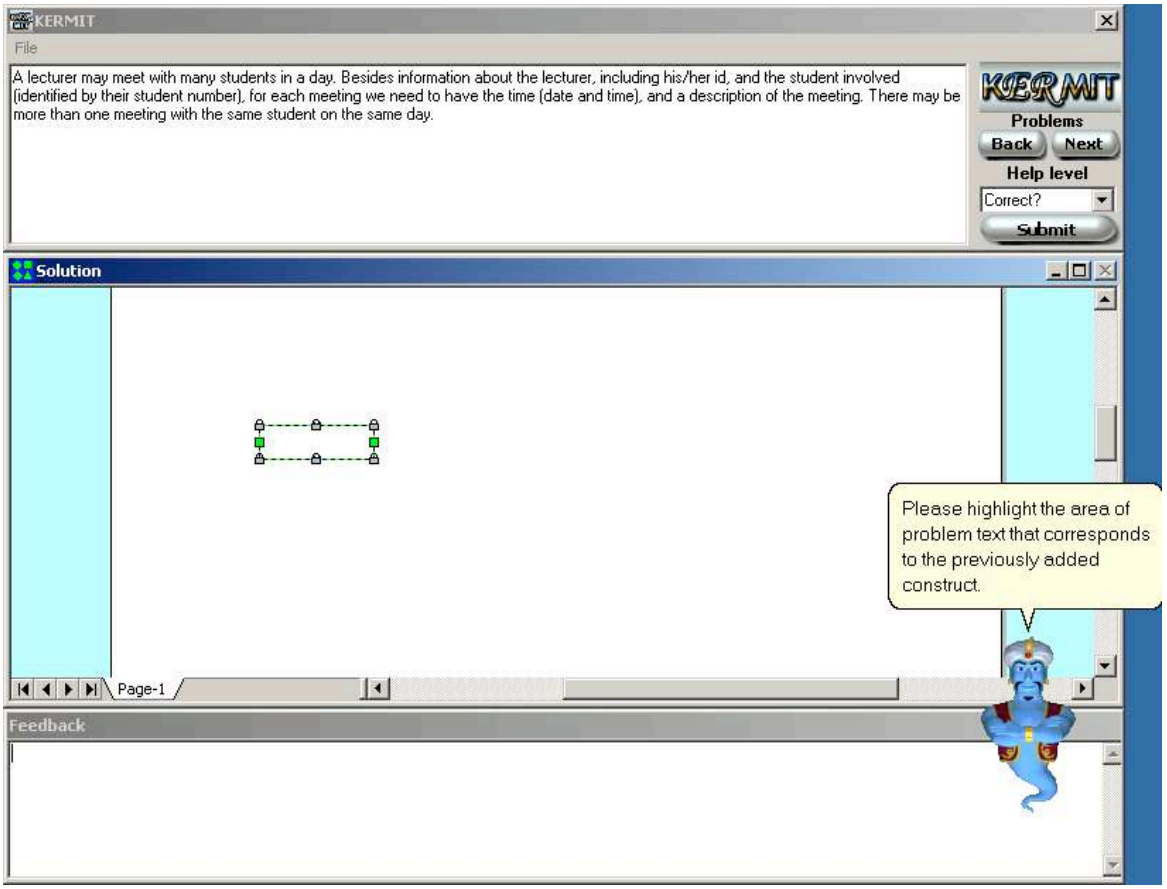



Fig. 5. The student creates an entity type

*Problem description*

Typical problems given to students in database modelling, such as those provided by KERMIT, involve modelling a database that satisfies a given set of requirements. The sample problem (Elmasri & Navathe, 2003) displayed in Figure 4 outlines the requirements of a database for a company. Students are required to construct ER models by closely following the given requirements. It is common practice with most students to either make notes regarding the problem or to underline phrases of the problem text that have been accounted for in their models. Some students highlight the words or phrases that correspond to entities, relationships and attributes using three different colours. This practice is very useful as they closely follow the problem text, which is essential to producing a complete solution.

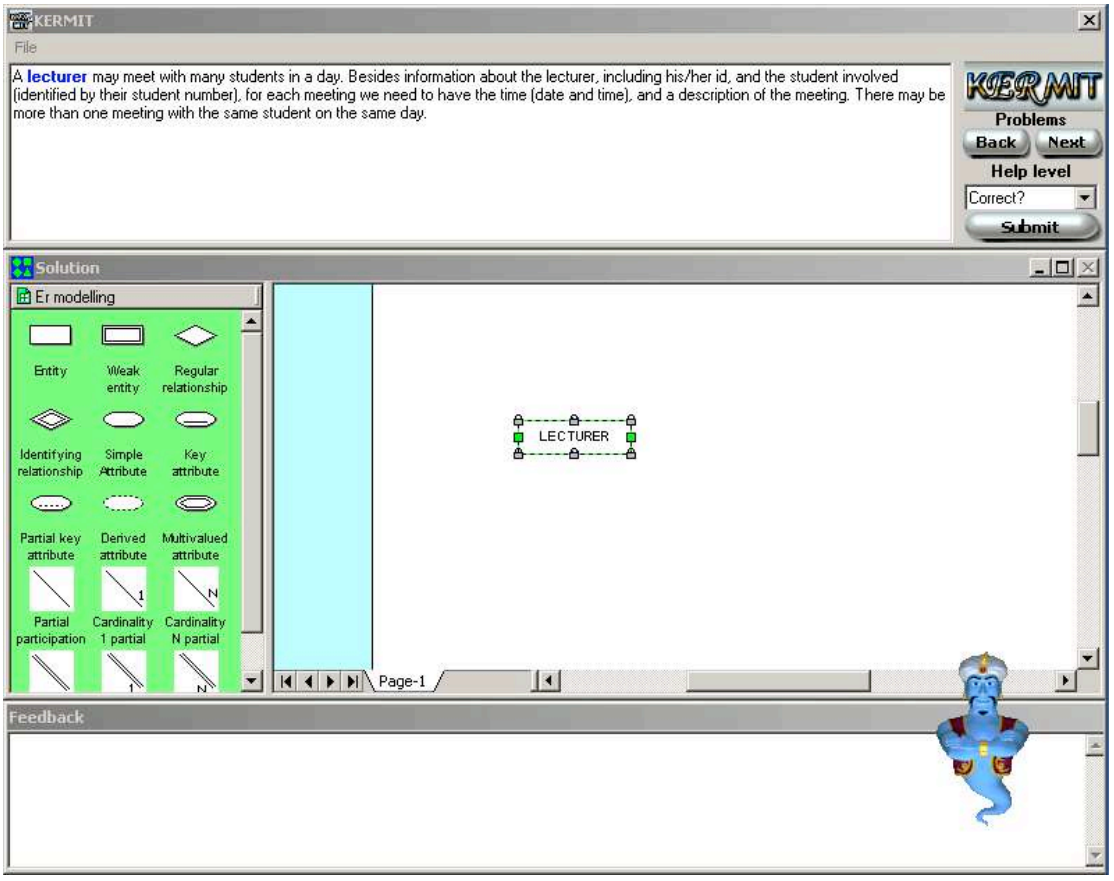



Fig. 6. The entity type with a name

KERMIT is designed to support this behaviour. Whenever a student creates a new object, the system requires that it be named, by selecting a word or a phrase from the problem text. The highlighted words are coloured depending on the type of object. When the student highlights a

phrase as an entity name, the highlighted text turns bold and blue. Similarly the highlighted text turns green for relationships and pink for attributes.

This feature is extremely useful from the point of view of the student modeller for evaluating solutions. There is no standard that is enforced in naming entities, relationships or attributes, and the student has the freedom to use any synonym or similar word/phrase as the name of a particular object. Since the names of the objects in the student solution (SS) may not match the names of construct in the ideal solution (IS), the task of finding a correspondence between the constructs of the SS and IS is difficult. For example, one may use 'HAS' as the relationship name, while another may use 'CONSISTS_OF'. Even resolutions such as looking up a thesaurus will not be accurate as the student has the freedom to name constructs using any preferred word or phrase. This problem is avoided in KERMIT by forcing the student to highlight the word or phrase that is modelled by each object in the ER diagram. The system uses a one-to-one mapping of words of the problem text to the objects of its ideal solution to identify the corresponding SS objects.

The feature of forcing the student to highlight parts of the problem text is also 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 KERMIT, as students are required to focus their attention on the problem text every time they add a new object to the diagram. Moreover, the student can make use of the colour coding to ascertain the number of requirements that they have already modelled.

*Feedback presentation*

The goal of the feedback is to improve the student's knowledge of the domain, and therefore it is essential that these messages are presented in an effective manner. KERMIT presents feedback in two ways: using an animated agent and in a conventional text box.

Animated pedagogical agents are animated characters that support students' learning. Studies have shown that animated pedagogical agents have a strong positive effect on students' motivation and learning (Lester et al., 1999; Johnson et al., 2000; Mitrovic & Suraweera, 2000). The interface of KERMIT is equipped with such an agent (the Genie) that presents instructional messages verbally and displays a strong visual presence using its animations, which are expected to contribute towards enhanced understanding and motivation levels. The Genie provides advice to the students, encourages them and enhances the credibility of the system by the use of emotive facial expressions and body movements, which are designed to be appealing to students. In general, the Genie adds a new dimension to the system, making it a more interesting teaching tool.

The Genie was implemented using Microsoft Agent (Microsoft) technology. The MS agent can be easily incorporated, as it is equipped with a comprehensive set of APIs. KERMIT uses these to control the Genie, which performs animated behaviours such as congratulating, greeting and explaining. The Genie is very effective in introducing KERMIT's interface to a new student. It moves around the interface pointing to elements in the interface, explaining their functionality. The goal of this initial introduction is to familiarise new students with the interface.

Although the animated agent presents the feedback, that text disappears once the agent completes its speech. However, the student may find it useful to refer to the feedback later on while constructing the ER model. For this reason, the interface also contains a static text box (the

bottom part of the interface) that displays the feedback message until the student re-submits the solution. This is especially useful in situations where more than one error has been identified, and feedback addresses all errors, when students need to refer back to the feedback. From the pedagogical point of view, this also reduces the cognitive load, since students do not have to remember the feedback that was presented by the agent.

**Problems and Ideal Solutions**

KERMIT contains predefined database problems, each with a natural language description of requirements, and the ideal solution, specified by a human database expert. In this section we describe the way problems and solutions are represented in the system.

*Internal representation of solutions*

In order to evaluate the student's solution efficiently, it is essential for KERMIT to represent it internally in a convenient form. Since KERMIT incorporates *MS Visio* as its ER modelling workspace, it either must use *MS Visio*'s internal representation of the diagram or maintain its own representation. *MS Visio* represents each object in the diagram as a generic object, and therefore does not distinguish between the different types of ER objects. All the objects in a diagram are arranged in a list, in the order in which they were added to the workspace.

Accessing *Visio*'s internal representation for evaluating a student's solution against the constraint base is inefficient since constraints naturally deal with a particular group of objects such as entities or relationships. Each time a constraint in the knowledge base is evaluated, the list of objects in *Visio* that represents the internal structure of the diagram would have to be sequentially scanned to extract the particular group of constructs. This method is inefficient and would increase the response time. Moreover, accessing *Visio*'s internal data structures through its APIs using a VB program is slower than accessing a data structure maintained by the VB program itself. Due to these drawbacks, KERMIT dynamically maintains its own internal representation of the ER diagram.

KERMIT maintains two lists of objects: one for entities and one for relationships. The attributes are contained as a list within the entity or relationship object to which they belong. Attributes that are components of a composite attribute are stored as a list within their parent attribute. Each relationship has a list of participating entity objects that keeps track of its participating entities.

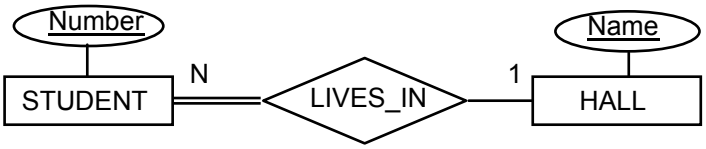



Fig. 7. The ER schema for the scenario "Students live in halls"

The procedure of building the internal representation of the student's solution is based on the student's interactions. When the student adds a new object to the workspace, the system adds information about it into a corresponding list. All syntactically illegal constructs are recorded in a

separate list. For example, when an attribute is added to the workspace, it will be included in the illegal constructs list until it is connected to an entity or relationship. Once the attribute becomes part of an entity or relationship, it is added to the list of attributes of the entity or relationship to which it belongs.

KERMIT contains ideal solutions for all of the problems in its database. These ideal solutions (IS) are stored in a compact textual form. When a new problem is given to the student, the system reads its stored solution, parses it and builds a runtime internal representation in the memory. An ideal solution is also represented internally with objects grouped using lists during runtime, similar to the SS. KERMIT uses this representation of the IS to compare it to the SS according to its constraint base.

The stored textual version of the IS consists of two lists: an entities list and a relationships list. The entities list consists of the names of all the entities including their attributes. The entity name is followed by the attributes, which are listed within parentheses. In the case of composite attributes, the components of the attribute are listed enclosed by '|' symbols. As an example consider the entities list of the ideal solution to the scenario "Students live in halls". The correct ER schema is depicted in Figure 7, and its textual representation is outlined in Figure 8.

**Entities** = "STUDENT<E1>(Number<K1>),HALL<E2>(Name<K1>)"

**Relationships** = "LIVES_IN<R1>()-<E1>np,<E2>1t-"

Fig. 8. Internal representation of the ideal solution for the scenario "Students live in halls"

Each object in a solution is assigned a unique identifier, composed of a single character code that specifies the type of the object and an integer. For example, 'E' is used for regular entities and 'W' for weak entities. The integer in the ID makes it unique. The solution in Figure 7 contains two entities, namely STUDENT and HALL. The former entity has the identifier of E1 (see Figure 8), as it is the first entity that has been created. Its key attribute is Number, and its identifier is K1. There is only one relationship, LIVES_IN, whose ID is R1. This relationship has no attributes, and the ids of participating entities are also represented. E1 (i.e. the STUDENT entity type) participates partially in this relationship, and there may be many instances of STUDENT participating in the relationship (there can be many students living in a hall); these constraints are represented by *np* in Figure 8. Similarly, the HALL entity type participates totally, and there is one hall per student; this is represented by *1p*.

*Internal representation of problems*

The text of each problem describes the requirements of a database that the student is to design. Since KERMIT does not possess any NLP abilities, we developed an effective internal representation that enabled the system to identify the semantic meaning of each object. As discussed previously, KERMIT forces students to highlight the word or phrase modelled by each object in their solutions. The system uses these highlighted words to form correspondences between the student's and the ideal solution.

The problem text is represented internally with embedded tags that specify the mapping of its important words to the IS objects. These tags have a many-to-one mapping to the objects in

the ideal solution. In other words, more than one word may map to a particular object in the ideal solution to account for duplicate words or words with similar meaning. The set of tags embedded in the problem text are identical to the tags assigned to the objects of the ideal solution. The tags are specified by a human expert when the problem is added to the system. They are not visible to the student since they are extracted before the problem is displayed. The position of each tag is recoded in a lookup table, which is used for the mapping exercise. Whenever the student creates an object, the corresponding tag from the problem text matching the name of the object is used to relate the object in the student's solution to the appropriate object in the ideal solution.

**Knowledge base**

The knowledge base is an integral part of any ITS. The quality of the pedagogical instructions provided depends critically on the knowledge base. As stated previously, KERMIT uses CBM to model the domain knowledge and generate student models. In previous work (Mayo & Mitrovic, 2001; Mitrovic & Ohlsson, 1999, Mitrovic et al. 2001, 2002) we have discussed CBM and its implementations in SQL-Tutor and CAPIT. Each constraint specifies a fundamental property of a domain that must be satisfied by any correct solution. Constraints are modular and problem-independent, and therefore easy to evaluate. Each constraint is a collection of patterns, and therefore the evaluation process is very efficient computationally.

One of the advantages of CBM over other student modelling approaches is in its independence from the problem-solving strategy employed by the student. CBM models students' evaluative, rather than generative knowledge and therefore does not attempt to induce the student's problem-solving strategy. CBM does not require an executable domain model, and therefore is applicable in situations in which such a model would be difficult to construct (such as database design or SQL query generation).

Furthermore, CBM eliminates the need for bug libraries, i.e. collections of typical errors made by students. Cognitive tutors contain buggy rules and use them to provide error-specific feedback to students (Anderson et al., 1996; Corbett et al., 1998). When a student's action matches one of the misconceptions from a bug library, a teaching system based on a bug library responds by offering an explanation of why the action is incorrect. Bug libraries are extremely difficult to acquire, as it is necessary to collect and analyse numerous instructional sessions in order to identify the misconceptions, a time-consuming and error-prone process. Secondly, the effectiveness of the system would depend on the completeness and correctness of the bug library. If a specific misconception is missing from the bug library, the system will be able to inform the student that his/her action is incorrect, but will not be able to provide error-specific feedback. Thirdly, bug libraries do not transfer well between different populations of students (Payne & Squibb, 1990). If a bug library is developed for a specific group of students, other populations may have different misconceptions, and therefore the effort in developing a bug library is not justified. (Please see (Mitrovic, Koedinger & Martin, 2003) for a comparison of CBM and cognitive tutoring)

On the contrary, CBM focuses on correct knowledge only. If a student performs an incorrect action, that action will violate some constraints. Therefore, a CBM-based tutor can react to misconceptions although it does not represent them explicitly. A violated constraint means that student's knowledge is incomplete/incorrect, and the system can respond by generating an appropriate feedback message. Feedback messages are attached to the constraints directly, and

they explain the general principle violated by the student's actions. Feedback can be made very detailed, by instantiating parts of it according to the student's action.

The domain knowledge of KERMIT is represented as a set of constraints used for testing the student's solution for syntax errors and comparing it to the ideal solution. Currently KERMIT's knowledge base consists of 92 constraints. Each constraint consists of a relevance condition, a satisfaction condition and feedback messages. The feedback messages are used to compose hints that are presented to the students when the constraint is violated.

The constraints in the knowledge base have to be specified in a formal language that can be parsed and interpreted by the system. It is imperative that the formal representation is expressive enough to test the subtle features of student solutions and compare them to ideal solutions. We have chosen a simple Lisp-like functional language. It contains a variety of functions such as 'unique', 'join' and 'exists'. The lists of entities and relationships are addressed using aliases (e.g. 'SSE' is used for the list of entities of the SS). More examples of the internal representations of the constraints can be found in the following sections.

In this section, we describe the process of acquiring constraints and then present examples of syntactic and semantic constraints. Finally, we discuss how the constraints are used to diagnose students' solutions.

*Knowledge acquisition*

It is well known that knowledge acquisition is a very slow, labour intensive and time consuming process. Anderson's group have estimated that ten hours or more is required to produce a production rule (Anderson et al., 1996). Although there is no clear-cut procedure that can be followed to identify constraints, this section discusses the paths that were explored in discovering the constraints of KERMIT's knowledge base.

Most syntactic constraints of KERMIT were formulated by analysing the target domain of ER modelling through the literature (Elmasri & Navathe 2003). Due to the nature of the domain, the acquisition of syntactic constraints was not straightforward. Since ER modelling is an ill-defined domain, descriptions of its syntax in textbooks are informal. This process was conducted as an iterative exercise in which the syntax outline was refined by adding new constraints. Semantic constraints are harder to formulate. We analysed sample ER diagrams and compared them against their problem specifications to formulate basic semantic constraints.

```
id         = 10
relCond    = "t"
satCond    = "unique (join (SSE, SSR))"
Fedback1   = "Check the names of your entities and relationships. They must be unique."
Feedbck2   = "The name of <viol> is not unique. All entity and relationship names must be
       unique."
Feedback3 = "The names of <viol> are  not  unique. All entity and relationship names must be
       unique."
construct = "entRel"
conceptID = 1
```

Fig. 9. Constraint 10

*Syntactic constraints*

The syntactic constraints describe the syntactically valid ER schemas and are used to identify syntax errors in students' solutions. These constraints only deal with the student's solution. They vary from simple constraints such as "an entity name should be in upper case", to more complex constraints such as "the participation of a weak entity in the identifying relationship should be total".

Constraint 10, presented in Figure 9, is a syntactic constraint. It specifies that all names of entities and relationships should be unique. The relevance condition (*relCond*) of this constraint is set to *true* (denoted by *t*) and is always satisfied, meaning that this constraint is relevant to all student solutions. Its satisfaction condition checks that all names the student has assigned to entities and relationships are unique. In addition to the relevance and satisfaction conditions, each constraint contains three messages (*feedback*, *feedback1*, *feedback2*) that are used to generate feedback when the student violates the constraint. The first message is general, and is used to give hints to students. The other two messages are used as templates for generating detailed feedback messages. During the generation of feedback, the <viol> tag embedded in the message is replaced with the names of the constructs that have violated the constraint. *Feedback1* is singular and is used for situations where a single construct has violated the constraint, whereas *feedback2* is plural and is used for cases where many constructs of the solution have violated the constraint. The types of constructs that violate the constraint are given as the *construct* attribute. In this example, the type of violated constructs can be either an entity or a relationship (denoted by *entRel*). The construct attribute is used in generating a very general feedback message that specifies the type of constructs that contain errors.

Table 2
Concepts covered by constraints

| ID | Concept |
|---|---|
| 0 | Syntax and notation |
| 1 | Regular entity types |
| 2 | Weak entity types |
| 3 | Isolated regular entity types |
| 4 | Regular relationship types |
| 5 | Identifying relationship types |
| 6 | n-ary regular relationship types |
| 7 | Simple attributes |
| 8 | Key attributes |
| 9 | Partial key attributes |
| 10 | Derived attributes |
| 11 | Multivalued attributes |
| 12 | Composite attributes |
| 13 | Multivalued composite attributes |

The constraints are organized into fourteen concepts, given in Table 2. The concepts cover different types of objects, their different arrangements and notation and syntax rules. The concepts are used for problem selection. When selecting a problem for the student, the system

decided on the concept first, and then selects a problem for the chosen concept. Each constraint is assigned to only one concept. The specific concept that the constraint deals with is specified as the *conceptID* of the constraint. Constraint 10 belongs to the concept with the identifier of 1, which is 'regular entities'.

*Semantic constraints*

Semantic constraints compare the student's solution to the ideal one. These constraints are usually more complex than syntactic constraints. Constraint 67 (Figure 10) deals with composite multivalued attributes, which are equivalent to weak entities. A weak entity type is used to model a set of entities that do not have an identifying attribute. Instead, such entities can be identified through their relationship with a regular entity type. For example, in the company database, the employee entity type is a regular one, as each employee can be identified by his/her (unique) employee number. The same database stores information about dependents of employees, but there are no unique attributes defined for them. Each dependent can be uniquely identified by his/her name, and the ID of the employee who supports the dependent.

```
id      = 67
relCond = "each obj ISE
        (and (notNull (matchSS (obj)))
        (and (= type (obj), type (matchSS (obj)))
                (> (countOfType (attributes (obj), mComp), 0)))))"
satCond = "each obj RELVNT
                (each att (ofType (attributes (obj)), mComp)
                        (or (and (notNull (matchAtt (att, (matchSS (obj)))))
                                (= (type (matchAtt (att, (matchSS (obj)))) mComp))
                           (and (and (notNull (matchSS (att))) (= (type att) w))
                                   (belongs (matchSS (att), obj)))))"
Feedback1 = "Check whether your entities have all the required multivalued composite attributes.
        Your entities are missing some multivalued composite attributes. You can represent composite
        multivalued attributes as weak entities as well."
Feedback2 = "The <viol> entity is missing some composite multivalued attributes. You can
        represent composite multivalued attributes as weak entities as well."
Feedback3 = "<viol> entities are missing some composite multivalued attributes. You can represent
        composite multivalued attributes as weak entities as well."
construct = "ent"
conceptID = 13
```

Fig. 10. Constraint 67

As stated above, composite multivalued attributes can be modelled alternatively as weak entities. If the ideal solution contains a composite multivalued attribute, constraint 67 compares it to a composite multivalued attribute in the student's solution, or to a weak entity. The relevant objects include IS entities, which possess a multivalued composite attribute, for which there is a corresponding entity in the SS. The constraint is satisfied if all the corresponding entities in the SS have a matching multivalued composite attribute (of type *mComp*) or a matching weak entity. This constraint illustrates the ability of the system to deal with alternative correct student

solutions that are different from the IS specified by a human expert. KERMIT knows about equivalent ways of solving problems, and it is this feature of the knowledge base that gives KERMIT considerable flexibility.

If constraint 67 is violated, the student will get the general feedback message (*feedback1*) first. If the student needs more help to correct the error, a more detailed message (*feedback2* or *feedback3*, depending on the number of constructs that violate the constraint) will be given. Please note that the <*viol*> variable will be replaced by the name of the entity that violates constraint 67, thus providing a reference to the student.

The equivalent solutions identified by Constraint 67 are illustrated in Figure 11. The entity *E1*, in the schema labelled (a), is the owner of the weak entity *W1*. According to the cardinality of the entity *E1* in the identifying relationship *I1*, *E1* may own a number of *W1* weak entities. Schema (a) is equivalent to Schema (b), where *W1* is represented as a multivalued attribute of *E1*. All attributes of the weak entity *W1* (e.g. *A1*) are represented as components of the multivalued attribute *W1*. In this scenario, even though the same database can be modelled in two different ways, KERMIT is only given one schema as its ideal solution. KERMIT's constraints are able to identify that the other schema is also an equivalent representation of the solution.

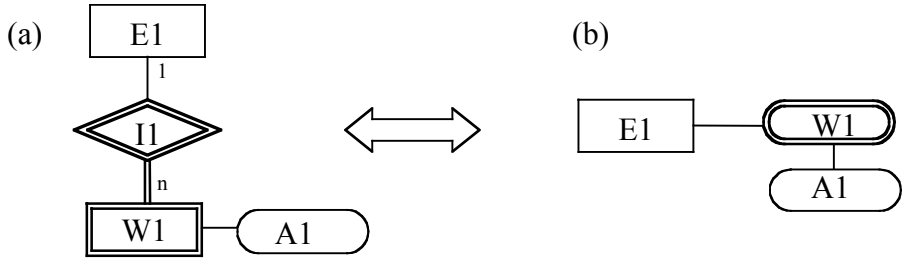



Fig. 11. A weak entity can also be represented as a multivalued attribute

The ability to identify alternative correct solutions is very important, especially in domains where problem-solving strategies do not exist and therefore a problem solver is not available. In the ER domain, the differences between alternative solutions are small, as illustrated on the example of constraint 67. However, in other domains, these differences may be greater. In our previous work (Mitrovic & Ohlsson, 1999), SQL-Tutor was also able to recognise alternatives to the pre-specified correct solution, as constraints checked whether the student included all the necessary elements of the solution in any allowable form. In the SQL domain, the differences between alternative queries may be substantial. There may be many constraints that are necessary to ensure that two solutions are equivalent. In domains where there are many alternative approaches to solving problems, constraints will be more complex, and the process of generating the constraint set will be more demanding. However, the ability to recognize alternative solutions only depends on the correctness and the completeness of the constraint base. If the constraint set is incomplete/incorrect, the system may not be able to identify equivalent ways of solving the same problem, and in such cases the student's solution would be rejected as incorrect. As the knowledge base of a good quality is a normal requirement for any knowledge-based system, this is not a limitation of the approach used.

**Student modeller**

KERMIT maintains two kinds of student models: short-term and long-term ones. Short-term models are generated by matching student solutions to the knowledge base and the ideal solutions. The student modeller iterates through each constraint in the constraint base, evaluating each of them individually. For each constraint, the modeller initially checks whether the current problem state satisfies its relevance condition. If that is the case, the satisfaction component of the constraint is also verified against the current problem state. Violating the satisfaction condition of a relevant constraint signals an error in the student's solution.

The short-term student model consists of the relevance and violation details of each constraint, discovered during the evaluation of the problem state. The short-term model is only dependent on the current problem state and does not account for the history of the constraints such as whether a particular constraint was satisfied during the student's last attempt. The pedagogical module uses the short-term student model to generate feedback to the student.

The long-term student model of KERMIT is implemented as an overlay model. In contrast to the short-term model, the long-term model keeps a record of each constraint's history. It records information on how often the constraint was relevant for the student's solution and how often it was satisfied or violated. The pedagogical module uses this data to select new problems for the student. The long-term student model is saved in a file when the student logs out.

**Pedagogical module**

The pedagogical module (PM) is the driving engine of the whole system. Its main tasks are to generate appropriate feedback messages for the student and to select new practice problems. KERMIT individualises both these actions to each student based on their student model. Unlike ITSs that use model tracing, KERMIT does not follow each student's solution step-by-step. It only evaluates the student's solution once it is submitted. During evaluation, the student modeller identifies the constraints that the student has violated.

*Feedback generation*

The feedback from the system is grouped into six levels according to the amount of detail: *correct*, *error flag*, *hint*, *detailed hint*, *all errors* and *solution*. The first level of feedback, *correct*, simply indicates whether the submitted solution is correct or incorrect. The *error flag* indicates the type of construct (e.g. entity, relationship, etc.) that contains the error. *Hint* and *detailed hint* offer a feedback message generated from the first violated constraint. *Hint* is a general message such as "There are attributes that do not belong to any entity or relationship". On the other hand, *detailed hint* provides a more specific message such as "*The 'Address' attribute does not belong to any entity or relationship*", where the details of the erroneous object are given. Not all detailed hint messages give the details of the construct in question, since giving details on missing constructs would give away solutions. A list of feedback messages on all violated constraints is displayed at the *all errors* level. The ER schema of the complete solution is displayed at the final level (*solution* level).

Initially, when the student begins to work on a problem, the feedback level is set to the *correct* 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 *detailed hint* level. In other words, if the student submits the solutions four times the feedback level would reach the *detailed 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 compile the correct ER model. Automatically incrementing the levels of feedback is terminated at the *detailed hint* level to encourage to the student to concentrate on one error at a time rather than all the errors in the solution. Moreover, if the system automatically displays the solution to the student on the sixth attempt, it would discourage them from attempting to solve the problem at all, and may even lead to frustration. 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 errors', 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.

*Problem selection*

KERMIT examines the long-term student model to select a new practice problem for the student. In selecting a new problem from the available problems, the system first decides what concept is appropriate for the student on the basis of the student model. The concept that contains the greatest number of violated constraints is targeted. We have chosen this simple problem selection strategy in order to ensure that students get the most practice on the concepts with which they experience difficulties. In situations where there is no obvious "best" concept (i.e. a prominent group of constraints to be targeted), the next problem in the list of available problems, ordered according to increasing complexity, is given.

**Authoring tool for adding new problems**

The ideal solutions are represented internally in an encoded textual representation, as discussed previously. Similarly the problem text is also stored internally with embedded tags. Therefore, adding new problems and their solutions to the teaching system requires extensive knowledge of their internal representations. In order to ease the task of adding new problems, an authoring tool that administers the process of inserting new problems and automatically converting the problems and solutions to their internal representations was developed.

The authoring tool offers a number of benefits. It eliminates the burden of having to learn the complicated grammar used to represent the ideal solutions internally. As a consequence, teachers and other database professionals can add new problems and their ideal solutions to KERMIT easily. This feature makes it possible for the system to evolve without the need for programming resources. Furthermore, it makes it possible for teachers to customise the system by modifying the problem database to consist of problems that they select. As a result, database teachers would have better control over the subject material presented to the students.

The process of adding new problems using the authoring tool consists of three phases. The author needs to enter the problem text, then draw the ER diagram and finally specify the correspondences between the words of the problem text and the objects in the ideal solution. Initially, the user is given a text box in which to insert the problem text. At the completion of this phase, the user is presented with an interface, similar to the problem-solving interface presented to students, in which they can construct the ideal solution to the problem. Once the user completes the ideal solution to the problem using the ER modelling workspace, the authoring tool generates an image (in GIF format) of the ideal solution and saves it in the solutions directory to be used for the *complete solution* feedback level. The final phase involves the human teacher specifying the positions of the tags that need to be embedded in the problem text. The authoring tool automatically generates a unique ID for each construct in the solution and iteratively goes through each construct prompting the user to select the words in the problem text that correspond to the construct. It is up to the human teacher to make sure that he or she has specified all the relevant words of the problem text that correspond to a particular construct. Lastly, the tool adds the new problem with its embedded tags and its ideal solution converted to its internal representation.

## 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 KERMIT, described in this section.

### Pilot Study

The pilot study was conducted to evaluate the effectiveness of KERMIT and its contribution to learning. This study involved two versions of the system. One group of students used the full version, which generates the student model and offers various levels of feedback. The other group worked with ER-Tutor, a cut-down version of the system. We wanted to have a version of the system that would be similar to a classroom situation, where students do not get individual feedback. However, we also wanted all students to work on computers. Therefore, ER-Tutor contained the same problems and the drawing tool as KERMIT, but neither evaluated students' solutions nor offered individualized feedback. The only level of feedback available to students in ER-Tutor was the complete solution. The interfaces of both systems were similar, but with the option of selecting feedback and the feedback textbox missing from the ER-Tutor. Therefore, in the pilot study we were interested to see the effects of the individualization on students' learning. The pilot study also assessed the students' perception of the two systems.

There were other differences between the two systems besides the student model and feedback options. When a new construct is created in KERMIT, the system requires the student to name it by highlighting a phrase from the problem text. In ER-Tutor the student is not required to do that, and can type any name. Furthermore, there is a tutorial in KERMIT, which is shown at the beginning of the session, explaining the various features of the system, including highlighting of the problem text. Students using ER-Tutor were not shown this tutorial, and therefore had more time for interaction. The students who were using KERMIT were required to complete the

current problem before moving on to the next one, whereas ER-Tutor allowed students to skip problems as they pleased.

The pilot study took place at Victoria University, Wellington (VUW). Twenty eight volunteers from students enrolled in the Database Systems course (COMP302) offered by the School of Mathematical and Computer Sciences at VUW participated. The course is offered as a third year computer science paper, which teaches ER modelling as defined by Elmasri and Navathe (2003). The students who participated in the pilot study had previously learnt database modelling in the lectures and labs of the course.

*Procedure*

The study involved two streams of one-hour sessions. The participants were randomly allocated to the groups. Although the study was originally planned for a two-hour session, each group was only given only one hour due to resource shortages at VUW. Initially each student was given a document that contained a brief description of the study and a consent form. The students sat a pre-test and then interacted with the system. Finally, the participants were given a post-test and a questionnaire. Students were asked to stop interacting with the system after approximately 45 minutes into the study, as the study had to be concluded within an hour.

A pre- and post-test (given in Appendix A) were used to evaluate the students' knowledge before and after interacting with the system. To minimise any prior learning effects, we designed two tests (A and B) of approximately the same complexity. They contained two questions: a multiple choice question to choose the ER schema that correctly depicted the given scenario and a question asking the students to design a database that satisfied the given set of requirements. The tests were short because of the short duration of the study; if more questions were given in the tests, the students would have no time to interact with the system. In order to reduce any bias on either test A or B, the first half of each group was given test A as the pre-test and the remainder were given B as the pre-test. The students who had test A as their pre-test were given test B as their post-test and vice versa.

All the participants interacted with either KERMIT or ER-Tutor, composing ER diagrams that satisfied the given set of requirements. They worked individually, solving problems at their own pace. The set of problems and the order in which they were presented was identical for both groups. A total of six problems were ordered in increasing complexity.

The system assessment questionnaire (given in Appendix C) recorded the student's perception of the system. The questionnaire contained fourteen questions. Initially students were questioned on previous experience in ER modelling and in using CASE tools. Most questions asked the participants to rank their perception on various issues on a Likert scale with five responses ranging from *very good* (5) to *very poor* (1), and included the amount they learnt about ER modelling by interacting with the system and the enjoyment experienced. The students were also allowed to give free-form responses. Finally, suggestions were requested on enhancement of the system.

*Learning*

All the important events such as logging in, submitting a solution and requesting help that occurred during an interaction session with KERMIT are recorded in a log specific to each

student. An entry in the student log contains the date and the time associated with it. The data extracted from the student logs are summarised in Table 3.

Table 3
System interaction details for the pilot study

| | KERMIT | | ER-Tutor | |
|---|---|---|---|---|
| | **mean** | **s. d.** | **mean** | **s. d.** |
| Time spent on problem solving (min.) | 23:24 | 7:27 | 31:56 | 8:48 |
| No. of attempted problems | 1.64 | 0.50 | 3.50 | 0.85 |
| No. of completed problems | 1.21 | 0.70 | 0.86 | 0.95 |
| Time spent per problem (min.) | 14:25 | 4:54 | 9:06 | 5:04 |
| No. of attempts per problem | 6.53 | 3.47 | N/A | N/A |

The ER-Tutor group spent eight minutes more interacting with the system. One reason for the difference is that the students who used KERMIT were given the online tutorial. The students who used KERMIT spent more time per problem, as they needed to complete a problem before moving on to the next one. This is reflected in the mean number of attempted problems and the mean number of completed problems for both groups. Students in the ER-Tutor group attempted almost twice as many problems as the KERMIT group. However, the KERMIT group had a high completion rate of 74% whereas the ER-Tutor group had a completion rate of 25%. The students in the KERMIT group on average submitted 6.53 solutions for each problem (ranges from 1 to 31).

The domain knowledge of KERMIT is represented as constraints. If the constraints represent an appropriate unit of knowledge of the domain, then learning should follow a smooth curve with a decreasing trend in terms of constraint violations (Anderson, 1993). We evaluated this prospect by analysing the student logs and identifying each problem-state in which a constraint was relevant. Each constraint relevance occasion was rank ordered from 1 to R. Mitrovic and Ohlsson (1999) refer to these as occasions of application. For each occasion, we recorded whether a relevant constraint was satisfied or violated. The analysis was repeated for each participant of the study.

From the analysis we calculated, for each participant, the probability of violating each individual constraint on the first occasion of application, the second occasion and so on. The individual probabilities were averaged across all the constraints in order to obtain an estimation of the probability of violating a given constraint C on a given occasion. The probabilities were then averaged across all participants and plotted as a function of the number of occasions when C was relevant, as shown in Figure 12. As the number of uses increases, the set of constraints that were relevant for that number of times diminishes in size. At n=12, at least two thirds of the participants have used a constraint, while at the end of the series, there might be just one participant. Hence, a single failure at the end of the series will have a much bigger impact on probability than at the start of the curve. We have arbitrarily chosen n=12 to reduce this effect.

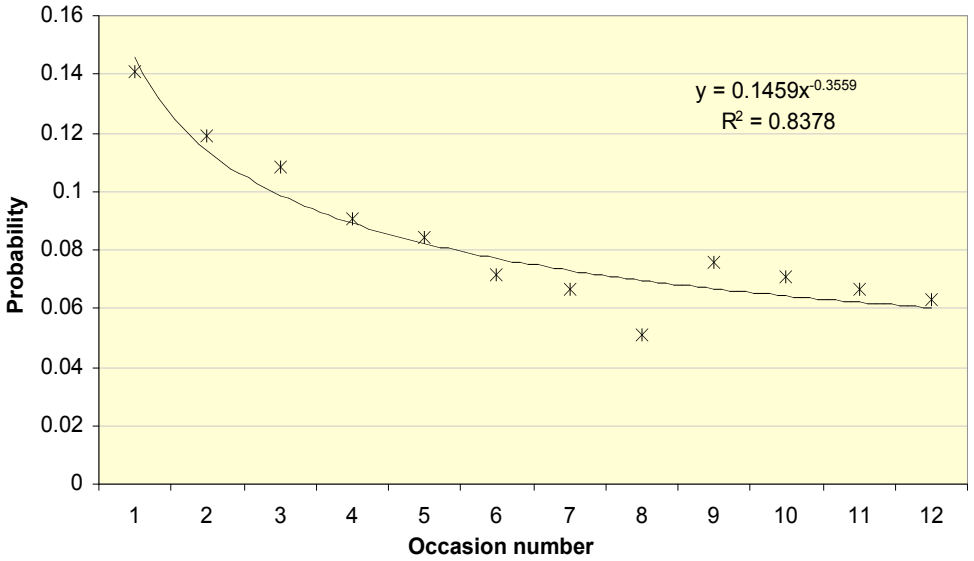



Fig. 12. Probability of violating a constraint as a function of the occasion when that constraint was relevant, averaged over all participants in the pilot study

The graph shows a regular decrease in probability that can be approximated by a power curve. The curve fits very closely to the data points with an $R^2$ power-law fit of 0.84. The initial probability of violating a constraint is approximately 14%. This probability is low because the students had learnt ER modelling in lectures and practised constructing ER models in tutorials prior to the study. After twelve occasions, the probability of violating a constraint dropped down to 6% (45% of the original). The graph shows that students learn (i.e. the probability of violating a given constraint decreases) with practice (i.e. as the occasion that they use the constraint increases). The data supports the belief that KERMIT's constraint base partitions the ER domain knowledge into psychologically relevant units that are learned independently and that the degree of mastery of a given unit is a function of the amount of practice on that unit.

*Subjective analysis*

All the participants completed a questionnaire at the end of the study. Table 4 gives the mean responses of the participants regarding their attitude towards the system. The KERMIT group required more time to learn the interface, as we expected, since the KERMIT's interface is more sophisticated then that of ER-Tutor. Students using KERMIT were forced to highlight words of the problem text to indicate the semantic meaning of each construct, whereas students using ER-Tutor had no such requirement.

Table 4
Mean responses from the user questionnaire

| | KERMIT | | ER-Tutor | |
|---|---|---|---|---|
| | **mean** | **s. d.** | **Mean** | **s. d.** |
| Time to learn interface (min.) | 13.21 | 9.32 | 10.00 | 15.14 |
| Amount learnt | 2.43 | 0.85 | 2.64 | 1.08 |
| Enjoyment | 3.64 | 1.08 | 3.43 | 0.94 |
| Ease of using interface | 3.50 | 0.65 | 3.71 | 0.99 |
| Usefulness of feedback | 3.00 | 0.96 | 2.79 | 1.25 |

When asked to rate the amount they learnt from the system, the mean response for the KERMIT group was 0.21 higher than the other group. The difference was found to be statistically insignificant. There are a few factors that may have influenced such low mean rankings for the amount learnt. The students who used KERMIT commented that the interaction time was too short. Some students also provided free-form comments, suggesting that the final solutions were not helpful as there was no indication of what was wrong with their own solutions.

Students found KERMIT's interface more complicated to use, as expected. 86% of participants from both groups would recommend the system that they used to other students. 71% of the KERMIT group wanted more feedback, compared to 86% of the ER-Tutor group. The students who used KERMIT were very enthusiastic about using the system. They wrote positive comments about the system, emphasizing the usefulness of feedback, the high quality of the interface and their enjoyment.

*Pre- and post-test performance*

The mean scores on the pre- and post-tests are given in Table 5. The KERMIT group scored a mean of 5.86 out of a possible 12 for the pre-test, while the mean of the ER-Tutor group was 6. Since the difference of mean scores is statistically insignificant we can conclude that the two groups are comparable.

Table 5
Mean pre- and post-test scores of the two groups

| | **Pre-test** | **s. d.** | **Post-test** | **s. d.** |
|---|---|---|---|---|
| KERMIT | 5.86 | 1.46 | 6.50 | 2.47 |
| ER-Tutor | 6.00 | 2.18 | 6.29 | 2.09 |

The KERMIT group scored 0.64 higher in the post-test, whereas the other group gained 0.29. The difference is insignificant. A statistically significant improvement cannot be expected from such a short interactive session with the system.

We computed the effect size and power, which are the two measures commonly used to determine the effects and validity of an experiment. Effect size is a standard method of

comparing the results of one pedagogical experiment to another. The common method to calculate the effect size in the ITS community is to subtract the control group's mean gains score from the experimental group's mean gain score and divide by the standard deviation of the gain scores of the control group (Bloom, 1984). This calculation yields (0.64 – 0.29) / 2.46 = 0.15. The resulting effect size is very small in comparison to an effect size of 0.63 published in (Albacete & VanLehn, 2000) and 0.66 published in (Mitrovic et al., 2002). Both papers report on the experiments where the sessions lengths were two hours. Better results on the effect size have been obtained in studies where interactions lasted for a whole semester or an academic year. Bloom (1984) reports an effect size of 2.0 for one-on-one human tutoring in replacement of classroom teaching and Anderson and co-workers (Anderson et al., 1996) reports an effect size of 1.0 for a study that lasted for one semester. Considering these results, yielding an effect size of 0.15 with a study that lasted for only half an hour is quite promising.

Chin (2001) published another method of calculating the effect size as the omega squared value ($\omega^2$). It gives the magnitude of the change in dependent variable values due to changes in the independent variables as a percentage of the total variability. $\omega^2$ is calculated using $\omega^2 = \sigma_A^2 / (\sigma_A^2 + \sigma_{S/A}^2)$, where $\sigma_A^2$ is the variance of the effects of varying the independent variable and $\sigma_{S/A}^2$ is the random variance among participants. According to this formula we get an effect size of 0.03, which is considered small in social sciences (Chin, 2001). An effect size of 0.15 is considered large. The omega-squared value of the experiment further points out that the amount of time allocated for the participants to interact with the system was insufficient.

Power or sensitivity gives a measure of how easily the experiment can detect differences. Power is measured as the fraction of experiments that for the same design, the same number of participants and the same effect size would produce a given significance. In other words, the power of 0.5 means that half of the repeated experiments would produce non-significant results. Chin (2001) recommends that researchers should strive for a power of 0.8. We calculated the power of this experiment to find out how easy it is to detect differences in the pre- and post-test. The calculation yielded a power of 0.13 at a significance of 0.05, which is quite low. The low power value can be attributed to the low number of students, each group having only fourteen participants.

*Discussion*

Although the pilot study was short, we got some promising results. Students who used KERMIT displayed a slightly higher gain score in comparison to the ER-Tutor group. More importantly, the pre- and post-test scores demonstrated that using KERMIT did not hamper students' abilities in ER modelling.

The pilot study also enabled us to identify several features of the system that needed improvement. During the pilot study, we have identified several bugs in the system, and also some problems with the constraint base. The implementation of KERMIT was fine-tuned and eight new constraints were defined. Also, the students found one of the tests used as pre/post tests to be a little harder than the other.

Some students struggled with highlighting words in the problem text to specify each construct's semantic meaning. A typical mistake was to add a new construct to the diagram, highlight a word from the problem text and rename the construct to have a different semantics. In such cases KERMIT cannot give useful hints, as it is designed to ignore the construct's name

assigned by the user and only considers its tag associated with the highlighted area of the problem text. This problem can be avoided by preventing the users from renaming constructs, and using the highlighted text as the name.

Students who used KERMIT were forced to complete each problem before moving on to the next problem, whereas students in the ER-Tutor group were free to abandon problems at any time. Students' perception has been further affected by this variation. This is also a flaw of the experiment, since the group who used ER-Tutor were treated differently, disqualifying them as a true control group. Therefore we decided to allow students to abandon problems in both systems.

Some students had difficulties in understanding the feedback messages presented by KERMIT. The feedback messages of the version of KERMIT used in the pilot study were short and less descriptive. Some messages delivered small hints that novice users struggled to understand. We therefore revised the feedback messages of the constraint base making them more descriptive.

## Evaluation Study

An evaluation study was carried out at the University of Canterbury, Christchurch in August 2001. Similar to the pilot study, this study involved a comparison of two groups of students learning ER modelling by using KERMIT and ER-Tutor. KERMIT was enhanced in the light of the findings from the pilot study. The study involved sixty-two volunteers from students enrolled in the Introduction to Databases course (COSC 226) offered by the Computer Science department. The course, offered as a second year paper, teaches ER modelling as outlined by Elmasri and Navathe (2003). The students had learnt ER modelling concepts during two weeks of lectures and had some practice during two weeks of tutorials prior to the study.

### *Procedure*

The evaluation study was conducted in two streams of two-hour laboratory sessions. Each session proceeded in four distinct phases identical to the pilot study. In the pilot study, we discovered that the multiple-choice questions in the tests were ineffective, since over 75% of the students had made the correct choice. These multiple-choice questions were replaced by a question where the students were asked to specify the cardinality and participation constraints of a relationship. In order to be certain of the two tests having equal complexity, both the questions in test A and B dealt with a binary relationship from the university database. Since we were not satisfied by the evaluation of the student's abilities from the tests used in the pilot study, we added an extra question that involved a partially completed ER model. The students were asked to complete the ER model that included an identifying relationship with a regular entity and a weak entity. The task also included specifying the cardinality and participation constraints as well as specifying the partial key of the weak entity. Both tests (given in Appendix B) contained similar scenarios that produced similar ER models. The tests were marked by one person, who was not aware of what groups the participants belonged to.

Table 6
Mean system interaction details

| | KERMIT | | ER-Tutor | |
|---|---|---|---|---|
| | **Mean** | **s. d.** | **mean** | **s. d.** |
| Time spent on problem solving (min.) | 66:39 | 21:22 | 57:58 | 34:38 |
| Time spent per completed problem (min.) | 23.36 | 6:55 | 23.46 | 21.40 |
| No. of attempted problems | 4.36 | 1.45 | 4.10 | 2.55 |
| No. of completed problems | 1.75 | 1.14 | 1.97 | 1.20 |
| No. of attempts per problem | 6.05 | 2.83 | N/A | N/A |

*Learning*

The results, summarised in Table 6, show that the students in the experimental group spent more time interacting with the system and solving problems, than the control group. The average times spent on completing a problem for both groups were very similar. These findings suggest that even though the students using KERMIT were forced to indicate the semantic meaning of each construct by highlighting a word in the problem text, their performance was not degraded. The average number of attempts per problem is 6.05, with the maximal number of attempts was 25.

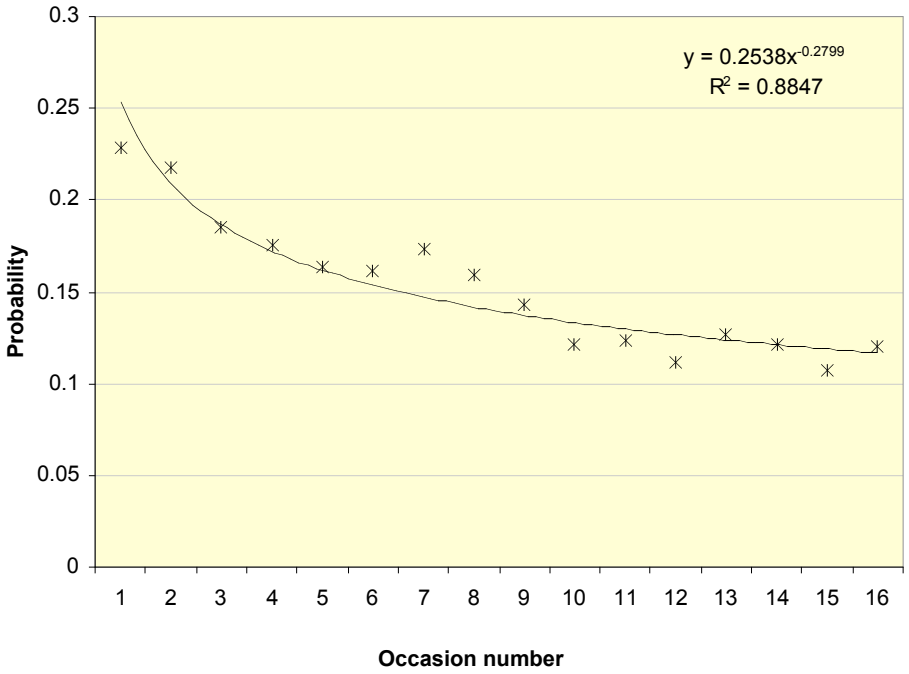



Fig. 13. Probability of violating a constraint as a function of the occasion when that constraint was relevant, averaged over all participants

The mastery of constraints was analysed by examining the student logs. Figure 13 illustrates the probability of violating a constraint plotted against the occasion number for which it was relevant, averaged over all participants. The data points show a regular decrease, which is approximated by a power curve. The power curve displays a close fit with an $R^2$ power-law fit of 0.88. The probability of 0.23 for violating a constraint at its first occasion of application has decreased to 0.12 at its sixteenth occasion of application displaying a 53% decrease in the probability. These findings are analogues to the results from the pilot study, where a decrease of 45% was displayed. The results of the mastery of constraints for this experiment further strengthen the claim that the students learn ER modelling by interacting with KERMIT.

*Subjective analysis*

All the participants were given a questionnaire at the end of their session to determine their perceptions of the system. Table 7 displays a summary of the responses. The students in both groups required approximately the same time to learn the interface. Since KERMIT's interface is more complicated, forcing the users to highlight words in the problem text to indicate the semantic meaning of constructs, we expected that students who used KERMIT would require longer to learn its interface. It is encouraging to see that the students did not find the relatively more complex interface harder to learn. This is further illustrated by the similar mean numbers of problems attempted and completed by each student, as shown in Table 7.

The mean response when asked to rate how much they learnt by interacting with KERMIT was 3.19. The control group using ER-Tutor had a mean rating of 3.06. This difference was found to be insignificant. The free-form comments from the experimental group emphasized the importance of feedback for their learning. Both groups of students, on average, rated their enjoyment of the system on a similar scale.

The students who used ER-Tutor rated its interface easier to use in comparison to the students who used KERMIT. The difference of 0.46 in favour of ER-Tutor's interface is statistically significant ($t = 1.78$, $p < 0.01$). This result was expected since KERMIT's interface is more complex than ER-Tutor's.

Table 7
Mean responses from the user questionnaire for the evaluation study

| | **KERMIT** | | **ER-Tutor** | |
|---|---|---|---|---|
| | **mean** | **s. d.** | **mean** | **s. d.** |
| Time to learn interface (min.) | 11.50 | 11.68 | 11.94 | 14.81 |
| Amount learnt | 3.19 | 0.65 | 3.06 | 0.89 |
| Enjoyment | 3.45 | 0.93 | 3.42 | 1.06 |
| Ease of using interface | 3.19 | 0.91 | 3.65 | 1.08 |
| Usefulness of feedback | 3.42 | 1.09 | 2.45 | 1.12 |

The mean rating for the usefulness of feedback presented by each system is considerably higher for the experimental group who interacted with KERMIT. The difference is statistically significant ($t = 3.45$, $p < 0.01$). These results are analogous with our expectations due to the

difference in the information content presented as feedback from each system. Students who used KERMIT were offered individualised feedback on their solutions upon submission. On the other hand the students who used ER-Tutor only had the option of viewing the completed solution to each problem. 74% of the students who used ER-Tutor indicated the need for more detailed help other than the complete solution, compared to 61% of the students who used KERMIT.

Students who used KERMIT had a better perception of the system as a whole in comparison to the group who used ER-Tutor. This was shown in their responses to whether they would recommend the system to others, where approximately 84% of the students who used KERMIT indicated that they would recommend the system to others, compared to 68% of the control group students.

*Pre- and post-test performance*

The mean scores of the pre- and post-test (out of a possible 22) are shown in Table 8. The difference in scores on the pre-test is statistically insignificant, confirming that the two groups initially had equal knowledge in ER modelling and that they are comparable.

Table 8
Mean pre- and post-test scores for the evaluation study

| | **Pre-test** | **s. d.** | **Post-test** | **s. d.** |
|---|---|---|---|---|
| KERMIT | 16.16 | 1.82 | 17.77 | 1.45 |
| ER-Tutor | 16.58 | 2.86 | 16.48 | 3.08 |

The experimental group achieved a significantly higher score on the post-test ($t = 4.91$, $p < 0.01$). Conversely, the difference in pre- and post-test scores of the group who used ER-Tutor is statistically insignificant. The difference in post-test scores of the two groups is statistically significant ($t = 2.07$, $p < 0.05$). We can conclude from these results that students who used KERMIT learnt more about ER modelling using KERMIT than the control group students.

The effect size and power for the experiment were calculated. The common method used in the ITS community to calculate the effect size yields 0.63, which is comparable with the effect size of 0.63 published by Albacete and Vanlehn (2000) and 0.66 published in (Mitrovic et al., 2002). Both published results are also results from experiments that spanned a two-hour session. An effect size of 0.63 with the students interacting with the system for approximately an hour is an excellent result.

The effect size calculated using the $\omega^2$ value was 0.12. As Chin (2001) points out that an effect size of 0.15 is considered large in social sciences, the effect size of 0.12 calculated for this experiment can be considered as a relatively large effect size. This value suggests a considerable change in the gain score as the group (either control or experimental) that a student belongs to changes. In other words, the gain score is highly dependent on whether a student interacted with KERMIT or ER-Tutor.

We also calculated the power of the experiment, which is a measure of how easily the experiment can detect differences. Chin (2001) recommends that researchers should strive for a

power of 0.8. The power of this experiment was calculated as 0.75 at significance 0.05, which is an excellent result.

*Discussion*

The results show that students' knowledge increased by using KERMIT, Students who interacted with KERMIT achieved significantly higher scores on the post-test, suggesting that they acquired more knowledge in ER modelling. Subjective evaluation shows that the students in the experimental group felt they learnt more than their peers in the control group. It is surprising to record a high mean ranking of approximately 3 for the control group, when asked how much they learnt from ER-Tutor. This may be due to the typical student misconception of assuming that they learnt a lot by analysing the complete solution. The student responses to the questionnaire suggested that most students appreciated the feature of being able to view the complete ER model. The student's perception may have further been influenced by a sense of complacency from being able to view the complete solution. As an observer during the experiments, the author noticed that some students attempted to replicate the system's solution, which is not likely to result in deep learning.

There were other encouraging signs that suggested that KERMIT was an effective teaching tool. A number of students who participated in the study using KERMIT inquired about the possibility of using KERMIT in their personal time for practicing ER modelling. Moreover, there were a few students who requested special permission to continue interacting with KERMIT even after the allocated two hours for the session had elapsed.

## CONCLUSIONS

This paper has discussed the design and implementation of KERMIT, developed to assist students learning ER modelling. KERMIT's effectiveness in teaching ER modelling was evaluated in the two classroom experiments. The results of the final classroom evaluation proved that KERMIT is an effective educational tool. The participants who used the full version of KERMIT showed significantly better results in both the subjective and objective analysis in comparison to the students who practiced ER modelling with a conventional drawing tool.

The student modelling technique used in KERMIT, Constraint Based Modelling has previously been used to represent domain and student knowledge in SQL-Tutor (Mitrovic, 1998; Mitrovic & Ohlsson, 1999) and in CAPIT (Mayo et al., 2000; Mayo & Mitrovic, 2001). SQL-Tutor teaches a declarative language, and the evaluation performed on this system showed that CBM is well-suited towards representing knowledge necessary for specifying queries. CAPIT is a system that teaches punctuation, which is a very restricted domain requiring students to master a small number of constraints. In both cases, the analysis of students' behaviour while interacting with these systems proved the sound psychological foundations of CBM and the appropriateness of constraints as the basic units of knowledge. The research presented in this paper demonstrated that CBM can also be used to effectively represent knowledge for open-ended tasks such as design. This is an important result of this research that further strengthens the credibility of CBM. In the following section we discuss the applicability of CBM to other design tasks. Finally, we discuss the avenues for future work.

## CBM and Design Tasks

Although there has been a lot of research on design tasks within specific disciplines, the theory of generic design (i.e. domain-independent characterization of design tasks) has proven to be extremely challenging. Goel and Pirolli (1992) define generic design as a radical category, which is described in terms of prototypical examples and some unpredictable variations of them. They define a dozen criteria to describe design task environments, which allow for identification of design tasks. We describe some of these criteria here. Design tasks are ill-structured problems, because their start/goal states and problem-solving algorithms are underspecified (Reitman, 1964). The start state is usually described in terms of ambiguous and incomplete specifications. The problem spaces are typically huge, and operators for changing states do not exist. The goal state is also not clearly stated, but is rather described in abstract terms. There is no definite test to use to decide whether the goal has been attained, and consequently, there is no best solution, but rather a family of solutions. Design tasks typically involve huge domain expertise, and large, highly structured solutions. Typical examples of design tasks include architecture, software design, mechanical engineering and music composition.

Although design tasks are underspecified, Goel and Pirolli (1992) identify a set of 12 invariant features of design problem spaces, such as problem structuring, distinct problem-solving phases, modularity, incremental development, control structure, use of artificial symbol systems and others. Problem structuring is the necessary first phase in design, as the given specifications of a problem are incomplete. Therefore, the designer needs to use additional information that comes from external sources, the designer's experience and existing knowledge, or needs to be deduced from the given specifications. Only when the problem space has been constructed via problem structuring, problem solving can commence. The second feature specifies three problem-solving phases: preliminary design, refinement and detail design. Design problem spaces are modular, and designers typically decompose the solution into a large number of sparsely connected modules and develop solutions incrementally. When developing a solution, designers use the limited commitment mode strategy, which allows one to put any module on hold while working on other modules, and return to them at a later time.

Database design shares these common features of generic design (as discussed in Section 2). In this paper, we have shown that CBM can be used effectively to support students learning conceptual database design. Goel and Pirolli (1988) argue that design problems by their very nature are not amenable to rule-based solutions. On the other hand, constraints are extremely suitable for representing design solutions: they are declarative, non-directional, and can describe partial or incomplete solutions. A constraint set specifies all conditions that have to be simultaneously satisfied without restricting how they are satisfied.

One of the features of design tasks is the personalized stopping rules and evaluation functions (Goel & Pirolli, 1992). As there are no right or wrong answers in design, the evaluation functions are necessarily personalized (i.e. the solution is developed when the designer is satisfied with it). In KERMIT there is an ideal solution for each problem. The ideal solution is chosen by the teacher to highlight some important domain principles. If the ideal solutions exist, constraint can compare the student's solution to the ideal one however, in some domains it might not be possible to come up with the ideal solution. For example, in house design, the student might be given the total area of the house, the location and shape of the section the house is to be built on, the number of bedrooms etc, but there may be many solutions that satisfy all the

problem specifications and the general domain constraints. In such domains, semantic constraints will not compare the student solution to the ideal solution; instead, they would check whether the student's solution satisfies the requirements of the problem.

CBM can support all features of design tasks. Each constraint tests a particular aspect of the solution, and therefore supports modularity. Incremental development is supported by being able to request feedback on a solution at any time. At the same time, CBM supports the control structure used by the designer (student) as it analyses the current solution looking at many of its aspects in parallel: if a particular part of the solution is incomplete, the student will get feedback about missing constructs. CBM can be used to support all problem-solving phases. Therefore, we believe that CBM can be applied to all design tasks.

**Future Work**

There are a number of future avenues that can be explored to further improve KERMIT. The system currently requires the user to indicate the semantic meaning of each construct by highlighting a word from the problem text. A more flexible approach is to allow students to use their own names and improve the system by incorporating a natural language processing module to identify correspondences between the student's solution constructs and the ideal solution constructs. Students using KERMIT with this enhanced system would find the interface significantly easier to use since their progress would not be hampered by having to highlight words in the problem text. Another current project involves opening the student model so that the student can inspect the content of the model (Hartley & Mitrovic, 2002). Our hypothesis is that the visualization of the student model will encourage the student to reflect on his/her knowledge, and that such model would also support the student in gaining a deeper understanding of the structure of the domain knowledge.

KERMIT's long-term student model is implemented as a simple overlay model. The long-term student model could be improved by using normative theories. A Bayesian network could be used to represent the student model and could also predict the student's behaviour with respect to the constraints. The probabilistic student model can be used to select feedback and new problems for the student.

The current system only presents general hint messages on the errors in the student's solution. The feedback of the system could be enhanced to provide support for deep learning. We have recently started a new project, which will enhance KERMIT to support self-explanation (Weerasinghe & Mitrovic, 2002).

The current version of the system is implemented as a stand-alone Windows program. The system could be enhanced to run as a web-based system to enable a number of students working on multi-platforms to use the system simultaneously. Enhancing the system to function over the Internet would also allow the possibility of distance learning, where students could learn ER modelling from the system from the comfort of their own home.

**ACKNOWLEDGEMENTS**

The work presented here was supported by the University of Canterbury MSc scholarship to the first author, and the research grant U6430 to the second author. The comments from the

anonymous reviewers are most appreciated. We thank Ken Koedinger for advising on the evaluation studies, Pavle Mogin for letting us perform the pilot study in his course, and Danita Hartley for helping with data analyses. This research could not have been done without the support of other past and present members of ICTG.

## REFERENCES

Albacete, P. L., & VanLehn, K. (2000). The Conceptual Helper: an Intelligent Tutoring System for Teaching Fundamental Physics Concepts. In G. Gauthier, C. Frasson & K. VanLehn (Eds.) *Intelligent Tutoring Systems* (pp. 564-573). Berlin: Springer.

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

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

Batra, D., & Antony, S. R. (1994). Novice Errors in Conceptual Database Design. *European Journal of Information Systems*, 3(1), 57-69.

Batra, D., Hoffer, J. A., & Bostrom, R. P. (1990). Comparing Representations with Relational and EER Models. *Communications of the ACM*, 33(2), 126-139.

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.

Chen, P. P. (1976). The Entity Relationship Model - Toward a Unified View of Data. *ACM Transactions Database Systems*, 1(1), 9-36.

Chin, D. N. (2001). Empirical Evaluation of User Models and User-adapted Systems. *User Modeling and User Adapted Interaction*, 11(1), 181-194.

Constantino-Gonzalez, M. d. l. A., & Suthers, D. D. (1999). A Coached Computer-Mediated Collaborative Learning Environment for Conceptual Database Design. In S. Lajoie & M. Vivet (Eds.) *Artificial Intelligence in Education* (pp. 645-647). Amsterdam: IOS Press.

Constantino-Gonzalez, M. d. l. A., & Suthers, D. D. (2000). A Coached Collaborative Learning Environment for Entity-Relationship Modeling. In G. Gauthier, C. Frasson & K. VanLehn (Eds.) *Intelligent Tutoring Systems* (pp. 324-333). Berlin: Springer.

Constantino-Gonzalez, M. d. l. A., Suthers, D. D., & Lcaza, I. J. (2001). Designing and Evaluating a Collaboration Coach: Knowledge and Reasoning. In J. Moore, C. L. Redfield & W. L. Johnson (Eds.) *Artificial Intelligence in Education* (pp. 176-187). Amsterdam: IOS Press.

Corbett, A. T., Trask, H. J., Scarpinatto, K. C., & Hadley, W. S. (1998). A Formative Evaluation of the PACT Algebra II Tutor: Support for Simple Hierarchical Reasoning. In B. P. Goettl, H. M. Halff, C. L. Redfield & V. J. Shute, (Eds.) *Intelligent Tutoring Systems* (pp. 374-383). Berlin: Springer.

Elmasri, R., & Navathe, S. B. (2003). *Fundamentals of Database Systems*. Boston: Addison Wesley.

Goel, V., & Pirolli, P. (1988). Motivating the Notion of Generic Design with Information Processing Theory: the Design Problem Space. *AI Magazine,* 10, 19-36.

Goel, V., & Pirolli, P. (1992). The Structure of Design Problem Spaces. *Cognitive Science*, 16, 395-429.

Hall, L., & Gordon, A. (1998). A Virtual Learning Environment for Entity Relationship Modelling. *SIGCSE Bulletin*, 30 (1), 345-353.

Hartley, D., & Mitrovic, A. (2002). Supporting Learning by Opening the Student Model. In S. Cerri, G. Gouarderes & F. Paraguacu (Eds.) *Intelligent Tutoring Systems* (pp. 453-462). Berlin: Springer.

Johnson , W. L., Rickel, J.W., & Lester, J.C. (2000). Animated Pedagogical Agents: Face-to-face Interaction in Interactive Learning Environments. *Artificial Intelligence in Education*, 11, 47-78.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring goes to School in the Big City. *Artificial Intelligence in Education*, 8(1), 30-43.

Lester, J., Towns, S., & Fitzgerald, P. (1999). Achieving Affective Impact: Visual Emotive Communication in Lifelike Pedagogical Agents. *Artificial Intelligence in Education,* 10, 278-291.

Mayo, M., & Mitrovic, A. (2001). Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *Artificial Intelligence in Education*, 12(2), 124-153.

Mayo, M., Mitrovic, A., & McKenzie, J. (2000). CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation. In Kinshuk, C. Jesshope & T. Okamoto (Eds.) *Advanced Learning Technology: Design and Development Issues* (pp. 151-154). Los Alamitos, CA: IEEE Computer Society.

Microsoft, Microsoft Agent, http://msdn.microsoft.com/msagent/default.asp?

Mitrovic, A. (1998). Experiences in Implementing Constraint-Based Modelling in SQL-Tutor. In B. P. Goettl, H. M. Halff, C. L. Redfield & V. J. Shute (Eds.) *Intelligent Tutoring Systems* (pp. 414-423). Berlin: Springer.

Mitrovic, A., Koedinger, K., & Martin, B. (2003). A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling. In P. Brusilovsky, A. Corbett & F. de Rosis (Eds.) *User Modeling 2003* (pp. 313-322). Berlin: Springer.

Mitrovic, A., & Suraweera, P. (2000). Evaluating an Animated Pedagogical Agent. In G. Gauthier, C. Frasson & K. VanLehn (Eds.) *Intelligent Tutoring Systems* (pp. 73-82). Berlin: Springer.

Mitrovic, A., Martin, B., & Mayo, M. (2002). Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor. *User-Modelling and User Adapted Interaction*, 12, 243-279.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001b). Constraint-based Tutors: a Success Story. In L. Monostori, J. Vancza & M. Ali (Eds.) *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (pp. 931-940). Berlin: Springer.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a Constraint-based Tutor for a Database Language. *Artificial Intelligence in Education, 10* (3-4), 238-256.

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

Payne, S., & Squibb, H. (1990). Algebra Mal-rules and Cognitive Accounts of Errors. *Cognitive Science*, *14*, 445-481.

Reitman, W.R. (1964). Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-defined Problems. In M. W. Shelly & G. L. Bryan (Eds.) *Human Judgements and Optimality.* New York: Wiley.

Suraweera, P., & Mitrovic, A. (2001). Designing an Intelligent Tutoring System for Database Modelling. In M. J. Smith & G. Salvendy (Eds.) *Human-Computer Interaction* (pp. 745-749). Mahwah, NJ: Lawrence Erlbaum.

Visio, http://www.microsoft.com/office/visio/

Weerasinghe, A., & Mitrovic, A. (2002). Enhancing Learning through Self-Explanation. In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson & C-H Lee (Eds.) *Computers in Education* (pp. 244-248). Los Alamitos, CA: IEEE Computer Society.

## Appendix A: Pre- and Post-Test from the Pilot Study

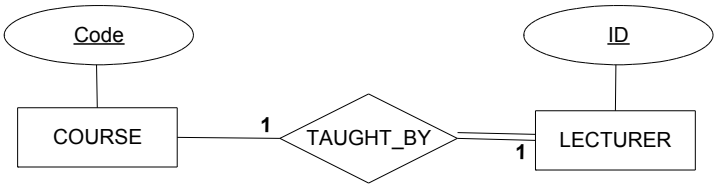
### Test A

Please specify your user ID you use to log on to the system:

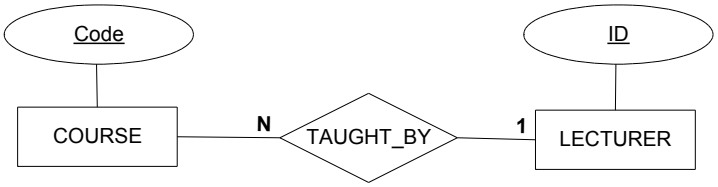
*Answer both questions.*

***1.*** Select the most appropriate ER schema that best describes the given following situation: *Lecturers teach courses.*

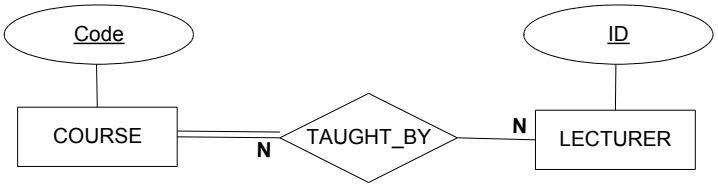
(a)



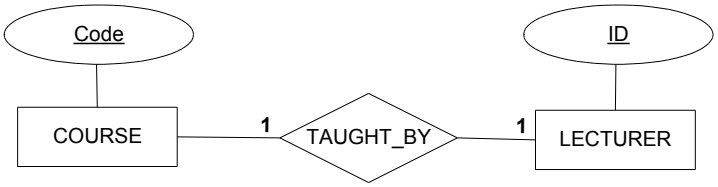

(b)




(c)




(d)




2. Draw an ER diagram that captures the following situation. For each course a student has taken, we need to know the final grade. Each course has a unique course code.

**Test B**

Please specify your user ID you use to log on to the system:

*Answer both questions.*

*1.* Select the most appropriate ER schema that best describes the given following situation: *Courses recommend textbooks.*

(a)


(b)


(c)


(d)


2. Each course, with a unique course code, consists of several sections. For each section, we know the topic it covers, and the number of lectures and labs it contains. The same topic can be covered in several courses, but the number of lectures and labs will differ in that situation.

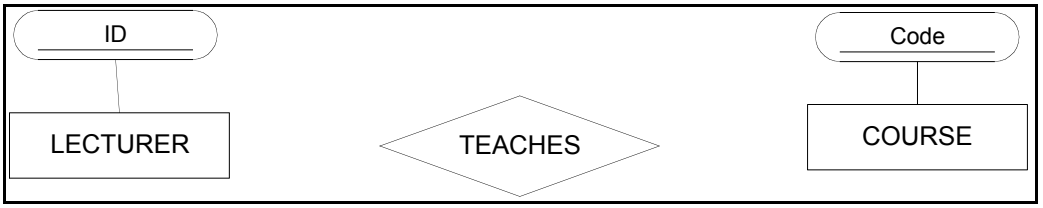
## Appendix B: Pre- and Post-Test from the Evaluation Study

### Test A

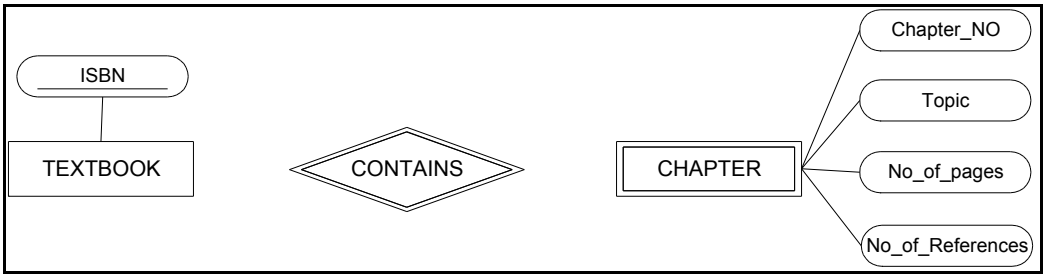
**Please specify your computer name:** **PC**

*Answer all questions.*

1. The following ER model represents the relationship between lecturers and courses. Please specify the cardinality and participation constraints.




2. The following partially completed ER model, represents the scenario described below. Please complete the ER model. A textbook, with a unique ISBN, contains a number of chapters. For each chapter we know its chapter number, topic, total number of pages and total number of references. Different textbooks may cover the same topics.



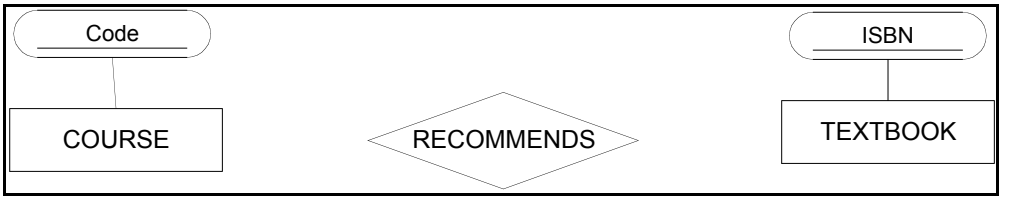

3. Please draw an ER diagram that captures the following situation. For each course a student has taken, we need to know the final grade. Each course has a unique course code and a student has his/her student ID.

## Test B

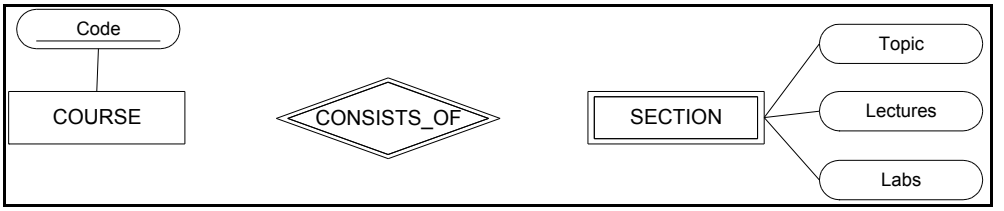
**Please specify your computer name:** PC

*Answer all questions.*

1. The ER diagram below represents the relationship between courses and recommended textbooks. Please specify the participation and cardinality constraints.




2. The ER diagram below is a partially completed ER model that represents the information given below. Please complete the ER diagram. Each course, with a unique course code, consists of several sections. For each section, we know the topic it covers, and the number of lectures and labs it contains. The same topic can be covered in several courses, but the number of lectures and labs will differ in that situation.




3. Please draw an ER diagram that models the requirements given below. Sometimes students work in groups. Each group has a unique number and students have their student IDs. A student may have different roles in various groups he/she belongs to.

## Appendix C: User Questionnaire

### KERMIT : Knowledge-based Entity Relationship Modelling Intelligent Tutor

Thank you for using KERMIT. 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.

Please specify your computer name: _____

1. **What is your previous experience with ER 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 ER modelling from using the system? (please circle one)**

**Nothing** ——————————— **Very much**

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

**Please comment**

4. **Did you enjoy leaning with KERMIT? (please circle one)**

**Not at all** ——————————— **Very much**

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

**Please comment**

5. **Would you recommend KERMIT to other students? (please circle one)**

| | |
|---|---|
| **(a)** | **Yes** |
| **(b)** | **Don't know** |
| **(c)** | **No** |

6. **Did you find the interface easy to use? (please circle one)**


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

**Please comment**

7. **Did you find the feedback from KERMIT useful? (please circle one)**


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

**Please comment**

8. **Would you prefer more details in feedback? (please circle one)**

| | |
|---|---|
| **(a)** | **Yes** |
| **(b)** | **Don't know** |
| **(c)** | **No** |

**Please comment**

9. **Did you encounter any software problems or system crashes? (please circle one)**

| | |
|---|---|
| **(a)** | **Yes** |
| **(b)** | **No** |

**If yes, please specify which**

10. **What did you like in particular about KERMIT?**
11. **Is there anything you found frustrating about the system?**
12. **Do you have any suggestions for improving KERMIT?**