View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UC Research Repository

# An Intelligent Teaching System for Database Modelling

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science at the University of Canterbury

**Pramuditha Suraweera**

This thesis is dedicated to my dear parents

# Acknowledgements

I thank my supervisor, Antonja Mitrovic, for her invaluable advice and guidance. Thank you Tim Bell, my associate supervisor, for your helpful advice, and Ken Koedinger who provided valuable guidance. I also thank Jane McKenzie for proof reading drafts of this thesis.

My sincerest thanks to Natashka Waduge for all her support.

The University of Canterbury Masters Scholarship helped fund this research, and made my life easier.

John Edwards, the Information Services Manager of the University of Canterbury, accommodated me as a part-time programmer during the duration of this research. Thank you for all your help.

I thank Danita Hartley for her assistance in analysing the data generated from the evaluation studies.

Finally, thanks to the numerous participants in the evaluation studies. These include students from the Computer Science Department at the University of Canterbury and students from the School of Mathematics and Computer Sciences at Victoria University, Wellington.

# Contents

# Abstract

Database (DB) modelling, like other analysis and design tasks, can only be learnt through extensive practice. Conventionally, DB modelling is taught in a classroom environment where the instructor demonstrates the task using typical examples and students practise modelling in labs or tutorials. Although one-to-one human tutoring is the most effective mode of teaching, there will never be sufficient resources to provide individualised attention to each and every student. However, Intelligent Teaching Systems (ITS) offer bright prospects to fulfilling the goal of providing individualised pedagogical sessions to all students. Studies have shown that ITSs with problem-solving environments are ideal tools for enhancing learning in domains where extensive practice is essential. This thesis describes the design, implementation and evaluation of an ITS named KERMIT, developed for the popular database modelling technique, Entity Relationship (ER) modelling.

KERMIT, the Knowledge-based Entity Relationship Modelling Intelligent Tutor, is developed as a problem-solving environment in which students can practice their ER modelling skills with the individualised assistance of the system. KERMIT presents a description of a scenario for which the student models a database using ER modelling constructs. The student can ask for guidance from the system during any stage of the problem solving process, and KERMIT evaluates the solution and presents feedback on its errors. The system adapts to each individual student by providing individualised hint messages and selecting new problems that best suit the student.

The effectiveness of KERMIT was tested by three evaluations. The first was a think-aloud study to gain first-hand experience of the student's perception of the system. The second study, conducted as a classroom experiment, yielded some positive results, considering the time limitations and the instabilities of the system.

The third evaluation, a similar classroom experiment, clearly demonstrated the effectiveness of KERMIT as a teaching system. Students were divided into an experimental group that interacted with KERMIT and a control group that used a conventional drawing tool to practice ER modelling. Both group's learning was monitored by pre- and post-tests, and a questionnaire recorded their perception of the system. The results of the study showed that students using KERMIT showed a

significantly higher gain in their post-test. Their responses to the questionnaire reaffirmed their positive perception of KERMIT. The usefulness of feedback from the system and the amount learnt from the system was also on a significantly higher scale. Their free-form comments were also very positive.

# Chapter 1

# Introduction

Over the past thirty years, considerable research has been invested in developing software systems that effectively teach students. These software systems are called Intelligent Tutoring Systems (ITS). The need for such systems began with the observation that individualised teaching is the most powerful instructional medium. Empirical studies have shown that individualised human tutoring over normal classroom instruction could raise students' achievement by two standard deviations [Bloom, 1984]. However, having individualised human tutoring for each student is not logically and financially possible since students greatly outnumber teachers. If ITSs can be developed to 'emulate' a human tutor, then one-to-one tutoring for all students would become a possibility.

To emulate a human tutor, the computer would have to be programmed with domain knowledge and pedagogy. Natural language processing (NLP) would be used to communicate with the student in a similar way in which a student communicates with a human tutor. However, this initial approach of emulating a human teacher has proven to be an overly ambitious goal. Firstly, NLP is yet to be sufficiently effective to support a natural language conversation between a student and a tutor. As a result, most researchers have chosen to develop ITSs with alternate interfaces. Secondly, it was realised that a computer program can never replace a human tutor. There will always be students for whom the software is not suitable. The role of computers in teaching has therefore been redefined to the more realistic goal of complementing the teacher. ITSs can be used as a complement to classroom teaching in that students are able to learn at their own pace. By using ITSs as support tools for learning, teachers will have more time to focus on one-to-one tutoring for students who are struggling.

This thesis describes an intelligent teaching system developed for students learning database modelling. The introductory chapter contains a high-level overview of the remainder of the thesis. ITSs are described in detail in Section 1.1. Section 1.2 discusses the central problem; specifically,

the difficulties that students experience in learning databases. A solution is proposed in Section 1.3: An ITS for database modelling. Students using such a system can individually learn database modelling from the customised responses received from the system. Section 1.4 outlines the remainder of the thesis.

## 1.1 Intelligent Tutoring Systems

Intelligent Teaching Systems dynamically reason about and customise their response to each individual student. In this regard they are distinguished from earlier instructional systems such as computer-based instruction (CBI) and computer aided instruction (CAI). In traditional CAI systems, all feedback is organised into blocks or chunks called *frames*, and encoded branching instructions define both the topic and feedback to be presented. If the student answers the set of questions correctly, then the next frame in the sequence is presented. If the student answers the questions incorrectly, an alternative screen is presented. These sequences are static and predefined by the author.

In ITSs the domain expertise of the subject matter is not represented merely as a set of *frames*, but actually as a dynamic model of the domain knowledge, with a set of rules by which the system can reason. These systems have the ability to evaluate multiple correct solutions rather than a single idealized expert solution. The ITS reasons about its pedagogical knowledge and then dynamically generates its own path through the knowledge in order to respond to the student's behaviour.

Most ITSs are designed with the implicit assumption that the learner will *learn-by-doing*. Typically they present a problem to the student and offer explanations and hints in response to the student's behaviour. In this regard ITSs are ideal tools for teaching domains where practice is essential. The student can practise problem solving at his or her own pace with individualised guidance from the system.

There are a number of success stories regarding intelligent tutors. In a special case, students working with an Air Force troubleshooting tutor for only 20 hours gained proficiency equivalent to 40 months (almost 4 years) of on-the-job experience [Lesgold, et al., 1990]. In another example, students using the LISP tutor completed programming exercises in 30% less time than those receiving traditional classroom instruction, and scored 43% higher in the final exam [Anderson, et al., 1996]. In a third study students who used the algebra tutor, PAT, outperformed the students in the comparison classes by 15% on standard tests and 100% on tests targeting the objectives focused on by the tutor [Koedinger, et al., 1997].

## 1.2 The problem: learning database modelling

Conceptual database design is an integral part of the database design process. The task demands translation of data requirements of an application into a data representation. The Entity Relationship (ER) model, originally proposed by P. Chen [Chen, 1976] is a popular conceptual database design methodology. It views a word as consisting of entities, and relationships between them. Since most information systems require a database, the correctness of the database design has a direct impact on the information systems that use the database.

Conceptual DB modelling is a complex task that requires DB design expertise. Although the ER model is considered to have simple constructs [Batra, et al., 1990], learners experience considerable difficulties in constructing ER models. This was confirmed by the author's observations during his personal experiences in tutoring a second year introduction to database course at the University of Canterbury. The lecturer of the course, who has been teaching database courses since 1991, has also observed that students find it difficult to grasp the concepts of database modelling.

Experiments conducted by Batra and colleagues have shown that although novices face little difficulty in modelling entities, they are challenged when modelling relationships [Batra, et al., 1990]. The root of these difficulties is their incomplete knowledge of database design [Batra & Antony, 1994]. Another source of difficulty experienced by novices is that, given a set of entities, there are potentially a large number of possible relationships [Batra & Antony, 1994]. For example, consider an ER model with five entities. There are ${}^5C_2$ or ten ways of connecting two entities with a relationship (binary relationship).

ER modelling is traditionally taught in a classroom environment where solutions to typical database problems are explained. Due to the limited time available for lectures, only a very few problems can be demonstrated. Group tutorials may accompany lectures, allowing students some opportunity of hands-on experience in constructing ER models, but the tutor can only provide limited individual assistance to the students.

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 be expected to gain expertise in the domain by attending lectures. Even if some effort is made to offer students individual help through tutorials, since a single tutor must cater for the needs of the entire group of students, 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.

As with many areas of systems analysis and design, ER modelling can only be learnt through practice. In the case of ER modelling it is difficult for students to practise solving problems with no guidance, as there is no one correct solution for a given problem. The students are not knowledgeable enough in the domain to validate their own answers.

## 1.3 A solution: an ITS for database modelling

Empirical studies [Corbett, et al.; Koedinger, et al., 1997; Mitrovic & Ohlsson, 1999] have shown that ITSs can effectively teach domains that require extensive practice. An ITS with a problem solving environment is an ideal tool for students learning database modelling. A learning environment which offers customised feedback for students will allow them to learn how to model databases at their own pace.

KERMIT, the **K**nowledge-based **E**ntity **R**elationship **M**odelling **I**ntelligent **T**utor, is an ITS designed for teaching ER modelling and is fully implemented by the author. KERMIT is developed as a problem-solving environment for ER modelling. The system presents a description of a scenario, which the student has to model using the ER modelling constructs. Whenever guidance is requested from the system, KERMIT evaluates the student's solution and presents hints regarding any errors that were discovered. The main aim of the system is to individualise instruction towards each student.

The effectiveness and the students' perception of KERMIT were evaluated during three evaluation studies. The initial study was conducted in the form of think-aloud protocols and did not result in concrete data from which to deduce any inferences about the system. The second evaluation was conducted as a classroom experiment, where the students' learning was monitored by pre- and post-tests and their perception was recorded in questionnaires. The study did not yield any significant results mainly due to the influence of the limited time that was available for the study. It, however resulted in some positive results, including the students who used KERMIT demonstrating a slightly higher gain in their post-test.

The third evaluation, conducted in a similar manner to Study 2, resulted in a number of statistically significant results, with students using KERMIT recording a significantly higher gain in their post-test in comparison to the control group. This suggests that the students who used KERMIT gained more knowledge on ER modelling within a time period similar to the control group. The

student's responses to the questionnaire also demonstrated the students' positive perception of KERMIT. Students who used KERMIT indicated that they gained more knowledge from KERMIT and that KERMIT's feedback was more useful in comparison to the control system.

## 1.4 A guide to the thesis

The context for this thesis is outlined in Chapter 2, focusing on the architecture of a ITS and describing its components and functionalities. In particular it explains Constraint Based Modelling (CBM), the student modelling technique implemented in KERMIT. Chapter 3 supplies the remainder of the background, describing database modelling and teaching it. Firstly, Entity Relationship modelling, the specific high-level database modelling technique implemented in KERMIT, is outlined. Secondly, existing database modelling tools including CASE tools and knowledge-based CASE tools are surveyed. Finally, a survey of computer-based teaching systems for database modelling is presented. In Chapter 4, the design and implementation details of KERMIT are presented. The chapter introduces KERMIT's architecture and describes each of its components in detail. It also describes the authoring tool that was developed for adding new problems to KERMIT. The results of the extensive evaluations are detailed in Chapter 5. It describes the preliminary evaluation and the results of the two classroom evaluations that were conducted at the University of Victoria and University of Canterbury. Finally, Chapter 6 presents the conclusions and discusses future directions of this research project.

# Chapter 2

# ITS Architecture

Intelligent Tutoring Systems make inferences about student knowledge and interact intelligently with students based upon individual representations of their knowledge. The architecture of a typical ITS, as shown in Figure 2.1, consists of four main components: a *domain module*, a *student modeller*, a *pedagogical module* and an *interface*. Typically these four modules combine to generate a tutoring system's intelligence.

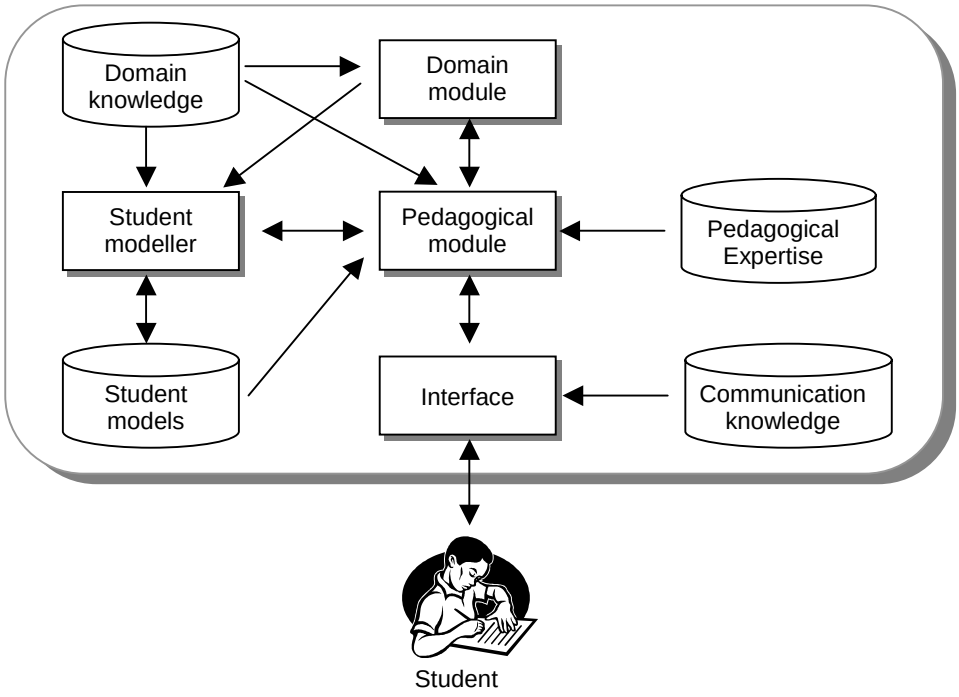



Figure 2.1: Typical architecture of an ITS

The student using the system interacts with its interface. Depending on the system's implementation, either the student may make a request for his/her solution to be evaluated, or the pedagogical module (PM) may decide to evaluate the student's solution or problem state. Evaluating a solution triggers a number of actions within the teaching system. Initially, the domain module solves the problem given to the student and produces a correct solution. The student modeller then compares the system's solution against the student's solution, making use of the domain knowledge. The student modeller updates the student model to reflect the student's new state of knowledge. The PM would then use its pedagogical knowledge of the system to provide feedback to the student, or to select a new topic that best suits the student.

Building a teaching system demands building each of the four modules using a variety of artificial intelligence technologies including *knowledge representation*, *control* and *knowledge acquisition* [Woolf, 1992]. *Knowledge representation* is how knowledge is used to model the domain, human thinking, learning processes and teaching strategies. Two aspects of knowledge representation are required for building ITSs. Firstly, the knowledge that teachers use to understand the domain, diagnose student behaviour and select new topics should be discovered. Secondly, a good representational scheme for encoding the domain knowledge should also be established. A knowledge base may store concepts, activities and relations between topics, etc.

*Control* refers to interpreting the knowledge base and finding appropriate pieces of knowledge to make a diagnosis, a prediction or an evaluation. It is essential that the knowledge base is separated from the instructional strategies and control (heuristics) to enable the developer to work with each module independently. This enables the developer to add new knowledge without concentrating on the control schemes.

*Knowledge acquisition* is a complicated task for any ITS. It requires identification and encoding of expertise. *Knowledge acquisition* is an incremental process in which the discovered knowledge is added to the existing knowledge base. A knowledge base may contain questions, examples, analogies and explanations. It should also contain tutoring primitives such as topics, and also reasoning on how and when to present each primitive.

The remainder of the chapter takes a detailed look at each of the five main components of a typical ITS architecture. Section 2.1 describes the domain module. The critical component of an ITS, the student modeller, is described in Section 2.2. It describes some popular short term and long term student modelling methodologies. In particular the section outlines constraint based modelling and overlay modelling, which are the student modelling techniques used in KERMIT. The pedagogical module is detailed in Section 2.3. The description includes brief accounts of various

pedagogical strategies used in ITSs. Section 2.4 discusses the interface, outlining the desirable qualities of an effective interface. It also describes animated agents, which are lifelike animated characters that inhabit learning environments.

## 2.1 Domain module

The *domain module* uses the knowledge of experts (facts and rules of the domain) to solve problems in the domain. The quality of the domain knowledge may range from expert knowledge sufficient to solve the problem itself, through to a subset of the expert knowledge adequate for the purposes of teaching. Expert knowledge can be represented in various ways, including semantic networks, frames, production systems and constraints. Choosing a representation for the domain knowledge is critical, since typically the knowledge representation is used to solve the given problem and to explain the solution. This suggests that the representation should be powerful enough for problem solving and should be useful for explanations.

There are two ways in which knowledge can be represented in a tutor system: *black box* and *glass box*. Earlier tutoring systems used the *black box* form in which the problem solving methodology was hidden from the student. As an example consider SOPHIE I [Brown, et al., 1975], an earlier teaching system developed for the domain of electrical circuits. It could answer any question that the student posed about electrical measurement values of any point in a complex circuit. However, as a result of using complex simulation techniques to derive electrical measurements, it was not able explain the reasoning behind how the reported values were calculated. These explanations are extremely important to enable students to learn the concepts of the domain, and only providing the correct solution is likely to encourage shallow learning.

More recent research work has focused on *glass box* models whose knowledge is represented in a way that more directly matches human capability, with the possibility of offering richer explanations to the student. The teaching systems called *cognitive tutors*, developed at the Advanced Computer Tutoring Project at Carnegie Mellon University have implemented a *glass box* model for representing their knowledge. Cognitive tutors trace the student's actions within a tree of all possible correct solutions and give hint messages or feedback if they divert from the correct solution path. For example, PACT Algebra II Tutor's [Corbett, et al., 1998] knowledge base enables the system to trace the student's solution path. The system uses this trace to provide feedback or hints on problem solving actions when the student makes an error.

## 2.2 Student modeller

The student modeller evaluates the student's solutions and dynamically develops a representation of the current state of the student's knowledge and skill. The representation is called a student model (SM), and is developed by deducing the student's knowledge from their interactions with the system. It includes long-term knowledge, such as an estimation of the student's domain mastery, as well as short-term knowledge, such as the errors that the student has made in their most recent attempt. The student model should dynamically indicate the system's views of the student's strengths and weaknesses, as well as currently misunderstood knowledge. More specifically, a SM may contain attributes such as whether the student needs more challenges or more corrective advice. Other factors such as motivation can also be modelled. Typically, the long-term student model is saved when a student logs out of a teaching system, and is reloaded when he/she logs in again.

The student modeller is the most critical part of an Intelligent Teaching System. If the student is modelled in such a way that the characteristics of the student are not even approximated, then the quality of decisions made by the pedagogical module will be poor. Since the PM depends on the student model, a wrong approximation of the student would directly affect the pedagogical decisions made, regardless of the quality of the PM. Therefore, considerable research has been invested in discovering new student modelling methodologies.

Student models can be generated in either a top-down manner or a bottom-up manner. ITSs that use a model-driven approach, which attempts to model the student actions, builds student models in a top-down manner. Cognitive tutors [Anderson, et al., 1996] that trace the student's solution path from its knowledge base of all legal paths, build student models using a top-down approach. Conversely, systems that do not trace the student's actions and evaluate a final solution (data-driven) have to generate the student model in a bottom-up fashion. SQL-Tutor [Mitrovic, 1998c], a tutor developed for teaching the database language SQL, only evaluates the student's final state of the solution whenever the student requests guidance from the system.

Student models can be classified according to the persistence of their representation of the student's knowledge. While some student models contain a representation of the student's short-term knowledge, such as after a problem solving attempt, others build up a more long-term representation of the student's knowledge. Most ITSs implement a short-term as well as a long-term student model. The short-term student model is used to provide immediate help to the student,

such as providing hints or feedback, and the long-term student model is used for pedagogical actions such as selecting a new problem or a topic that best suits the student.

Sections 2.2.1 to 2.2.4 outline four popular student-modelling techniques: *model tracing*, *constraint based modelling* (CBM), *stereotypes* and *overlays*. *Model tracing* and CBM are short-term student modelling approaches. The main distinction between the two approaches is that in *model tracing* both procedural and declarative knowledge is represented, whereas in CBM only the declarative knowledge is represented. The two long-term student modelling approaches, *stereotypes* and *overlays*, are different in the amount of detail offered by each representation. *Stereotypes* are simple student models that are abstract classifications of students into groups. *Overlays* represent the student's domain knowledge as a subset of the domain expert's knowledge. In other words, it is a set of masteries over items of the domain.

### 2.2.1 Model tracing

Model tracing is based on the ACT-R [Anderson, et al., 1996] theory developed by Anderson and co-workers. The ACT-R cognition theory claims that there are two long-term memory stores: declarative memory and procedural memory. Declarative knowledge includes factual knowledge that the student uses (e.g. theorems in the mathematical domain) and is represented as *chunks*. Procedural knowledge includes goal-oriented knowledge (e.g. how to apply a mathematical theorem) and is represented as *production rules*. Model tracing also defines *buggy rules*, which describe incorrect or erroneous knowledge. There are eight principles that are derived from ACT-R theory, which are followed by model tracing tutors [Anderson, et al., 1996]. The key claim of the ACT-R theory is that "Cognitive skills are realised by production rules". Because of this claim, tutoring becomes the process of transferring production rules from the system to the student, so the students are tutored specifically on productions.

As an example, consider the set of production rules for computing the third angle of a triangle when two of the angles are known. The law governing the three angles of a triangle says that the sum of all three angles is $180^{o}$. The set of production rules, as shown in Figure 2.2, consists of two if-then rules. The first rule checks whether the current goal is to find all the angles of a triangle of which two of the angles are already known. If the check is satisfied, the rule sets a sub-goal to calculate the unknown angle of the triangle. Since production rules are procedural in nature, the second production rule only fires when the goal is to calculate the third angle of the triangle. In other words, the second rule will only fire after the first rule has been fired. The second rule

specifies the action of writing out the value of the third angle, which is calculated as ($180^o - \theta_1 - \theta_2$).

```
IF the goal is to find the sizes of all three angles (θ1, θ2
   and θ3) of a triangle and two angles (θ1, and θ2) are known
THEN set a sub-goal to calculate θ3 of triangle

IF the goal is to calculate θ3 of triangle
THEN write out θ3, where θ3 is (180o – θ1 – θ2)
```

Figure 2.2: Set of production rules for calculating the third angle of a triangle

The set of if-then rules used for calculating angles of a triangle, itemised in Figure 2.2, should also include *buggy rules* to detect erroneous solutions. In this example a simple *buggy rule* which identifies an erroneous solution, such as the rule illustrated in Figure 2.3, can be added. The rule simply catches situations when the student writes out a value for the third angle, which is not ($180^o - \theta_1 - \theta_2$).

```
IF the goal is to calculate θ3
THEN set a sub-goal to write out θ3, where θ3 is not
     (180o – θ1 – θ2)
```

Figure 2.3: Buggy rules for calculating the third angle of a triangle

The family of tutors developed using model tracing are called *cognitive tutors*. They are able to solve the problems in the domain and trace the student's solution path through a complex problem solving space. The tutors give immediate feedback on the student's problem solving performance, in addition to instructions on the underlying knowledge required for problem solving. Additionally, they have an explicit goal-structure of the problem, in order for the students to master the abstract as well as the concrete skills of the domain.

Cognitive tutors have been developed for a number of domains including algebra [Corbett, et al., 1998; Koedinger, et al., 1997] and geometry [Aleven & Koedinger, 2000]. Both the PACT Algebra tutor and the PACT Geometry tutor solve problems using production sets. They are able to trace the student's solution path and offer feedback on their performance. The tutors have been coupled with a long-term student modelling technique in order to represent the student's long-term knowledge in the domain.

A major issue in developing cognitive tutors is knowledge acquisition. Anderson and co-workers reported that an estimated time of ten hours or more was required to produce a single production rule. It is not economical or feasible to model domains that are larger and more complex than algebra and geometry. Moreover, although the approach of modelling all possible solution paths works for well-defined domains such as mathematics, it may not be realistic where the domain is ill-defined (e.g. database modelling).

Having to compose a bug library with a collection of *buggy rules* is also a major obstacle of model tracing that must be overcome. A bug library customised for a particular group of students may not be appropriate for another group, as different groups of students tend to make different mistakes. In this regard, composing a bug library that is robust is an extremely hard task, if not impossible. Furthermore, identifying typical errors that the students make is a labour intensive process since the space of incorrect knowledge is larger than correct knowledge.

### 2.2.2 Constraint based modelling

Constraint based modelling was introduced by Ohlsson [Ohlsson, 1994], and is based on his theory of learning from performance errors [Ohlsson, 1996]. CBM focuses on erroneous knowledge rather than describing the student's knowledge as in model tracing. The key assumption in CBM is that diagnostic information is in the problem state at which the student has arrived and not in the sequence of his/her actions. This assumption is supported by the fact that no correct solution can be arrived at by traversing a problem state that violates fundamental ideas or concepts of the domain.

Since the space of false knowledge is much greater than correct knowledge, in constraint based modelling knowledge about a domain is represented as a set of constraints on correct solutions. The set of constraints identify the correct solutions from the space of all possible solutions. CBM, unlike model tracing, only represents declarative knowledge. More precisely, CBM only represents factual knowledge of the domain such theorems.

The unit of knowledge in CBM is called a state constraint. Each constraint is an ordered pair $<C_r, C_s>$, where $C_r$ is the *relevance* condition and $C_s$ is the *satisfaction* condition. The *relevance* condition identifies the states in which the constraint is relevant and the *satisfaction* condition identifies the states in which the constraint is satisfied.

Consider the same example presented in Section 2.2.1, the law governing the sum of the three angles of a triangle in the domain of geometry. Suppose the $C_r$, the applicability condition, of the hypothetical constraint is the condition where two ($\theta_1$ and $\theta_2$) of the three angles ($\theta_1$, $\theta_2$ and $\theta_3$) are known. The $C_r$ would match any triangle with two known angles (examples are illustrated in Figure

2.4). The $C_s$ that defines the correctness of the solution is $\theta_3 = 180^o - (\theta_1 + \theta_2)$. In other words, $C_s$ defines that the third angle or the unknown angle should be the result obtained by subtracting the sum of the two known angles from $180^o$. The relevance condition ($C_r$) and the satisfaction condition ($C_s$) of the constraint are illustrated in Figure 2.5.

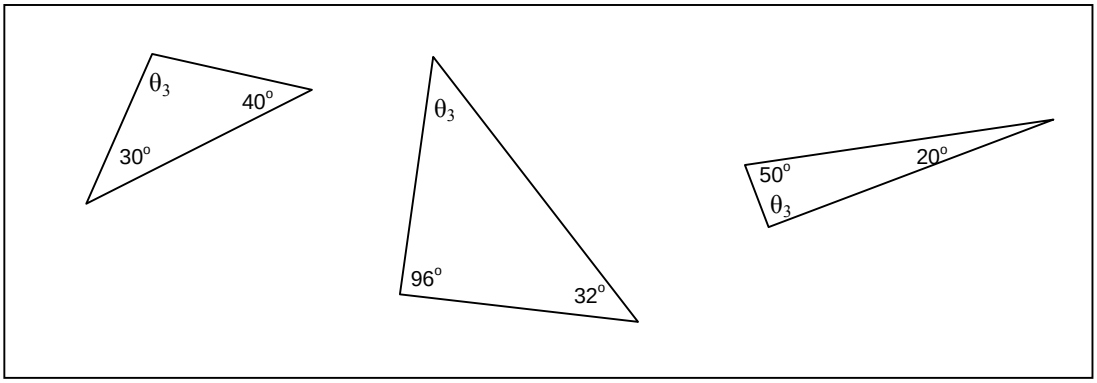



Figure 2.4: Examples of triangles with two known angles

$C_r$: The student is trying to compute the angles of a triangle and two angles ($\theta_1$ and $\theta_2$) are already known

$C_s$: The third angle ($\theta_3$) should be equal to $180^o - \theta_1 - \theta_2$

Figure 2.5: A constraint for the angles of a triangle

A state constraint can be represented as a pair of patterns, where each pattern is a list of elementary propositions combined with conjunctions or disjunctions. Alternatively, constraints can be implemented as pairs of functional predicates. To illustrate, consider the example in Figure 2.5. The $C_r$ may be implemented as `knownAngles(triangleIdentifier) = 2`, where `knownAngles` is a function that takes the identifier of a specific triangle as its argument and returns its number of known angles.

The computations required for testing whether a constraint has been satisfied or violated is straightforward: the algorithm is depicted in Figure 2.6. When evaluating a constraint, initially the student's solution (SS) is matched against $C_r$. The $C_s$ is only evaluated if the SS satisfies $C_r$, and if $C_r$ is not satisfied then the constraint is ignored. The constant is labelled as satisfied if SS satisfies $C_s$, or else is labelled violated.

CBM also requires little computational effort for constraint matching, since checking $C_r$ and $C_s$ is essentially pattern matching. Furthermore, an algorithm exists for merging constraints into a structure similar to a RETE network that increases the efficiency of constraint matching [Mitrovic, 1998a].

One other advantage of CBM over model-tracing is the considerably less time required to acquire knowledge. Mitrovic reported that she required an average of 1.1 hours of work to identify a constraint [Mitrovic & Ohlsson, 1999]. This is a significant saving when compared to Anderson's co-workers requiring ten or more hours to identify a production rule.

Constraints also offer a solution to one of the problem solving bottlenecks: the problem of identifying multiple correct solutions. The model tracing approach requires the enumeration of all correct solutions to a given problem. Constraints, however, can be used to identify all possible correct solutions relevant to a particular question. For example in the Entity Relationship modelling domain, an attribute with more than one value that belongs to entity E may be represented as a multivalued attribute or as a weak entity that has its owner as entity E.

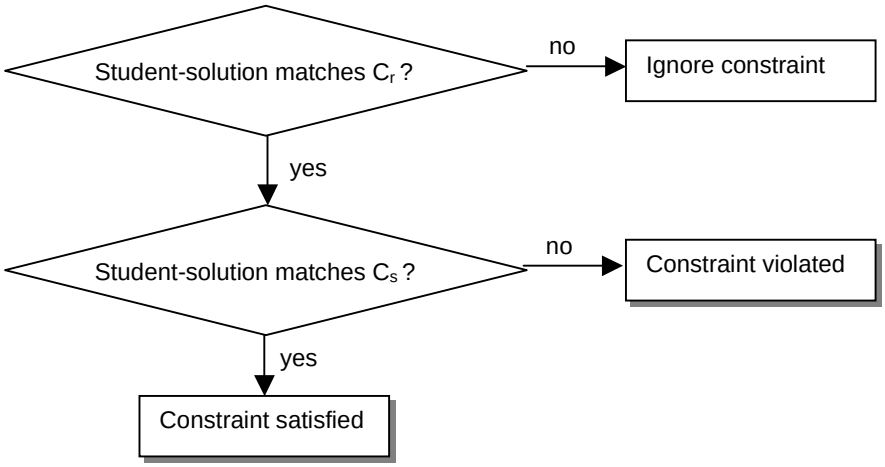



Figure 2.6: Algorithm to evaluate a constraint

Another major advantage of CBM over model tracing is that it does not require modelling of incorrect or erroneous knowledge. In model tracing incorrect knowledge has to be modelled as *buggy rules*. As discussed in Section 2.2.1, building a bug library is an intractable task.

Constraint based tutors have been developed for the domain of SQL, database modelling and punctuation [Mitrovic, 1998c; Mitrovic, et al., 2001b]. SQL-Tutor [Mitrovic, 1998b; Mitrovic, 1998c] is the teaching system developed for university level students that implements CBM for the popular database language, SQL. The ITS developed for the domain of database modelling, called KERMIT [Suraweera & Mitrovic, 2001], is also aimed at university level students. The system

teaches Entity Relationship modelling. CAPIT [Mayo, et al., 2000] is an ITS developed for the domain of capitalisation and punctuation, focusing on 10-11 year old students.

The difference in the timing of the presentation of feedback to the student is also a contrasting feature of cognitive or model tracing tutors and constraint based tutors. Cognitive tutors provide feedback immediately after a student has made a mistake, while CBM tutors only provide feedback when the student initiates a completion routine such as submitting the solution, or when he/she requests help from the system. Immediate feedback can be effective and efficient for novices since they learn the domain quicker with strict guidance from the system. However, it tends to restrict the student and encourages shallow learning. Delayed feedback, as implemented in CBM tutors, is unlikely to encourage shallow learning as the students are given an opportunity to reflect on their actions. CBM tutors also tend to be less restrictive as they allow the student to experiment with solving the problem and only evaluate the final state of their solution. However, they may not be ideal for novices who may feel lost with such loose guidance.

### 2.2.3 Stereotypes

Stereotyping is the simplest form of student modelling with students being assigned to a specific stereotype. There are two types of stereotypes: *fixed* [Winter & McCalla, 1999] and *default* [Kay, 2000]. Fixed stereotyping is a simple approach to student modelling, with the student cast as a predefined stereotype according to his/her responses. A basic method of fixed stereotyping is to assign a level to a student depending on their performance. This student modelling approach makes the broad assumption that all students within a stereotype possess the same domain knowledge and display the same problem solving behaviour. The system may move a student from one stereotype to the other, but the stereotypes themselves do not change. Although the approach is not useful for complex analyses, it is a realistic student modelling technique for open domains where knowledge cannot be decomposed into atomic units.

Default stereotypes are a more flexible approach. The student is initially assigned a default stereotype, which according to their responses, is gradually replaced by more individualised settings. The initial stereotypes can be used as starting values for a more complex student model such as an overlay model (discussed in Section 2.2.4). Kay has surveyed a number of such systems [Kay, 2000]. StyLE-OLM [Dimitrova, et al., 1999] is a learning environment for scientific terminology that uses this approach. The system engages in a natural language dialogue with the student and, using its rules, draws conclusions on the student's knowledge based on the student's responses. However, these default inferences can be debated in dialogue with the student.

### 2.2.4 Overlays

Overlay models represent the student's knowledge as a subset of the domain expert's knowledge. The model simply estimates the mastery of each element in the domain that an expert would be expected to know. The only precondition that needs to be considered for applying this approach is that the domain knowledge must be able to be broken down into generic items such as rules, concepts, facts, etc. An example of an overlay model for a domain that is decomposed into ten items is visualised in Figure 2.7.

The mastery of each item ranges from 0 to 100%. An expert of the domain is expected to master each item at 100%. Typically, the overlay model of the student is initially assigned 0% mastery for each item. The mastery changes dynamically according to the student's behaviour. In ITSs that have combined stereotypes with overlays, the overlay model would be initialised to the default stereotype.

A number of ITSs have used overlay models as their long-term student modelling methodology. SQL-Tutor [Mitrovic, 1998b; Mitrovic, 1998c], which implements CBM for short-term student modelling, uses an overlay model for modelling the student's long-term knowledge. Recent developments have used Bayesian networks to obtain more accurate probabilistic overlay models. CAPIT [Mayo, et al., 2000], the teaching system for capitalisation and punctuation, uses its Bayesian network as its long-term student model to select new problems for the student.

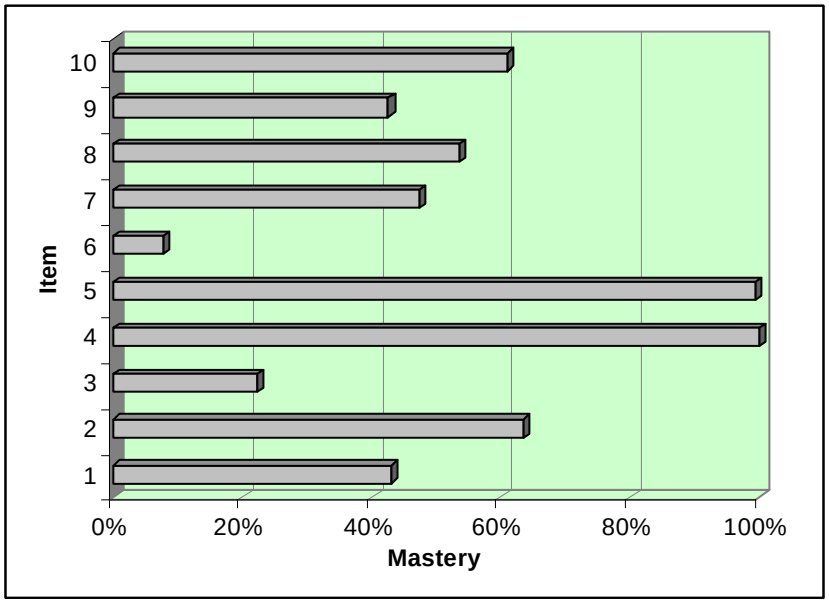



Figure 2.7: An overlay student model

## 2.3 Pedagogical module

The pedagogical module (PM) is the driving engine of the teaching system and is closely linked to the student modeller. It designs and controls instructional interactions with the student. It uses the student model and the domain knowledge to make its pedagogical decisions. The actions performed by the PM can be categorised as diagnostic and didactic. In other words, the PM forms and updates the student model and performs teaching actions that suit the SM, such as offering hints when the student is struggling, supplying advice, support and explanations, selecting a new topic, etc.

Pedagogical strategies range from the two extremes of teaching: didactic and discovery oriented learning. The didactic approach is a formal and more traditional method of teaching in which the learner is instructed, and the tasks are strongly goal oriented. Tutoring systems that adopt a didactic approach initiate and control the student's activity. All the activity in these systems is focused on the system's instructional goals. This traditional method of teaching is effective for novices since they require close guidance. However, knowledgeable students may find themselves restricted and not challenged.

Discovery oriented learning is a more informal teaching philosophy, involving learning from experience. Discovery learning environments are seen as having an advantage over the didactic approach in that they allow new knowledge to be constructed from direct concrete experiences in the concepts and capabilities that the learner already possesses. However, the drawback with such environments is that novices may take a long time to make the discoveries that are the goals of the system. Recent research has focused on guiding the discovery process in an effort to increase the efficiency of the learning process.

Some of the popular pedagogical strategies are *model tracing*, *computer coaching*, S*ocratic teaching* and *collaborative learning*. These strategies vary from formal, traditional teaching to open, discovery oriented learning. The relative positions of each tutorial strategy on a scale ranging from traditional teaching to discovery oriented learning is illustrated in Figure 2.8.

*Model tracing* is based on the ACT-R theory proposed by Anderson [Anderson, et al., 1996]. Tutoring systems that implement *model tracing* trace the student's actions within a tree of all possible actions. The system is in complete control in such systems. Most *model tracing* tutors provide immediate feedback on errors. These tutors are efficient for novices since they correct the student immediately after they make an error. However, students with some experience may find these tutors restrictive since they are forced to solve problems in the way adopted by the system.

Moreover, this approach may also encourage shallow learning as students may not learn the underlying concepts of the domain.

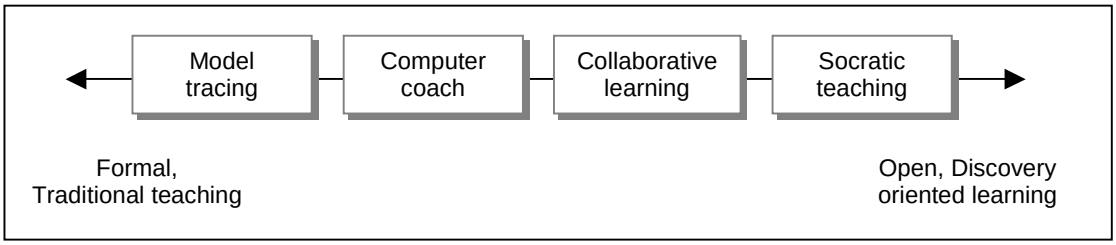



Figure 2.8: Pedagogical strategies for ITSs

The Practical Algebra Tutor [Koedinger, et al., 1997] developed at Carnegie Mellon University is a classic example of a model-tracing tutor. The tutor's interface consists of a worksheet on which the student enters the solutions to the given questions. If the student inserts an incorrect value the text is immediately highlighted to indicate that the entered value is incorrect.

The *computer coach* approach, in contrast to *model tracing*, provides delayed feedback. Systems that implement *model tracing* provide help only when the student requests it. This approach gives the student a sense of control over his/her learning experience. Teaching systems that have adopted this approach encourage the student to think about their solutions and self-explain the system's suggestions. These systems also discourage shallow learning, since they encourage the student to check their solutions before submitting them. One concern that is associated with *computer coaching* is whether students waste time attempting to solve problems, since the system does not give immediate feedback. Although this may be a cause of concern in the initial stages it may diminish eventually as the students learn more about the domain.

SQL-Tutor [Mitrovic & Ohlsson, 1999] is an example of a *computer coach* tutor. It provides feedback only when the student submits his/her solution. Once the student submits their solution, the system evaluates it and presents feedback depending on the solution. Although the student is in complete control of the timing of the feedback, the system selects new problems for the student.

Students are given total freedom in systems that implement *Socratic teaching*. Typically, the system leads the student to form general principles by posing questions and counter-examples. Teaching systems that implement *Socratic teaching* are designed as discovery learning environments in which the student is allowed to freely explore the problem space. Although knowledgeable students would find themselves challenged by such teaching systems, novices would easily become 'lost' in such environments. ITSs that implement this teaching approach are

considerably harder to develop since they have no control over the student's actions within the environment.

*Collaborative learning* tutors are designed with the philosophy of learning from peers. In order for the system to succeed as an effective teaching system, a number of students have to be logged in simultaneously. The students learn from each other and may also learn from the system. Some systems evaluate students' answers, whereas others tend to be tools for collaboratively solving problems in a particular domain.

COLER [Constantino-Gonzalez & Suthers, 1999; Constantino-Gonzalez & Suthers, 2000] is a teaching system developed to collaboratively learn database modelling. The system allows a number of students to log onto the system and collaboratively construct a database model that satisfies the given requirements. A construct is added to the collaborative model only if every participant gives consent for it to be included. Each participant has a private workspace, and if one has a database model that is considerably different to the collaborative database model, the system intervenes and suggests that it is shared with the others.

## 2.4 Interface

The interface acts as a mediator between the student and the tutoring system, enabling a student to conduct dialogues with the system. The teaching system uses the interface to teach principles of the domain. As the interface is of utmost importance to the system it should be designed within the principles of good interface design.

There are a number of essential characteristics that should be present in a teaching system interface. Firstly, it must provide motivation for the student to continue. Motivation is an essential ingredient for an effective ITS, as then the student will spend more time interacting with the system, with a resultant increase in knowledge acquisition.

Secondly, the interface can improve learning significantly by reducing the student's working memory load. If only a single component of a particular problem is the teaching focus with the rest of the problem made available within the user interface, then the student only has to concentrate on that and does not need to remember the irrelevant details. For example, consider SQL-Tutor [Mitrovic, 1998b; Mitrovic, 1998c]. The interface of the Windows version of SQL-Tutor is illustrated in Figure 2.9. It is essential to know the database schema to compose SQL statements. The interface frees the student from the burden of having to remember the database schema by displaying the relevant schema in the bottom section of the interface. The student only has to concentrate on composing the correct SQL statement for the given problem.

Thirdly, a good interface should visualise the goal structure of solving the problem, to help the student complete the task. SQL-Tutor's interface [Mitrovic, et al., 2001b] is a prime example of such an interface. As shown in Figure 2.9, the middle part of SQL-Tutor's interface contains the clauses of the SQL SELECT statement. The student's task of composing an SQL statement that satisfies the given problem is made easier since they simply have to fill-in the blank clauses.

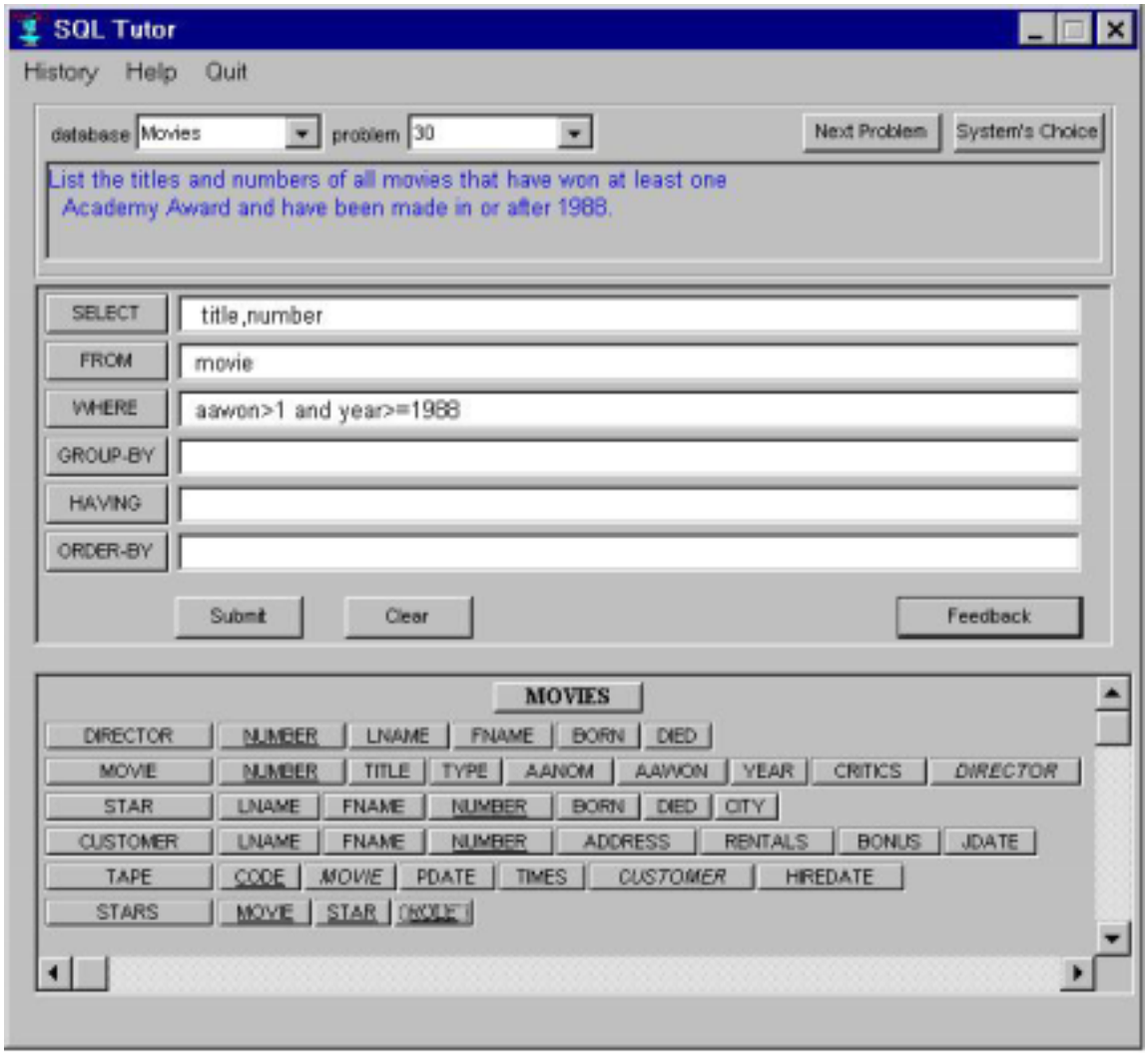



Figure 2.9: Interface of SQL-Tutor

Fourthly, an interface should structure the student's thinking towards achieving the goals of the system. SQL-Tutor's interface, illustrated in Figure 2.9, structures the student's thinking in composing SQL queries by having six text boxes for the six clauses of an SQL query, forcing them to input each clause separately. Moreover, the student is made aware that an SQL statement can contain up to six clauses and cannot contain any other clauses.

Recent ITSs have introduced lifelike animated characters to their interfaces to support learning. These characters are called animated pedagogical agents. They broaden the communication channel by using emotive facial expressions and body movements, which are very appealing to students. Pedagogical agents are extremely important for student motivation, as they provide advice and encouragement, empathize with students, and increase the credibility and utility of a system. Several studies have investigated the affective impact of agents on student learning and revealed the *persona effect*, which is that "the presence of a lifelike character in an interactive learning environment - even one that is not expressive - can have a strong positive effect on the student's perception of their learning experience" [Lester, et al., 1997a]. Experiments [Lester, et al., 1997b; Mitrovic & Suraweera, 2000; Towns, et al., 1998] have shown that students are much more motivated when the agent is present, tend to interact more frequently and find agents very helpful, credible and entertaining.

# Chapter 3

# Teaching Database Modelling

Database design is defined by Batini and co-workers to be the task of "designing the structure of a database in a given environment of users and applications such that all users' data requirements and all applications' process requirements are 'best satisfied'" [Batini, et al., 1986]. The process involves four distinguishable stages: requirements specification, conceptual design, logical design and physical design. Requirement specification involves identifying the information needs of various users or groups. The conceptual design phase models the users' and applications' views of information and may include a specification of the processing of information or use of the information [Batini, et al., 1986]. The goal of this stage is to produce a formal and accurate representation of the database requirements that is independent of any database management system (DBMS). The conceptual model is translated into the logical model of the chosen DBMS, such as a relational data model, during the logical design phase. Finally the logical data model is transformed into a form suitable for the specific DBMS during the physical design phase.

Database modelling is an essential part of the database design process. It allows the designer to obtain an increased understanding of the problem and to identify the basic components from which the solution will be built. A good model will allow the development of a database that is flexible and supports new features as they become necessary. Furthermore, a flexible reusable database model promotes stability, and therefore minimises the need to revise the database as new applications are created. Finally, a database built from a model that accurately depicts the establishment can be shared across all the establishment's functions, unlike a database that is built for a specific function.

The remainder of the chapter is organised as follows. Section 3.1 introduces Entity Relationship modelling, which is a popular database modelling technique. Computer Aided Software Engineering (CASE) tools developed for the database design process are described in Section 3.2. Conventional and knowledge-based tools are discussed, and a selection of currently commercial

non-knowledge based tools and knowledge-based CASE tools developed in research laboratories are outlined. Finally, Section 3.3 introduces computer based teaching systems developed for database modelling and gives a detailed account of two research attempts.

## 3.1 Entity Relationship modelling

Entity Relationship (ER) modelling is a popular database modelling technique, used frequently for the conceptual design of databases. Many CASE tools developed for database modelling employ ER modelling. ER modelling, originally proposed by P. Chen [Chen, 1976], views the world as consisting of entities and relationships between them. This section describes the basic data-structuring concepts of the ER model.

The ER model describes data as *entities*, *attributes* and *relationships*. Section 3.1.1 introduces *entities* and *attributes*, describing the types of *attributes* and the notion of an *entity type*. *Relationships* are introduced in Section 3.1.2. Finally, Section 3.1.3 describes *weak entities*.

### 3.1.1 Entities and attributes

An *entity* is the basic object represented in the ER model, which is a 'thing' in the real world with an independent existence. An *entity* can be an object with a physical existence such as a car or a person, or an object with a conceptual existence such as a company or a job. Each *entity* has particular properties, called *attributes*, that describe it. For example a student entity may be described by name, date of birth, address etc. A particular entity would have values for each of its attributes, which describe the entity.

A database usually contains groups of entities that are similar. For example a university database would contain personal information about all its students. These student entities share the same attributes but each individual student has their own values for each attribute. A group of entities that have the same attributes is called an *entity type* [Elmasri & Navathe, 1994]. Each *entity type* of the database is described by its unique name and its attributes. Conventionally the entity name is written in uppercase. As an example consider the student entity type with a number of attributes illustrated in Figure 3.1. The entity type is represented as a rectangle, while attributes are shown as ovals.

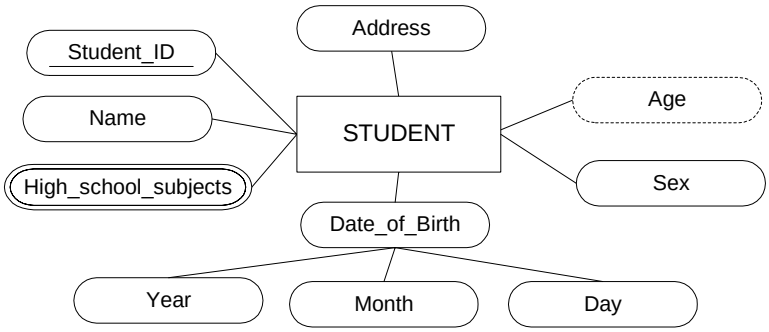



Figure 3.1: Student entity type

Attributes can be categorised into *simple* and *composite* according to their structure. *Simple* or *atomic* attributes are those that cannot be divided into parts with independent meanings. In other words if a simple attribute is divided, the divided parts would be meaningless. For example an attribute such as the sex of a student entity cannot be divided into parts with meaning. Conversely, *composite* attributes are those that can be divided into smaller subparts, which represent basic attributes with independent meanings. The date of birth attribute of the student entity type (refer to Figure 3.1) is a *composite* attribute with three components: year, month and day. *Composite* attributes are useful to model situations where sometimes the *composite* attribute itself is referred to as a unit and at other times its components are specifically referred to. Considering the example of date of birth, a user of the database may refer to a student's date of birth as a unit or may refer to the year in which a student was born.

The type of the attribute differs according to the number of values it stores: *single valued* and *multivalued*. A *single valued* attribute can hold only a single value, whereas *multivalued* attributes can hold more than one value. For example, suppose students take more than one subject for their high school examinations. One student may take three subjects; another may take five. The high school subjects that were taken by a particular student are represented as a multivalued attribute in Figure 3.1. As illustrated in the figure, *multivalued* attributes are represented as an oval with double lines.

Most attributes are stored in the database. However, there are attributes that are not stored in the database, but are derived from other attributes. These are named *derived* attributes. The age of a student is an ideal example. A student's birth date is *stored* in the database, and his/her age is calculated from this. *Derived* attributes, as illustrated in Figure 3.1, are represented in ER modelling as an oval with a broken outline.

An attribute whose values are distinct for each individual entity is called a *key* attribute. The value of a key attribute can be used to identify each entity uniquely. The student ID of the student entity type discussed earlier is a key attribute. Each student is assigned a unique student number when enrolling at university. A key attribute has its name underlined inside the oval, as illustrated in Figure 3.1. An entity must have at least one key attribute.

### 3.1.2 Relationships

A relationship is an association between two or more entities. More generally, a relationship type defines a set of associations between two or more entity types. Consider the example depicted in Figure 3.2. It illustrates the LIVES_IN relationship, represented as a diamond, between the STUDENT entity type and the HALL entity type, which signifies that students live in halls.


Figure 3.2: LIVES_IN binary relationship

The number of relationship instances in which an entity can participate is specified by the cardinality ratio [Elmasri & Navathe, 1994]. The LIVES_IN binary relationship STUDENT:HALL is of cardinality ratio N:1. This cardinality ratio means that many students can be related to a hall, but a hall can be related to only one student. The cardinality ratio is displayed as either '1' or 'N' above or below to the connection between the relationship and the entity, as shown in Figure 3.2.

Relationships also have a participation constraint that specifies whether the existence of an entity depends on its being related to another entity via the relationship type [Elmasri & Navathe, 1994]. There are two types of participation constraints: *total* and *partial*. The participation of the student entity type in the LIVES_IN relationship is specified as *partial*, meaning that some students live in halls, but not necessary all. On the contrary, hall entities participate *totally* in the LIVES_IN relationship, meaning that every hall has to have students (at least one) living in them.

The number of entities participating in a relationship is called the degree of a relationship. Hence, the LIVES_IN relationship is of degree two and can be called a binary relationship. An example of a relationship of degree three is illustrated in Figure 3.3. The ENROLLED_IN ternary relationship associates STUDENT, COURSE and DEPARTMENT entity types. The ENROLLED_IN relationship represents students enrolled in a course if the department that offers the course approves the enrolment.

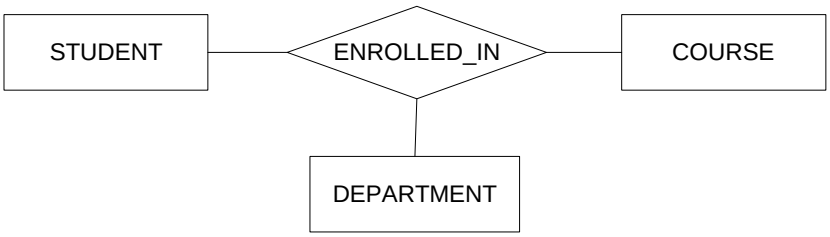



Figure 3.3: ENROLLED_IN ternary relationship

### 3.1.3 Weak entities

Entity types that do not possess a key attribute of their own are called weak entities. Weak entities are identified by being related to specific entities in combination with some of their attributes. The entity that identifies the weak entity is called the *identifying owner* and the relationship that relates the weak entity to its owner is called the *identifying relationship* [Elmasri & Navathe, 1994]. Weak entities have a *partial key*, which can be used in conjunction with its owner's key to uniquely identify the weak entity.

Figure 3.4 illustrates the weak entity NEXT_OF_KIN, related to the student entity, used to keep track of the student's next of kin. As illustrated in the figure, a weak entity is represented in ER modelling as a double lined rectangle. The NEXT_OF_KIN weak entity comprises four attributes. As it is a weak entity, it possesses a partial key attribute, *name*, represented by underlining its name with a broken line. This means that each next of kin of a student has a unique name.

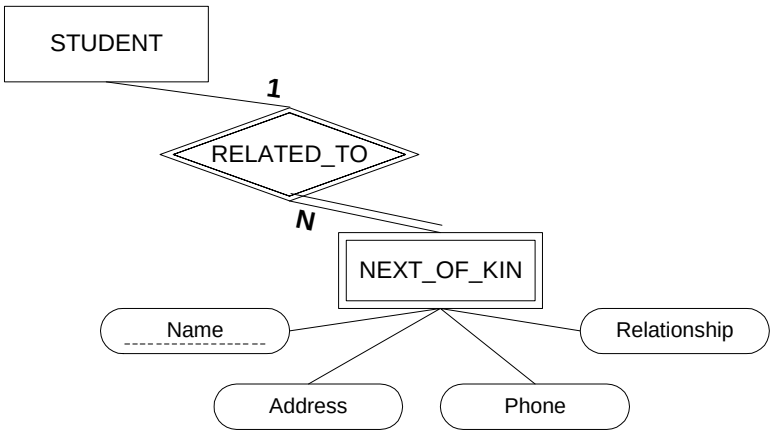



Figure 3.4: The weak entity NEXT_OF_KIN

The identifying relationship RELATED_TO is represented as a double lined diamond. The cardinality of the relationship specifies that a student can have a number of members as his/her next

of kin. As illustrated in the example, a weak entity must participate totally in the identifying relationship because the weak entity cannot be identified without its owner entity.

## 3.2 Database modelling tools

There are many CASE tools developed for database design. They can be categorised into two main groups depending on whether they possess any intelligence or not. None of the current commercial CASE tools possess any intelligence. Database design experts use them as tools during the process of database modelling. Conversely, knowledge based CASE (KB-CASE) tools are intelligent tools that are built by applying techniques from artificial intelligence with the goal of assisting database designers. KB-CASE tools are yet to be available commercially and are developed as research projects. This section presents a brief survey of computer-based tools used for modelling databases.

### 3.2.1 CASE tools for DB modelling

There are a number of commercial CASE tools developed for database design. This section concentrates on *Database Design Studio (DDS)*, *ER/Studio* and *MS Visio*, which are popular CASE tools for DB modelling.

*Database Design Studio* [DDS], developed by the company Chilli Source, is a CASE tool that supports Chen's ER model [Chen, 1976]. The interface of *DDS*, as illustrated in Figure 3.5, contains an ER diagram editor for the database design expert to construct an ER model. It allows the users to construct entities, relationships between them and specify the cardinalities of the participating entities. The tool is capable of automatically creating the logical schema as well as the physical schema. The logical schema is called a data structure diagram (DSD), which is dynamically created when the DSD pane is clicked. *DDS* has the ability to construct physical designs for most popular DBMS implementations. *DDS* is also able to identify a limited number of errors in the conceptual schema, such as entities that are missing a key attribute, during the process of creating the database schema.

*DDS*'s interface is intuitive to use and it allows the user to carry out all stages of the database process as a truly iterative process. Initially the user creates a conceptual schema and at its completion may view the logical schema. After viewing the logical schema he/she may decide to modify the initial conceptual schema. The system allows the user to switch between the conceptual schema and the logical schema and make modifications on one schema that would be reflected on the other. The user is given the freedom to carry out the process of database design as an iterative

process where any number of modifications on either the conceptual model or the logical model can be made until the final database is satisfactory.

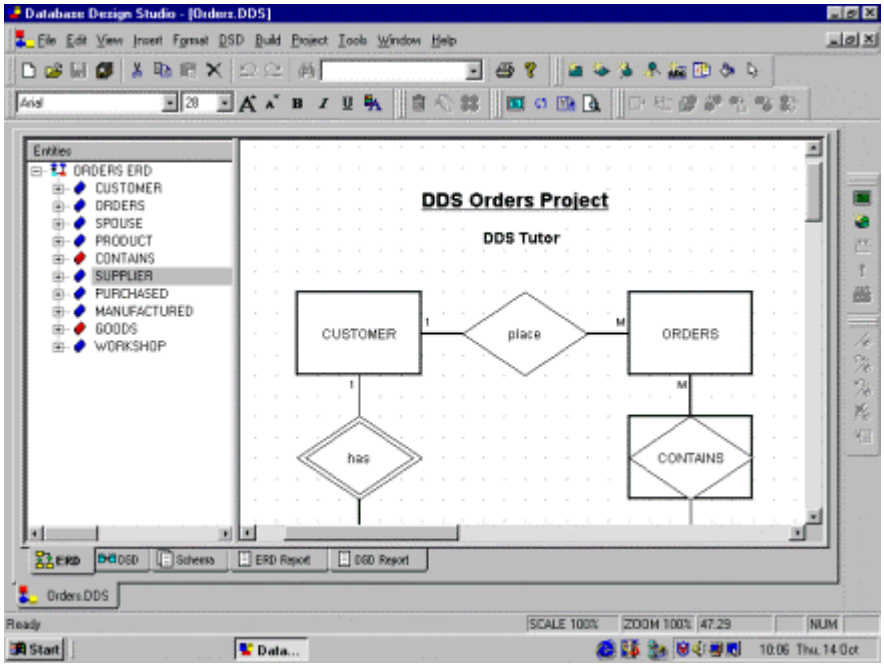



Figure 3.5: The user interface of *DDS*[1]

Embarcadero Technologies have developed a similar data modelling application named *ER/Studio* [ER/Studio]. It supports the conceptual modelling concepts of ER modelling initially proposed by Chen [Chen, 1976]. However, it uses a notation that is a modification of Chen's ER modelling notation. *ER/Studio* helps database designers to produce valid relational database models and enables database administrators to rapidly deploy and manage databases based on its models. *ER/Studio*, unlike *DDS*, is geared toward larger companies with complex data modelling requirements.

*ER/Studio* automatically generates the logical database schema and contains an interface (illustrated in Figure 3.6) that synchronises the conceptual and logical designs. The tool also allows the reverse engineering of databases. The system supports a variety of popular DBMSs.

*Visio* from Microsoft [Visio] is a tool that allows users to draw 2D diagrams by dragging and dropping objects from a template. These templates are called stencils and *Visio* contains a number of stencils for different notations. *Visio* also allows users to create their own stencils. However,

[1] Taken from http://www.5star-shareware.com/Business/DatabaseManagement/dds-screenshot.html

since *Visio* was designed as a generic drawing tool, it does not provide any specific assistance in database design. *Visio* will be further discussed in Section 4.3.1.

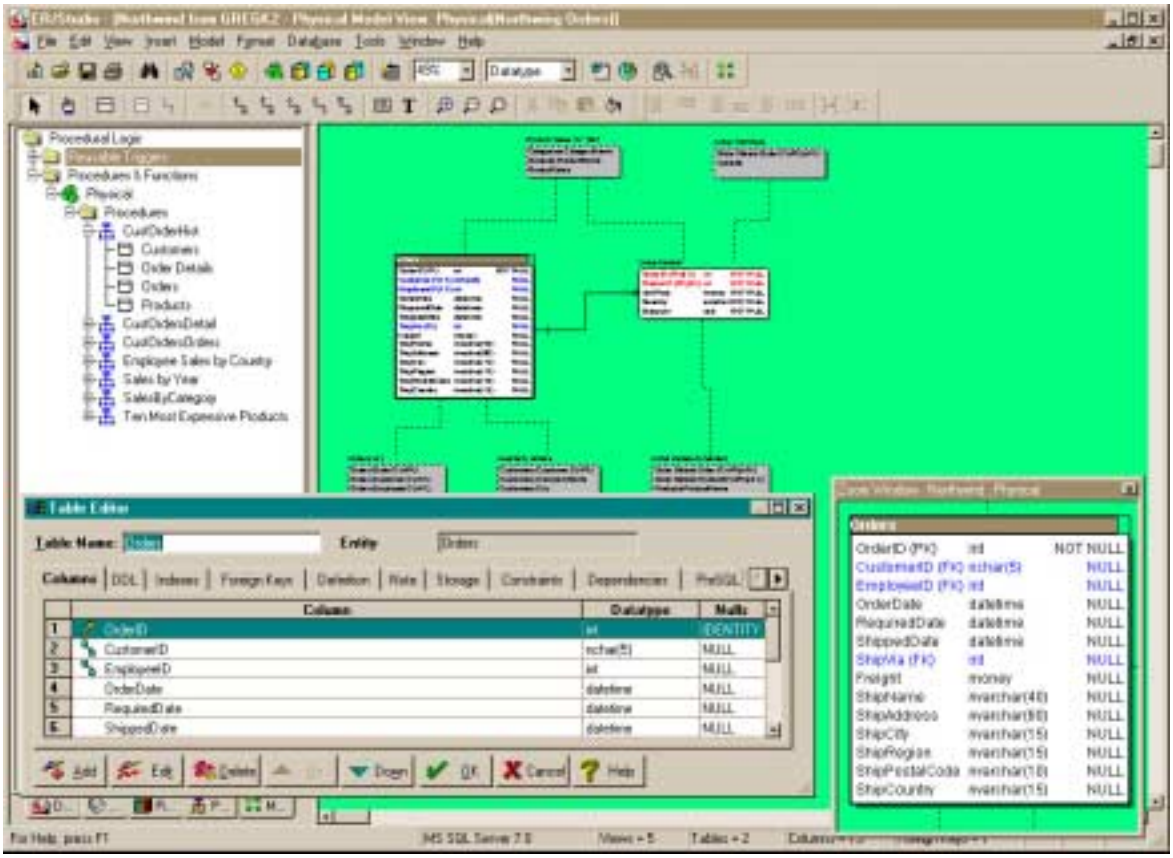


Figure 3.6: The user interface of *ER/Studio*[2]

### 3.2.2 Knowledge-based CASE tools for DB design

Database design is a complex and time-consuming process. It is often described as an art rather than a science. Traditionally, the task is carried out by a database design expert who obtains information about the user's requirements through interviewing, examining existing documents, systems and other such manuals. The whole process is a highly labour-intensive process, which has a number of weaknesses, the main ones being that expert database designers are scarce, and they are usually unfamiliar with the problem domain. Database experts have to learn about the domain from the end-users who may themselves have difficulty in expressing their information needs [Storey & Goldstein, 1993]. Another drawback to this approach is that the quality of the database

[2] http://www.embarcadero.com/products/design/erdatasheet.htm

design is highly dependent upon the capabilities and skills of the database designer [Noah & Lloyd-Williams, 1995].

Researchers have been working on applying techniques from artificial intelligence to provide effective automated assistance for this task to alleviate these problems. Their initial goal was to build fully automated knowledge based systems that produced a physical design of the required database by extracting the requirements from the end users. This aspiring initial approach soon proved to be an overly ambitious challenge. Firstly, natural language processing, even though it has made great advances, is not yet powerful enough to support natural language dialogue between the system and the end user. As a result, the built tools that simulate natural language dialogue tend to be restrictive in their applicability. Secondly, it is obvious that a tool can never replace an experienced database designer. There will always be a situation that requires modelling which is outside the domain of the system. The goal, therefore, was redefined to the much more realistic goal of building intelligent tools that provide assistance to database design experts.

Systems that offer automated assistance to database designers are referred to as *knowledge-based computer aided systems engineering* tools. These KB-CASE tools attempt to assist the designer by providing advice, suggesting alternative solutions and helping investigate the consequences of the design decisions [Noah & Lloyd-Williams, 1995]. They also contain the domain knowledge in DB design to explain design decisions.

The remainder of this section introduces three examples of knowledge based CASE tools: *Modeller*, *Expert Database Design System* (*EDDS*) and *Intelligent Interview System* ($I^2S$). The KB-CASE tools *Modeller*, and *EDDS* are designed as expert systems for database designers. They assume that the user is knowledgeable in database modelling concepts. Conversely, $I^2S$ is designed for end users of the database and does not expect them to possess any knowledge of database modelling concepts.

**Modeller**

*Modeller* is an expert system that produces a conceptual model of a database that satisfies the requirements from a user's description of an application [Tauzovic, 1989]. The conceptual model of the database is modelled as an ER schema. Once the conceptual model is complete, the system's logical design module converts the ER schema to a relational model, which represents the database as a collection of tables. Finally, the relational model is converted to a physical design by the physical modeller.

The input description of an application is given to *Modeller* in an English-like language. The user interactively inputs statements into the system, which translates them into corresponding

claims about entities, relationships and attributes. A user's statements may either add new conclusions to the evolving conceptual model or support the existing conclusions. During the interaction, the user can ask the system to take a broad look at the whole conceptual model, refine it and point out semantic problems. The system performs a number of pre-defined checks to detect errors. An example of a check performed by the system is that it verifies that each entity is connected to at least one other entity.

The knowledge base of *Modeller*, implemented using production rules, is organised into two taxonomies [Storey & Goldstein, 1993; Storey, et al., 1995]. One classifies the terms relevant to the ER model and the other stores facts about the problem being solved. The taxonomy that deals with ER modelling constructs has entities, relationships, attributes' domains and constraints at its top level. Each of the top-level elements is further classified into details of the element. The taxonomy that deals with problem facts includes model facts that are components of the conceptual model being developed and environmental facts that are the states of the modelling process. This characterisation is used in detecting possible design errors.

Although there are obvious advantages of extracting requirements directly from the end users, *Modeller* is designed to be used only by database professionals, and is intended to act as a consultant to database designers. The KB-CASE tool plays a relatively passive role, allowing the user to develop the ideas, and only provides assistance when requested.

**Expert database design system based on forms**

The *Expert Database Design System* is based on the concept that one can generate a conceptual schema for an organisational database by analysing forms that are available in the organisation [Choobineh, et al., 1988]. A form is a structured collection of variables used for data entry and retrieval. Initially, during the database design process using *EDDS*, a form is chosen and analysed. This form becomes the basis for constructing the ER schema, which is improved by analysing other forms. The process is iterated until all available forms of the application domain have been analysed. While analysing the forms, the system identifies the entities, relationships and attributes that make up the forms by initiating a dialogue with the user. The system also makes use of the dialog to determine the cardinalities of the entities participating in each relationship.

Similar to *Modeller*, the knowledge base of *EDDS* is implemented using production rules. The knowledge base is divided into four sections: form abstraction, design database, design status and data design knowledge base. The form abstraction section contains the representation of forms and form field flows. The design database section contains the evolving Entity Relationship schema.

The design status records the current status and the past design decisions. The data design knowledge base contains the database design rules.

The concept of only examining forms to understand the dataflow of an organisation is seriously flawed. The forms are bound to contain redundant information. There may also be forms, although initially designed for one purpose, that are used for a different purpose. Furthermore, if the traditional forms were used as the basis for creating a database for a new application, that new application would behave in an identical fashion to the traditional method. Examining the forms alone can discourage a designer from identifying new and efficient ways of performing a task. Another limitation of *EDDS* is that it is limited in its applicability since it relies on the existence of a comprehensive set of forms that are used to record data.

**Intelligent Interview System**

The *Intelligent Interview System* extracts information on requirements by the process of interviewing the end users of the database [Kawaguchi, et al., 1986]. The interviewing process is carried out using natural language dialogue between the system and the user. The system is designed to learn about different application domains and about the interviewing process during the interviews. In other words, the system has the objective of storing and building upon the information it extracted from each session.

Since $I^2S$ is a KB-CASE tool developed for end users of a database, it assumes that its users are not familiar with database concepts. The conceptual model of the database constructed by $I^2S$ is called a plan structure. The plan structure reflects an ER schema and consists of a set of plans, each of which represents an activity corresponding to a relationship. Initially the system requires the user to select a domain from its list of domains. The system then uses its knowledge of the domain to interview the user to extract information about the relationships of the database. The system only deals with relationships and cannot handle attributes of entities.

The system also uses its domain knowledge to infer what further knowledge might be needed from the user. In order to accomplish that, the system uses an ‘attention list’ consisting of things to ‘discuss’ with the user. This list is intended to prompt the user to express information requirements that he/she may have initially forgotten to specify. Finally, once the ER schema is complete, the relational schema is produced from it.

The knowledge base of $I^2S$ is represented using frames and production rules. In contrast to most other KB-CASE tools, $I^2S$ uses frames for planning. Attempting to apply planning concepts to the database design process is also a speciality of $I^2S$. Other knowledge of the system, such as

knowledge for converting the conceptual model to a logical model, is represented using production rules. These rules of the knowledge base have been extracted from experienced database designers.

The system's knowledge base is categorised into two parts: *domain-specific* and *domain-independent*. The *domain-specific* knowledge consists of facts about relationships one would expect to be found in a particular application. The *domain-independent* knowledge consists of knowledge about the interviewing process, planning and conceptual design. Questioning strategies that the system uses to extract information from the end users, similar to those a human database designer would use when interviewing an end user, are included in the knowledge about interviewing. The planning knowledge includes knowledge about how to search for unknown relationships that may be required. The knowledge about database design involves knowing how to translate the conceptual schema into a logical schema.

Although $I^2S$ possesses a number of interesting features, it cannot be used as a generalised database design tool as it only deals with relationships. The system relies heavily on its dictionary of verbs to recognise relationships. It is not able to recognise relationships that are expressed as nouns, such as 'marriage'. The system is restricted by the fact that it does not deal with attributes of entities. Only being able to model the domains about which the system is knowledgeable further restricts it and reduces its applicability.

$I^2S$ is specifically intended to interact with end users to obtain input to the requirements specification process. It employs a form of natural language dialogue that is considered to be appropriate for dealing with the end users. Although this is seen as an advantage for inexperienced users, the type of statements required as input for $I^2S$ may not be very natural to many users.

## 3.3 Computer-based teaching systems for DB modelling

Database design is a complex task that requires expertise. As with many areas of systems analysis and design, DB design can only be learned through extensive practice. Traditionally, database modelling was taught in classrooms where solutions to typical problems were explained. Computer-based teaching systems developed to assist students learning database modelling are exciting prospects to help students in their learning process. Since these tools are automated, students are given the opportunity of practising DB modelling at their own pace.

These software tutors can be categorised into non-intelligent tutors and intelligent tutors depending on their characteristics. Intelligent tutors tend to individualise their hints or instructions towards each student, whereas non-intelligent tutors tend to have a fixed path through the

instruction space that is traversed by all their students. The following sections investigate the two groups of tutors in further detail by presenting examples of systems that belong to each category.

### 3.3.1 Non-intelligent tutors

Software tutors developed as tools for students learning DB modelling, which do not attempt to adapt to the user's knowledge, are treated as non-intelligent tutors. There have been a few research attempts at developing educational tools to support traditional classroom teaching. This section outlines three examples of such systems: DBTool [Lim & Hunter, 1992], Concept Tutor and Didactic Tutor [Ahrens & Sankar, 1993].

**DBTool**

DBTool [Lim & Hunter, 1992] is a database design tool developed as an educational tool for teaching database design. The tool allows users to create entities, relationships and their attributes. It is capable of generating the corresponding relational schema from the constructed ER schema. DBTool contains a help function that allows the user to receive context sensitive help based on keywords, topics or subtopics that relate to creating ER diagrams using the tool.

The tool, although built as a teaching aid for ER modelling, is rather limited in its application. It only allows the creation of regular entities, regular relationships, key attributes, and simple attributes, whereas ER modelling contains a much larger set of constructs. Moreover, the tool does not support participation constraints of relationships. Although an online help system can be useful, the help function of DBTool is limited to textual descriptions, which can also be found in textbooks.

**Concept Tutor and Didactic Tutor**

Ahrens and co-workers have developed two software tutors named, the Concept Tutor and the Didactic Tutor, with the goal of familiarising novices with database design methods [Ahrens & Sankar, 1993]. The Concept Tutor is designed as a tutorial with textual explanations of the topics in database design. The tutorial contains concrete examples with solutions for each topic to enable the student to comprehend the topics more easily. The tutorial uses if-then rules to present decision rules followed in database modelling and shows a series of questions for each example that reason about these rules. The Didactic Tutor incorporates facilitators, such as allowing the student to participate in the decision rule reasoning process, to speed up the student learning process. The system involves the student in the decision rules reasoning process by asking multiple-choice

questions. The system then gives immediate positive or negative feedback. The tutor also questions the student during their interaction to clarify or reinforce instructions.

Although both the tutors are computer-based, they are static tutorials and do not possess the ability to adapt to the student. The content presented in each topic and the order of topics is fixed for all students. Both the tutors are extremely passive, having minimal interaction with the student. The Concept Tutor does not require the student to be involved in any interaction with the system besides moving on to the next topic. The Didactic Tutor attempts to involve the student in the reasoning process by posing multiple-choice questions and provides immediate feedback. Since both the systems require very little action from the student, both systems are unlikely to motivate the students to interact with the system in the long term.

### 3.3.2 Intelligent tutoring systems for ER modelling

Intelligent tutoring 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. Although intelligent tutoring systems have the potential to enhance student learning, there have been very few research attempts at developing such teaching systems for DB modelling. This section outlines ERM-VLE [Hall & Gordon, 1998a; Hall & Gordon, 1998c] and COLER [Constantino-Gonzalez & Suthers, 2000; Constantino-Gonzalez, et al., 2001], two computer based teaching systems developed for database modelling. Both systems concentrate on teaching ER modelling.

**ERM-VLE**

ERM-VLE [Gordon & Hall, 1998; Hall & Gordon, 1998a; Hall & Gordon, 1998b; Hall & Gordon, 1998c], developed by Hall and Gordon, is a text-based virtual learning environment for ER modelling. In text-based virtual reality environments, users have synchronous communications with one another and with the virtual world exclusively through the medium of text. The virtual environment of ERM-VLE is modelled as a set of interconnected rooms, which contain various objects such as an entity, attribute etc. The students interact with the environment with a restricted set of commands relating to communication, movement and object manipulation.

The objective of the learner 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. It is organised to reflect the task structure of ER model creation 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, as illustrated in Figure 3.7, has a pane named the 'current ERM' which 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 only interacts with the virtual world by issuing textual commands. The 'ERM world' pane contains a record of past interactions between the student and the world. The scenario, which the learner is dealing with, is represented in the 'Scenario' pane. The scenario is also built up dynamically depending on the student's progress, similar to the representation of the ER model.

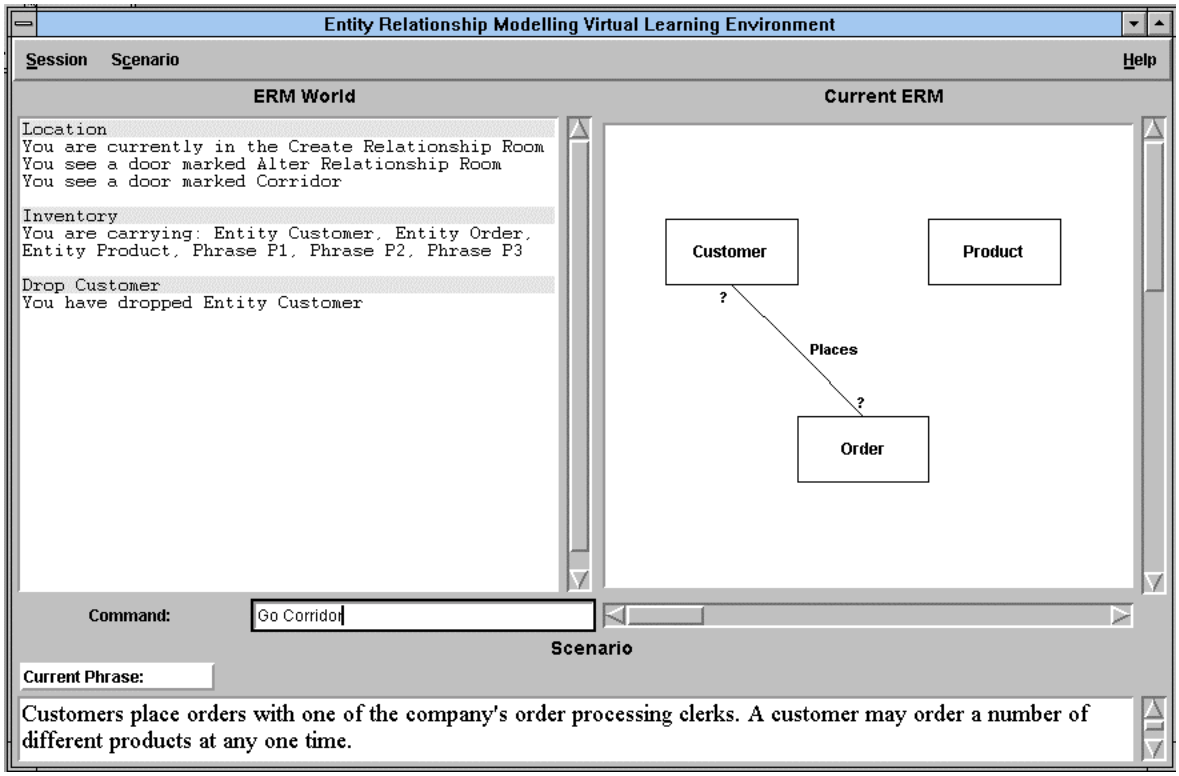



Figure 3.7: The user interface of ERM-VLE[3]

The solution for the scenario in the question is embedded in the virtual world. Correspondences between the phrases of the scenario and constructs of the ER model are stored in the solution. The

[3] http://puma.unn.ac.uk/~cmlh1/cadence/papers/ermvle.gif

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, 1998a]. 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.

The learner is quite restricted by the virtual environment 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 explanation about their mistakes. Moreover, the textual 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**

COLER [Constantino-Gonzalez & Suthers, 1999; Constantino-Gonzalez & Suthers, 2000; Constantino-Gonzalez, et al., 2001] is a web-based collaborative learning environment for Entity Relationship 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.

The interface of COLER 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 share workspace is activated. After each change in the shared workspace, the students are required to express their opinions by voting.

Since COLER's effectiveness is yet to be empirically evaluated, no conclusions can be drawn about its effectiveness as a teaching tool. Although the system encourages and supervises collaboration, it does not perform any evaluations of the ER schemas themselves to provide any 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 author's experience in tutoring it is very common for groups 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.

## 3.4 Discussion

Although CASE tools for DB modelling are effective tools for experts in the field, they offer minimal assistance to novices. Even though these tools may perform limited syntax checks on the conceptual model, they are unable to perform any semantic checks. Novices who use these systems can produce a perfectly legal DB model that is accepted by a CASE tool, which does not adhere to the requirements of the application domain. Even though there are effective CASE tools designed for DB experts, since they are sparse, the need for discovering better methods for teaching DB modelling is intensified.

Knowledge-based CASE tools, unlike traditional CASE tools, have been developed specifically for novices in ER modelling. Even though the initial goal of developing these systems was to construct systems that allowed novices to construct databases by simply specifying their requirements, the goal is yet to be completely realised. Moreover, KB-CASE tools are yet to be

capable, and may never be capable, of completely replacing a human DB designer, as they cannot fully understand and reason about the application domain. Consequently the need for human DB design experts continues to increase with the increasing need for databases to store data. These needs have given rise to the exploration of more effective methods and tools for teaching DB modelling.

Software tutors for teaching DB modelling can be used as teaching aids for novices in DB modelling. A major advantage of these systems, unlike classroom teaching, is that they allow students to learn at their own pace. The non-intelligent software tools, even though easier to develop in comparison to intelligent tutors, are limited in their effectiveness, since they are designed for a stereotypical student and do not possess the ability to individualise themselves to each student. In other words all the students using such a system would be offered the identical set of instructions.

DBTool [Lim & Hunter, 1992], the non-intelligent educational tool for teaching database design, has a number of shortcomings. The tool is very limited in its application, since it only deals with a limited set of constructs of ER modelling. The tool only provides very limited assistance to the student in the form of a help-function that allows students to perform context sensitive help based on keywords. Since, the tool is not capable of providing any semantic help regarding the application domain, students would also require a human tutor to effectively learn DB modelling concepts from the system.

The Concept Tutor and the Didactic Tutor [Ahrens & Sankar, 1993] were developed with the goal of familiarising novices with DB design. The Concept Tutor can be treated as a ‘computerised’ textbook since it only contains textual descriptions of each topic in DB design. This tutorial can be treated as a source of reference since it contains descriptions on each topic. The Didactic Tutor is a slightly modified version of the Concept Tutor that incorporates facilitators such as allowing the student to participate in the decision rule reasoning process to enhance the learning process. Both the tutors attempt to transfer knowledge of ER modelling, making the student read and understand the textual descriptions. Since ER modelling, like other areas of analysis and design, can only be mastered through extensive practice, these tutors would be very limited in their effectiveness in teaching ER modelling.

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. Student 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, COLOR, is yet to undergo a comprehensive evaluation to test its effectiveness. COLOR encourages and supervisors 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 COLOR to be an effective teaching system, a human expert must also participate in the process of collaboratively modelling an ER model.

# Chapter 4

# Design and Implementation

Empirical evaluations [Corbett, et al.; Koedinger, et al., 1997; Mitrovic & Ohlsson, 1999] have demonstrated that Intelligent Teaching Systems (ITS) developed for most domains are very effective. However, the very few previous research attempts at developing an ITS for database modelling have not been entirely successful. We have developed KERMIT, the Knowledge-based Entity Relationship Modelling Intelligent Tutor, an ITS for students learning database modelling. The system was developed with the goal of customising its pedagogical actions to each student depending on factors such as their learning ability, knowledge, etc. KERMIT offers problems close to real life and allows students to model their databases using the popular high-level conceptual database modelling technique, ER modelling.

This chapter discusses the design and implementation details of KERMIT. Section 4.1 gives an overview of the teaching system, including a brief outline of the domain. KERMIT's architecture is outlined in Section 4.2, including a high-level overview. Section 4.3 provides a detailed account of the teaching system's user interface. Details of problems and solutions, including their internal representations within the system, are given in Section 4.4. The knowledge base of KERMIT is discussed in Section 4.5. Descriptions of the student modeller and the pedagogical module are provided in Sections 4.6 and 4.7 respectively. Finally, the authoring tool developed for adding new problems to KERMIT is introduced.

## 4.1 Overview

KERMIT is an intelligent teaching system developed to assist students learning ER modelling. It is designed as a problem-solving environment, in which students are required to construct ER schemas that satisfy a given set of requirements. KERMIT assists students during problem solving

and guides them towards the correct solution by providing feedback. The feedback is tailored towards each student depending on his/her knowledge.

ER modelling, like other design tasks, requires extensive practice in order to excel in it. Therefore KERMIT is designed as a practice environment where students are given typical database design problems to solve with the assistance of the system. Since KERMIT is designed as a complement to classroom teaching and not as a substitute, when providing assistance, it assumes that the students are already familiar with the fundamentals of database theory.

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. 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 that illustrates their solution. The ER model is constructed using the workspace integrated into KERMIT's interface. Once the student completes their model or requires guidance from the system, their solution can be submitted for evaluation 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 hint 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 an interaction 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, 1994]. The following sections discuss KERMIT's design and implementation details.

## 4.2 Architecture

The main components of KERMIT are its user interface, pedagogical module and student module (illustrated in Figure 4.1). 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.

In contrast to typical ITSs, 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 intractable task. One of the major obstacles that would have to be overcome is natural language processing (NLP), as the problems in the domain are presented using natural language text. NLP would have to be used to extract the requirements of the database from the problem text. However, the NLP problem is far from being solved. Even state-of-the art NLP systems would struggle to process the database requirements' descriptions. The NLP problem can be avoided by specifying the problems in a formal language that is a sub-set of natural language. However, it is hard to avoid building parts of the solution into such a problem description.

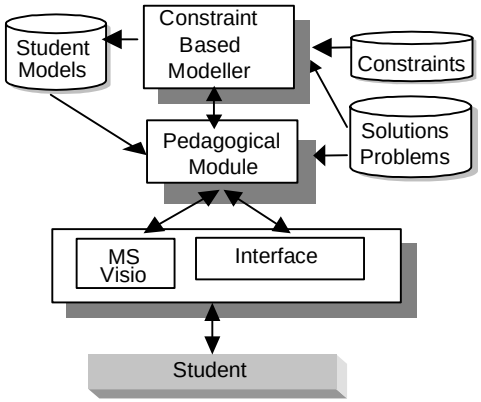



Figure 4.1: Architecture of KERMIT

Other complexities arise from the nature of the 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 extremely 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 Section 4.5 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 not only identify student solutions that are identical to the system's stored solution, but also identify correct solutions that are similar to the system's solution. In other words, the system is able to identify all correct solutions to a problem by using its knowledge base and its ideal solution.

## 4.3 User interface

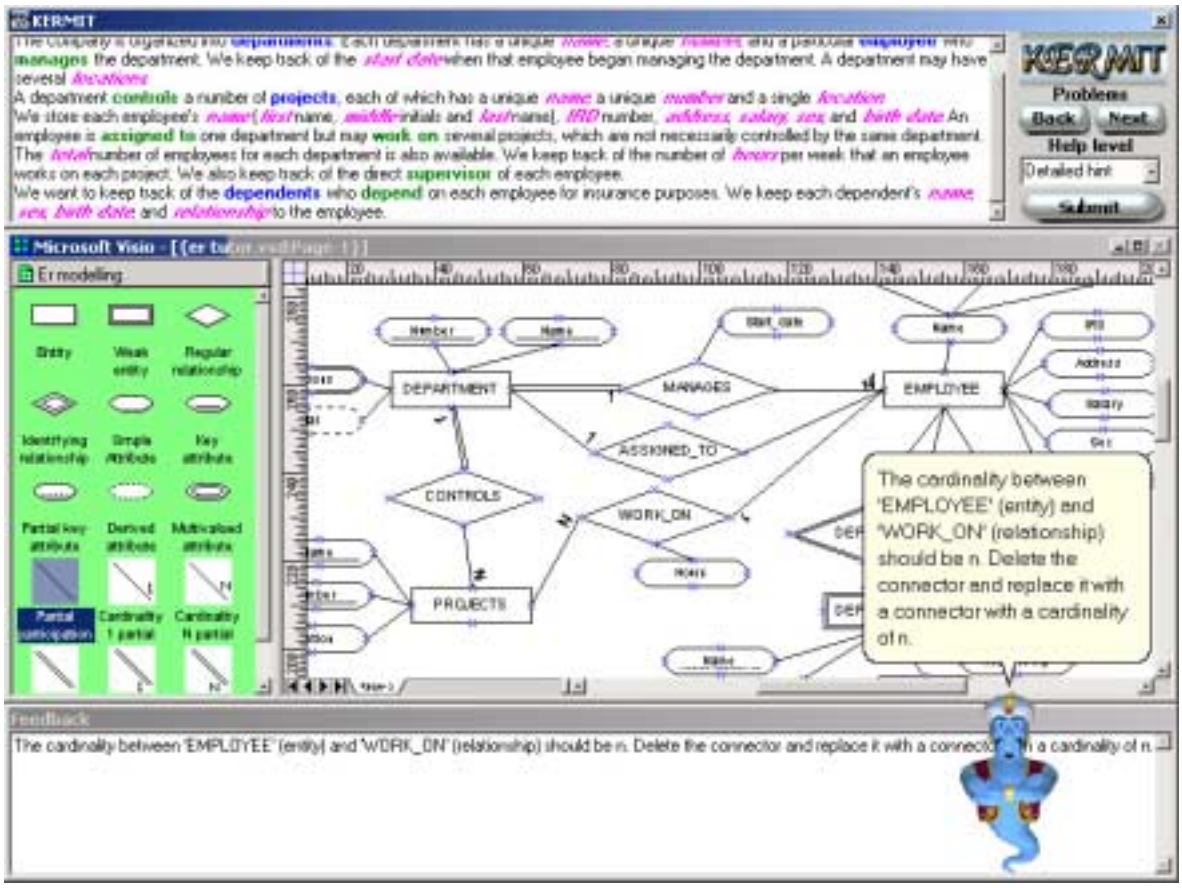



Figure 4.2: User interface of KERMIT

Each and every student using KERMIT will interact with its user interface to view a problem, construct an ER schema, and view feedback. KERMIT's interface, as illustrated in Figure 4.2, consists of four major components to fulfil these requirements. The top window displays the textual description of the current problem. The middle window is the ER modelling workspace where the student creates the 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 a button for submitting the student's solution to the system to obtain the system's feedback on the solution. 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.

### 4.3.1 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.

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 evaluated *ER/Studio* [ER/Studio] from Embarcadero Technologies and *Database Design Studio* [DDS] from Chilli Source. Since *ER/Studio* did not supporting Chen's ER modelling notation, we were forced to disregard the tool. Although *DDS* is an effective CASE tool that supports Chen's ER modelling notation, it does not offer any Application Programmer Interfaces (API) that can be used from external programs. The limitation of not being able to incorporate *DDS* with another program led to the exploration of other ER modelling tools.

*Microsoft Visio* [Visio] is a diagram composing tool that allows users to create 2D diagrams by dragging and dropping objects from a template. It offers a comprehensive set of APIs for external programs to control *Visio* and provides access to its internal representation of the diagram. Furthermore, *Visio* allows the creation of templates named stencils, a collection of objects that can be used for constructing diagrams. Due to the simplicity in customising *Visio* for ER modelling, and being able to integrate it with KERMIT, we selected *MS Visio* as the ER modelling workshop for KERMIT, creating a stencil for the required constructs (listed in Table 4.1). These constructs include entities, relationships, attributes and connectors. The student has to drag constructs from the stencil and drop them onto their virtual page to construct an ER schema.

Connectors are used to either connect attributes to entities or relationships, or to connect entities to relationships. The stencil contains six types of connectors for all possible combinations of participation (total/partial) and cardinality (1/n/no cardinality). We decided to restrict the user from inputting values to assign cardinalities to reduce errors caused by student misconceptions. This enforces a convention where cardinality can only be either '1' or 'n'. It is important that the correct type of connector is chosen to connect two constructs. For example, only a simple connector can be used to connect two attributes. Furthermore, the cardinality and participation of the entities in a relationship should be considered when choosing a connector connecting entities to a relationship.

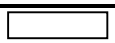
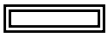
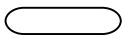
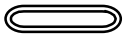
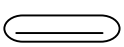
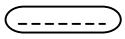
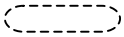
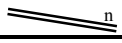
| Symbol | Construct |
|---|---|
|  | Regular entity |
|  | Weak entity |
|  | Regular relationship |
|  | Identifying relationship |
|  | Simple attribute |
|  | Multivalued attribute |
|  | Key attribute |
|  | Weak key attribute |
|  | Derived attribute |
|  | Simple connector |
|  | Total participation connector |
|  | Partial participation connector with cardinality 1 |
|  | Partial participation connector with cardinality n |
|  | Total participation connector with cardinality 1 |
|  | Total participation connector with cardinality n |

Table 4.1: Constructs of the ER modelling workspace

The end points of a connector can only be connected to entities, relationships or attributes at their connection points. Each entity, relationship and attribute has four connection points denoted by crosses, as illustrated in Figure 4.3. When an end point of a connector comes within a range of 20 pixels of a connection point of a construct, it automatically attaches (glues) itself to the construct. This feature is extremely useful in constructing ER models as it reduces the difficulties

of forming valid connections between constructs. This becomes a tedious task without this feature, since then the end points of the connectors have to be precisely joined to the connection points manually.

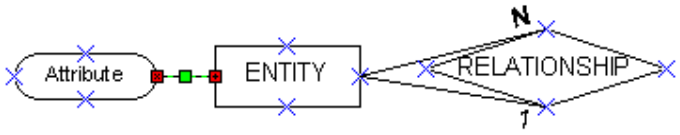



Figure 4.3: Connecting an attribute to an entity

### 4.3.2 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 displayed in Figure 4.2 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 model. We have also found that some students even highlight the words or phrases that correspond to entities, relationships and attributes using three different colours. This practice is very useful in forcing themselves to closely follow the problem text, which is essential to producing a complete solution.

KERMIT's interface is designed to simulate this behaviour. The student has to highlight a word or phrase of the problem text that corresponds to each new construct as they add them to their diagram. The highlighted words are coloured depending on the type of construct. When the student highlights a word or phrase after adding an entity, the highlighted text turns bold and blue. Similarly the highlighted text turns green for relationships and pink for attributes. (Illustrated in Figure 4.2.)

This feature is extremely useful from the point of view of the student modeller for evaluating solutions. Since there is no standard that is enforced in naming entities, relationships or attributes, the student has the freedom to use any synonym or similar word/phrase as the name of a particular construct. Since the names of the constructs 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 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

construct in the ER diagram. The system uses a one-to-one mapping of words of the problem text to the constructs of its ideal solution to identify the corresponding SS constructs.

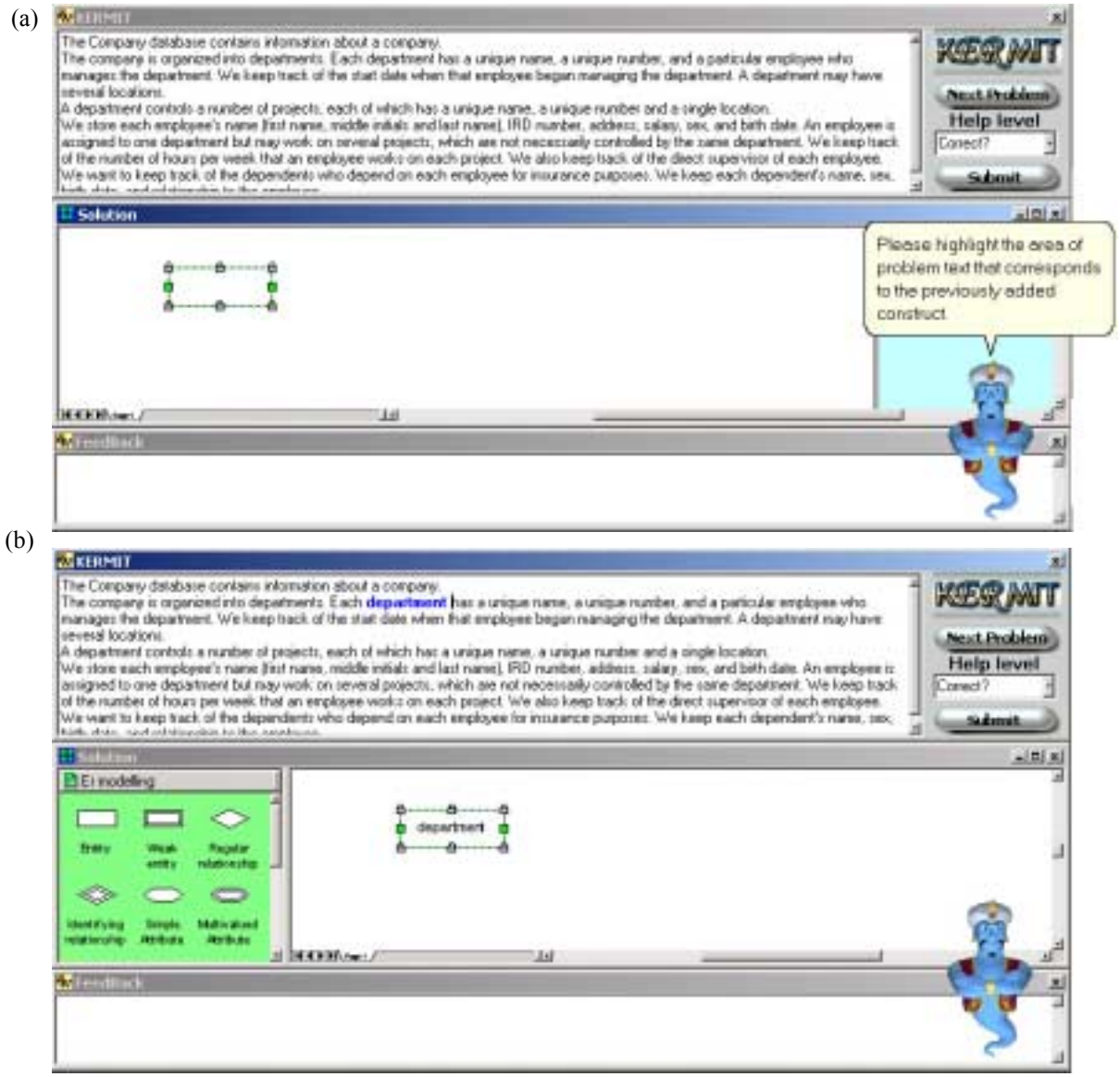



Figure 4.4: Sample scenario: the student adding the '*department*' entity

For example, consider the scenario shown in Figure 4.4. The student has added an entity to a diagram to model the department entity. After adding a construct to the workspace, KERMIT prevents the student from adding further constructs to their solution, by hiding the template. The Genie then asks them to highlight the word or phrase that corresponds to the newly added entity (illustrated in Figure 4.4a). When the student highlights a word from the problem text, the highlighted section turns blue since the added construct is an entity, and the entity is named as the

highlighted text (i.e. ‘department’). Once the student highlights a word of the problem text, the ER modelling template is re-opened to allow modelling of the database to continue (depicted in Figure 4.4b). The student may change the automatically assigned name of a construct by double clicking on the construct and typing in a desired name.

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 is forced to 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 construct to the diagram. Moreover, the student can make use of the colour coding to ascertain the number of requirements that they have already modelled by their diagram.

### 4.3.3 Feedback presentation

Once a student submits a solution, KERMIT evaluates it against its knowledge base and the ideal solution. The pedagogical module uses the results of the evaluation and composes the appropriate feedback message to be presented to the student. Since the goal of the feedback generated from the system is to improve the student’s knowledge of the domain, it is essential that these messages are presented in an effective manner. System feedback is presented in two ways: using an animated agent and using a conventional text box.

Animated pedagogical agents are animated characters that support student learning. The interface of KERMIT is equipped with such an agent (a 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 into a program 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.

Figure 4.5 illustrates the agent explaining the functionality of the next problem button and submit button of the interface.

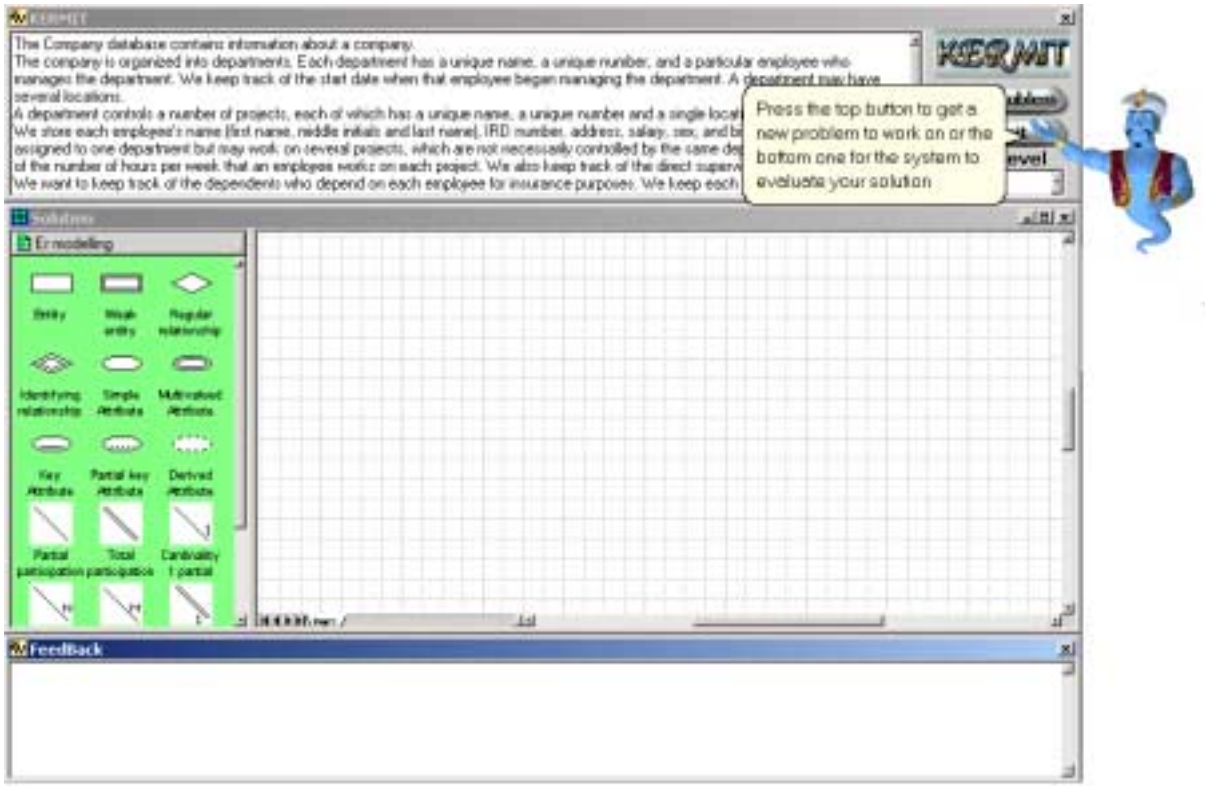



Figure 4.5: Genie explaining the functionality of the interface

The teaching system's interface also contains a static text box that displays the feedback messages, since the student may find it useful to be able to refer to the feedback while constructing their ER model (the bottom window of Figure 4.5). Although the animated agent presents the feedback, that feedback text disappears once the agent completes its speech. However, the text in the feedback textbox is displayed until the student re-submits the solution. Furthermore, in situations where more than one error has been identified, the students need to refer back to the feedback until they have corrected all their errors. From a pedagogical point of view, this also reduces the mental load on the students, since they do not have to remember the feedback that was presented by the agent.

## 4.4 Problems and Solutions

KERMIT's database of problems consists of typical database problems given in classrooms. Each problem contains a natural language description of requirements of a particular case study. The system also contains a database of ideal solutions for all problems, which are specified by a human database expert. These ideal solutions are ER schemas that fulfil all the problem requirements.

Both the problems and the ideal solutions have their own internal representations within the system. These are discussed in the following subsections.

### 4.4.1 Internal representation of solutions

In order to evaluate the student's solution efficiently, it is essential for KERMIT to have access to some internal representation of it. Since KERMIT has incorporated *MS Visio* as its ER modelling workspace, it either must use *MS Visio*'s internal representation of the diagram (SS) or maintain its own representation of the schema (SS) to analyse the solution. Since the diagrams in *MS Visio* are internally represented using a generic object to represent each construct in the diagram. It does not distinguish between the different types of ER modelling constructs. All the objects that are used to represent a *Visio* diagram are arranged in a list, in the order in which they were added to the workspace.

Accessing *Visio*'s internal representation for evaluating an SS against the constraint base is inefficient since constraints naturally deal with a particular group of constructs 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 of KERMIT when it was asked to evaluate a solution. Moreover, accessing *Visio*'s internal data structures through its APIs using a VB program is more time consuming in comparison to accessing a data structure maintained by the VB program itself. Due to these drawbacks, KERMIT dynamically maintains its own internal representation of the constructs in the workspace during runtime.

KERMIT's internal representation of the ER schema is organised by grouping similar constructs using lists. 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.

The procedure of building the internal representation of the student's solution is based on the student's interactions. When the student adds a construct to the workspace, a new object is instantiated to represent the new construct. After initialising the object, it is appended to the relevant list of objects. For example, when an entity construct is added to the workspace, a newly instantiated entity object would be appended to the list of entities. All the syntactically illegal constructs are recorded in a separate list. For example, when an attribute is added to the workspace it would 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 a database of ideal solutions for each of the problems in its database. These ideal solutions (IS) are stored in the solution database 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 memory. The IS 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 *Company* database problem. The correct ER schema for the *Company* database is depicted in Figure 4.6, and its textual representation is outlined in Figure 4.7. The solution contains four entities, namely EMPLOYEE, DEPARTMENT, PROJECTS and DEPENDENT. The EMPLOYEE entity has six attributes: *Name*, *IRD*, *Bdate*, *Sex*, *Salary* and *Address*. *Name* is a composite attribute, with *Fname*, *Mint* and *Lname* as its components.

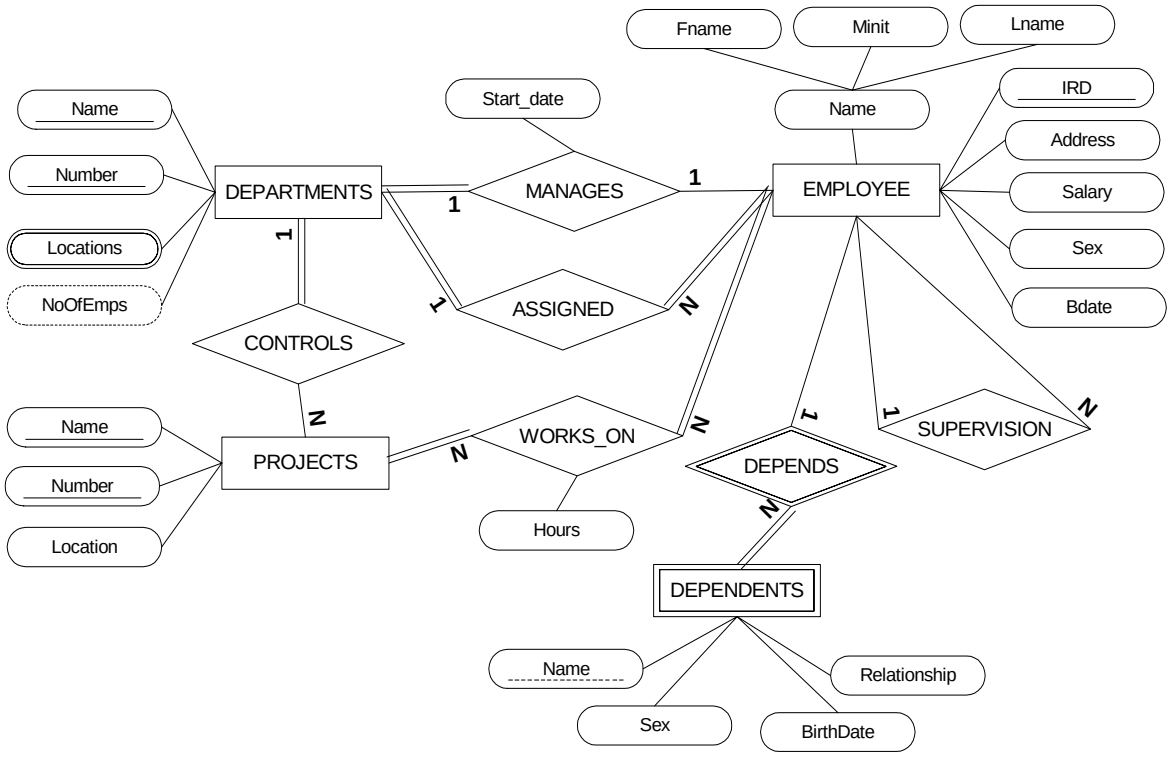



Figure 4.6: ER schema of the ideal solution for the *Company* database

```
Entities = EMPLOYEE<E3>(Name<S1>|Fname<S11>,Minit<S12>,Lname<S13>|,IRD<K1>,
    Bdate<S2>,Sex<S3>,Salary<S4>,Address<S5>),
    DEPARTMENT<E1>(Name<K1>,Locations<M1>,Number<K2>,NoofEmps<D1>),
    PROJECT<E2>(Location<S1>,Name<K2>,Number<K1>),
    DEPENDENT<W1>(Name<P1>,BirthDate<S2>,Relationship<S3>,Sex<S1>)
Relationships  = SUPERVSION<R5>()-<E3>1p,<E3>np-,
    MANAGES<R1>(StartDate<S1>)-<E3>1p,<E1>1t-,
    ASSIGNED<R2>()-<E3>nt,<E1>1t-,
    WORKS_ON<R3>(Hours<S1>)-<E3>nt,<E2>nt-,
    CONTROLS<R4>()-<E1>1p,<E2>nt-,
    DEPENDS<I1>()-<E3>1p,<W1>nt-
```

Figure 4.7: Internal representation of the ideal solution for the *Company* database

A unique identifier (id) is assigned to each construct in the ideal solution. The id is composed of a single character code that specifies the type of the construct and an integer. For example, 'E' is used for regular entities and 'W' is used for weak entities (see Table 4.2 for a comprehensive list of

codes used to represent different constructs). The integer in the id makes the id unique. Thus the first regular entity is assigned ‘E1’, the second ‘E2’, etc. From Figure 4.7, we can see that the DEPARTMENT entity was the first one that was created, as its code is ‘E1’. It has two key attributes (*Name* and *Number*), one multivalued attribute (*Location*) and a derived attribute (*NoOfEmps*).

| Construct type | Code |
|---|---|
| Regular entity | E |
| Weak entity | W |
| Regular relationship | R |
| Identifying relationship | I |
| Simple attribute | S |
| Multivalued attribute | M |
| Key attribute | K |
| Partial key attribute | P |
| Derived attribute | D |

Table 4.2: Constructs and the codes used to represent them internally

The relationships list contains the name of each relationship, its attributes and the ids of the participating entities. The attributes of the relationship are given within parentheses after the relationship name. The ids of the participating entities follow the attributes list with their cardinality (1/n) and participation (total/partial). To illustrate, consider the relationship named *Supervision*, found in the *Company* database. This relationship is a recursive relationship involving the entity ‘E3’ (Employee) and has no attributes, as illustrated in Figure 4.8. It is represented as ‘SUPERVSION<R5>()-<E3>1p,<E3>np-’. An instance of the e*mployee* entity type participates in this relationship as a *supervisor*, while a group of instances of the same entity type participate in the relationship as *supervisees*. Hence, the cardinality of the first participation of the entity, ‘E3’, is 1 (an employee has only one supervisor), and the participation is partial (not every employee is a supervisor). The cardinality of the *supervisee* participating in the relationship is n (a supervisor can supervise many employees) and the participation is partial (not all employees have a supervisor).

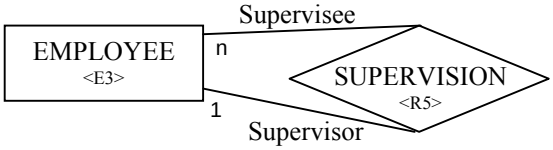



Figure 4.8: ER schema for SUPERVISION relationship

### 4.4.2 Internal representation of problems

The text of the problems offered by KERMIT describes the requirements of a database that the student reasons about and models using appropriate ER modelling constructs. Since KERMIT does not possess any NLP abilities, we were forced to develop an effective internal representation that enabled the system to identify the semantic meaning of each construct.

As discussed previously, KERMIT forces the student to highlight the word or phrase modelled by each construct in their solution (see Section 4.3.2 for details). The system uses these highlighted words to form a correspondence between the constructs of the SS and the IS. The connection is established by the assistance of a mapping maintained by the system, which maps particular words of the problem text to the IS constructs. The mapping, which is embedded in the problem text, is specified by a human expert during the phase of adding the problem into the system.

The problem text is represented internally with embedded tags that specify the mapping of its important words to the IS constructs. These tags have a many-to-one mapping to the constructs in the ideal solution. In other words, more than one word may map to a particular construct 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 constructs of the ideal solution. 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. Figure 4.9 illustrates an extract of the internal representation of the *Company* database problem.

The <**E4** Company should not be an entity type because there is only one instance of the company entity.> Company database contains information about a company. The <**E4**> company is <**R6** Organized should not be a relationship because company entity type should not be modelled.> organized into <**E1**> departments. Each <**E1**> department has a unique <**E1K1**> name, a unique <**E1K2**> number, and a particular <**E3**> employee who <**R1**> manages the department. We keep track of the <**R1S1**> start date when that <**E3**> employee began managing the department. A department may have several <**E1M1**> locations. A <**E1**> department <**R4**> controls a number of <**E2**> projects, each of which has a unique <**E2K1**> name, a unique <**E2K2**> number and a single <**E2S1**> location.

Figure 4.9: Internal representation of the *Company* database problem

As an example of the use of the embedded tags, consider the situation where the student creates the DEPARTMENT entity. The word 'department' in the problem text would be highlighted and the system would use its lookup table to identify the tag associated with the student's DEPARTMENT entity as 'E1'. This entity has two keys, 'name' and 'number' referred to as 'E1K1' and 'E1K2' respectively in the problem. When the student creates these attributes, the combination of entity and attribute code allows the system to easily identify the entity or relationship to which the attribute belongs. Similarly, the MANAGES relationship identified as 'R1' has a simple attribute, S*tart date* identified with code the 'R1S1'.

The embedded tags are also used to attach problem specific hints. They can be used to give direct, context specific advice to the student if he/she is noted as making a typical error. For example, the *Company* database, a typical error made by the students is to model *Company* as an entity. However, since there is only one instance of *Company* it should not be represented as an entity type in the database schema. Such a problem specific instruction/hint is incorporated in a compact manner by attaching the message to the particular tag. In Figure 4.9 we can see that the hint message is attached to the 'E4' tag. When the student highlights the word 'Company' from the first sentence when modelling a *Company* entity type, immediate feedback is provided that says, "Company should not be an entity type because there is only one instance of the company entity".

## 4.5 Knowledge base

The knowledge base is an integral part of any Intelligent Teaching System. The quality of the pedagogical instructions provided by an ITS depends critically on its knowledge base. The teaching system uses its knowledge base to evaluate the student's solutions and to produce instructional messages.

The domain knowledge of KERMIT is represented in a descriptive form consisting of constraints used for testing the student's solution for syntax errors and comparing it against the system's 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.

The constraint base consists of two types of constraints: syntactic and semantic. Section 4.5.1 and 4.5.2 describe these and give examples. Section 4.5.3 discusses the procedure followed to acquire knowledge.

### 4.5.1 Syntactic constraints

The syntactic constraints describe the syntactically valid ER schemas and are used to identify syntax errors in the ER model produced by the student. These constraints only deal with the student's solution and are independent of the problem. 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". (See Appendix A for a complete list of K*ER*MIT's constraints.)

```
id          = 10
relCond     = "t"
satCond     = "unique (join (SSE, SSR))"
feedBack    = "Check the names of your entities and relationships. They must be unique."
feedBack1   = "The name of <viol> is not unique. All entity and relationship names must be unique."
feedBack2   = "The names of <viol> are not unique. All entity and relationship names must be
               unique."
construct   = "entRel"
conceptID   = 1
```

Figure 4.10: Parts of Constraint 10

An example of a syntactic constraint, Constraint 10, is detailed in Figure 4.10. 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. Its satisfaction condition checks that each construct in the student's entities (`SSE`) and relationships (`SSR`) has a unique name. 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 and specific 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.

| Id | Concept description |
|---|---|
| | **Syntax and notation** |
| 0 | Errors in using notation |
| | **Entities** |
| 1 | Regular entities |
| 2 | Weak entities |
| 3 | Isolated regular entities |
| | **Relationships** |
| 4 | Regular relationships |
| 5 | Identifying relationships |
| 6 | n-ary regular relationships |
| | **Attributes** |
| 7 | Simple attributes |
| 8 | Key attributes |
| 9 | Partial key attributes |
| 10 | Derived attributes |
| 11 | Multivalued attributes |
| 12 | Composite attributes |
| 13 | Composite multivalued attributes |

Table 4.3: Concepts addressed by the constraint base

The constraint base addresses a total of fourteen concepts. The concepts, as listed in Table 4.3, include different types of constructs, different arrangements of constructs and errors using the notation. The list of concepts are used to divide the constraint base into groups in which the particular group of problems that the student has the greatest difficulty with can be focused upon during problem selection (refer to Section 4.7.2 for details on problem selection). Each constraint is assigned to only one concept. In cases where a constraint fits into more than one concept, the concept that has the strongest claim is assigned. The specific concept that the constraint deals with is specified as the *conceptID* of the constraint. The example considered, Constraint 10 (detailed in Figure 4.10), belongs to the concept with the identifier of 1, which is 'regular entities'.

Not all syntactic constraints are as simple as Constraint 10. For example, Constraint 33 (shown in Figure 4.11) has a far more complex relevance and satisfaction condition. The constraint verifies that in situations where there is a single owner for a weak entity, the owner should participate in the identifying relationship with a cardinality of one (demonstrated in Figure 4.12). The constructs that are relevant to this constraint are identifying relationships (of type *i*) from SSR (list of relationships in the SS), which have exactly one regular entity (of type e) participating in it. For the constraint to be satisfied, the cardinality of the participating regular entity should be one (i.e. 1c). In order to verify the satisfaction condition, each relevant identifying relationship (the RELVNT list contains all the constructs identified by the relevance condition) has to be examined to find the cardinalities of their regular entities.

```
id          = 33
relCond     = "each obj SSR
                    (and (= type (obj), i),
                         (= countOfType (entities (obj), e), 1))
satCond     = " each obj RELVNT
                    (each ent (ofType (entities (obj), e))
                          (= card (ent), 1c))
feedBack    = "Check the cardinalities of the identifying relationships. The cardinality of the
              owner should be 1 in identifying relationships with one owner (regular entity)."
feedBack1   = "The cardinality of the regular entity (owner) of the <viol> identifying relationship
              should be 1. The cardinality of the owner should be 1 in identifying relationships
              with one owner."
feedBack2   = "The cardinalities of the regular entities (owners) of <viol> identifying
              relationships should be 1. The cardinality of the owner should be 1 in identifying
              relationships with one owner."
construct   = "rel"
conceptID   = 5
```

Figure 4.11: All parts of Constraint 33

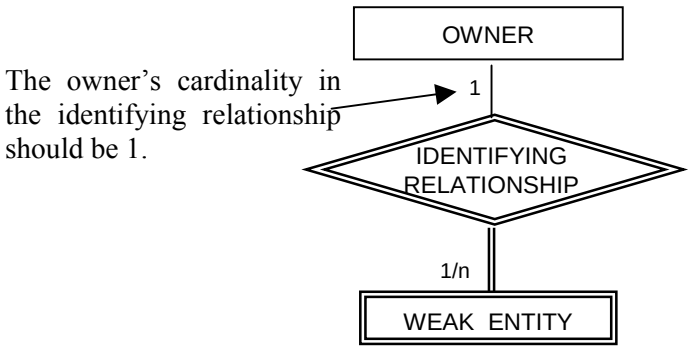



Figure 4.12: Illustration of Constraint 33

### 4.5.2 Semantic constraints

Semantic constraints operate on the relationships between the student's solution and the system's ideal solution. For example, "The student's solution should consist of all the entities that exist in the ideal solution" is a simple semantic constraint. Furthermore, "If a particular entity participates in a relationship of the ideal solution, the corresponding entity of the student's solution should also participate in the corresponding relationship" is another semantic constraint. (See Appendix A for a complete list of K*ER*MIT's constraints.)

```
id         = 37
relCond    = "each obj ISR (and
                        (and (= (count (entities (obj))) 2) (= type (obj) r))
                        (each ent (entities (obj))
                              (notNull (matchSS (ent))))"
satCond    = "each obj RELVNT
                   (notNull (matchSS (obj)))"
feedBack   = "Check whether you have all the needed binary relationships. You are missing some
              binary relationships."
feedBack1  = "Check whether you have all the needed binary relationships. You are missing a binary
              relationship. Check the complete solution to find out what is missing."
feedBack2  = "Check whether you have all the needed binary relationships. You are missing some
              binary relationships. Check the complete solution to find out which are missing."
construct  = "rel"
conceptId  = 4
```

Figure 4.13: Constraint 37

Constraint 37 (illustrated in Figure 4.13) is an example of a semantic constraint that verifies that the SS contains all the binary relationships in the IS. The constraint is only relevant for the binary relationships in the IS where the SS contains the two entities that participate in that relationship. The condition that checks the relevance of the constraint is relatively complex, containing a nested loop. It uses the `matchSS` function with an IS construct as its argument to find its corresponding construct in the SS (the constructs are matched by investigating their unique identifiers as discussed in Section 4.4.2). The existence of a matching construct in the SS is determined by using the `notNull` function, which returns true if such a construct exists. The relevance condition must also check that the relationships have exactly two entities participating in the relationship to ensure only binary relationships are considered. The detailed feedback message of this constraint is not as informative as other detailed feedback messages. It is designed to encourage students to discover the solution by themselves. Mentioning the names of missing constructs would make problem solving too simple for students, as it would reveal answers.

```
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)))))"
feedBack    = "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)."
feedBack1   = "The <viol> entity is missing some composite multivalued attributes (you can
               represent composite multivalued attributes as weak entities as well)."
feedBack2   = "<viol> entities are missing some composite multivalued attributes (you can represent
               composite multivalued attributes as weak entities as well)."
construct   = "ent"
conceptID   = 13
```

Figure 4.14: Constraint 67

Semantic constraints are usually more complex than syntactic constraints. Constraint 67, illustrated in Figure 4.14, is another example of a semantic constraint. This constraint deals with composite multivalued attributes of entities. Since they can also be modelled as weak entities, the constraint has to compare a composite multivalued attribute in the IS to a similar one in the SS or a weak entity in the SS. The constructs relevant to the constraint include IS entities, which possess a multivalued composite attribute, that have a corresponding entity in the SS. The constraint is satisfied if all the corresponding relevant 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 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.

The equivalent solutions identified by Constraint 67 are illustrated in Figure 4.15. 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.

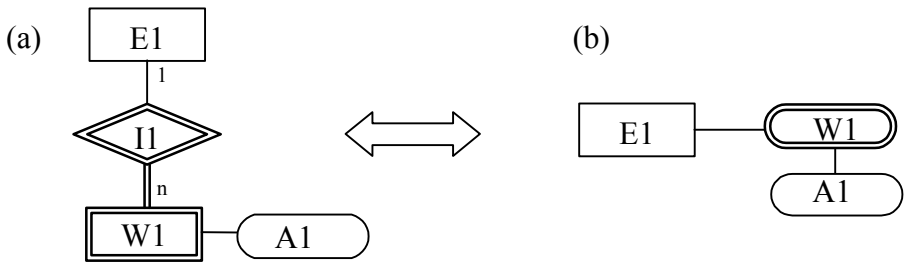



Figure 4.15: A weak entity can also be represented as a multivalued attribute

### 4.5.3 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, 1994]. Due to the nature of the domain itself, building an outline of the syntax of ER modelling was not straightforward. Since ER modelling is an ill-defined domain, descriptions of its syntax in textbooks are informal. An outline of the ER modelling syntax was produced with the assistance of an ER modelling expert. This process was conducted as an iterative exercise in which the syntax outline was refined by adding new constraints. The syntax outline was later used to formulate syntactic constraints using the constraint specification language.

Semantic constraints are harder to formulate. Initially, sample ER diagrams were analysed and compared against their problem specifications to formulate basic semantic constraints, such as a constraint to perform one-to-one comparisons of entities of a given solution and the ideal solution. Later, errors were introduced to the sample ER diagrams and the process of reverse engineering was followed to formulate constraints that identified the introduced errors.

Stage two student's answers to an exam question that required database modelling were also analysed to discover constraints. To avoid compromising the student's privacy, the relevant page of each student's answer script was photocopied. Analysing their solutions irrespective of the semantics of the problem yielded a number of syntactic constraints. Moreover, comparing the solutions against the correct solution yielded insights into a number of semantic constraints. This analysis also resulted in the discovery of a number of typical errors made by the students. The knowledge base has also been enriched by constraints that deal with these errors.

A small travel agency is about to design a database that will hold information that agency needs to conduct its business. The relevant information follows:

- The agency exclusively offers trips. Each trip is identified by a number. For a trip, the name, the price per person and the duration of the trip (in days) are stored. It is assumed that a trip is offered every week on the same day.
- A trip can be booked by a customer for a certain day. Each customer gets a booking number that uniquely identifies that customer. A customer can book different trips on different days.

Draw and ER diagram that captures the above information.

Figure 4.16: Exam question 11

The investigated exam question is outlined in Figure 4.16. It involved composing an ER schema for a travel agency. All the requirements of the database that needed to be modelled are given in the problem. The correct solution, depicted in Figure 4.17, contains two entities CUSTOMER and TRIP and a binary relationship between CUSTOMER and TRIP.

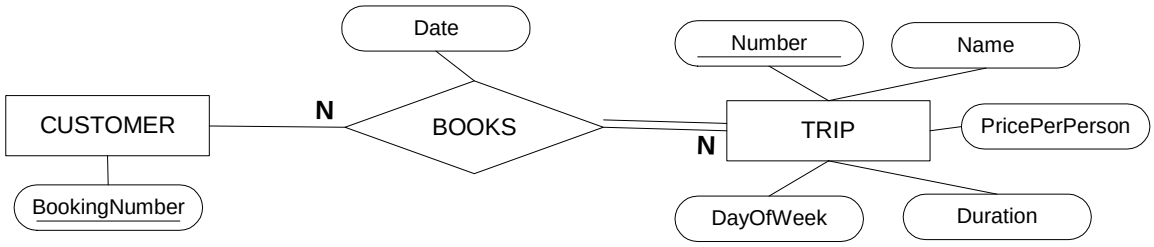



Figure 4.17: Correct solution for exam question 11

Figure 4.18 illustrates four examples of extracts from the students' answers to the exam question, which provided insights into new constraints. The student who composed Solution (a) has made the error of adding a key attribute to a relationship (i.e. *BookingNo* is a key attribute of the BOOK relationship). Moreover, the *Customer* entity in the schema does not possess a key attribute. The two new constraints added to the constraint base, after analysing this solution, were: relationships cannot have key attributes, and check that all regular entities have at least one key attribute.

Solution (b) demonstrates the student's lack of knowledge of binary relationships. Two binary relationships (OFFER_FOR and BOOKED_BY) have been used to represent the same relationship between the two entities (TRIP and CUSTOMER). The constraint that specifies that there should be exactly one binary relationship in the SS that corresponds to each binary relationship in the IS was discovered from this solution.

The student who composed Solution (c) has made the error of assigning the partial key attributes, *Number* and *Start date*, to the TRIP entity and BOOKED_BY relationship. This solution yielded the syntactic constraint that specifies that only a weak entity can have a partial key attribute.

Solution (d) displays the error of a missing attribute. The IS_BOOKED_BY relationship of the solution does not have the *Date* attribute. Investigating this solution produced a constraint that checks whether the entities and relationships of the SS contain all the corresponding attributes that are present in the IS.

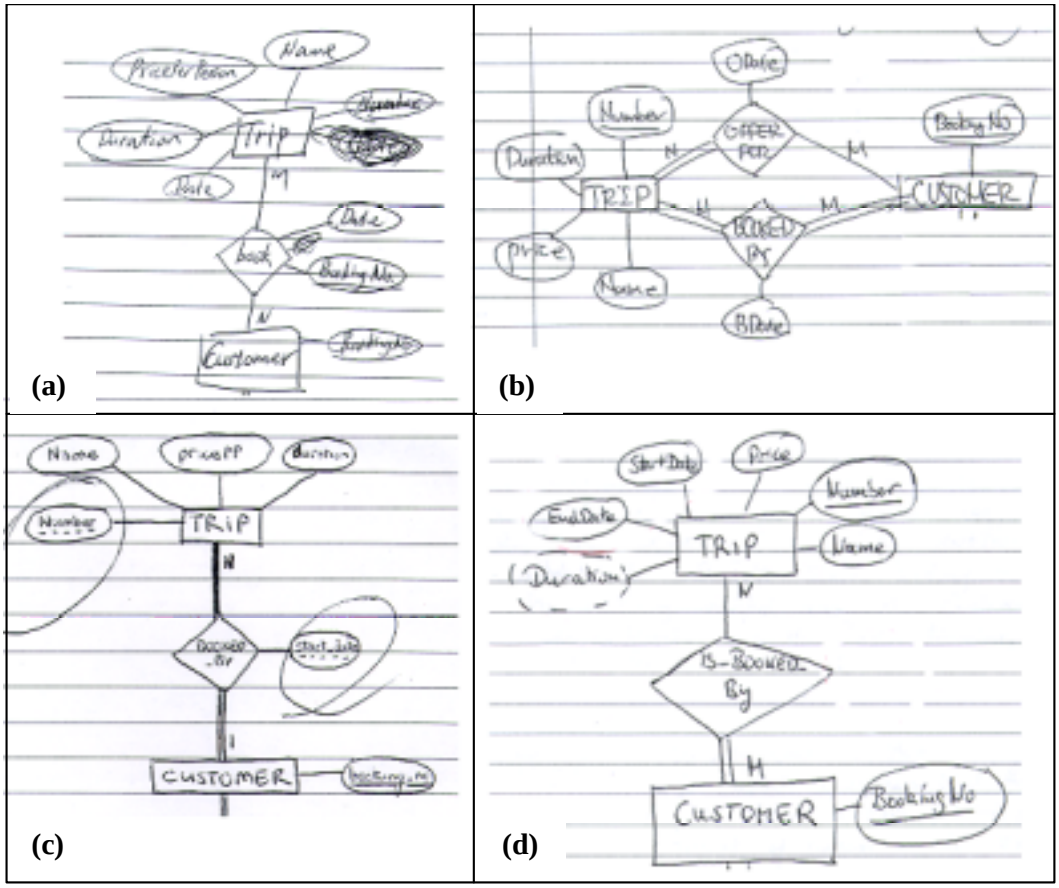



Figure 4.18: Students solutions to the exam question 11

## 4.6 Student modeller

Student modelling is the process of gathering relevant information in order to assess the current knowledge state of the student. The task of building a student model is extremely difficult and laborious due to the large search spaces involved [Self, 1990]. Although modelling the student's knowledge completely and precisely is an intractable problem, student models can be useful even if they are not complete and accurate [Ohlsson, 1994]. Human teachers use very loose models of their students, yet they are extremely effective in teaching. Similarly, even simple and constrained student models are sufficient for instructional purposes.

KERMIT uses constraint based modelling (CBM), which is a student modelling approach that reduces the complexity of the task of student modelling by only focusing on the errors. The domain knowledge in CBM is represented as a set of constraints, where each constraint defines a set of equivalent problem states. An equivalence class generates the same instructional action; hence the

states in an equivalence class are pedagogically equivalent. This is based on the assumption that there can be no correct solution to a problem that traverses a problem state that violates the fundamental concepts of the domain. Violation of a constraint signals the error which comes from incomplete and/or incorrect knowledge.

KERMIT identifies violated constraints by investigating the SS to identify syntax errors using its syntactic constraints and comparing the SS to the IS using its semantic constraints. 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. In the case where the constraint is relevant, then 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 SS.

KERMIT's student modeller includes a parser and an interpreter to evaluate the problem state against the constraints. Initially the relevance condition is checked against the problem state. In situations where the relevance condition is set as *true*, the step of verifying the relevance condition is skipped, as the constraint is always relevant. During the evaluation of the relevance condition, firstly, the condition is parsed and arranged into an abstract syntax tree; secondly, the abstract syntax tree is traversed in a depth first manner and evaluated from the bottom to the top. During the evaluation, the constructs in the SS or IS that satisfy the relevance condition are recorded in a list. The constraint is only relevant to the current problem state if there are constructs that satisfy the relevance condition. If the relevance condition is satisfied, the problem state is evaluated against the satisfaction condition. Evaluating the satisfaction condition follows the same procedure as the relevance condition, but only operates on the relevant constructs that were identified in the relevance condition. The constraint is branded as satisfied only if all the relevant constructs satisfy the satisfaction condition.

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 implanted as an overlay model. In contrast to the short-term student 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 long-term student model is saved in a file when the student

logs out. Each entry in the student model file consists of the constraint identifier, the total number of instances it was relevant and the total number of instances it was violated. The pedagogical module uses these data to select new problems for the student.

## 4.7 Pedagogical module

The pedagogical module (PM) is the driving engine of the whole teaching 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 discovers the constructs that the student has violated. Since the constraints in the constraint base are ordered following the conventional procedure of ER modelling, KERMIT targets the first violated constraint for instruction. The conventional ER modelling procedure follows modelling entities first, modelling relationships second and modelling attributes that belong to either entities or relationships last.

### 4.7.1 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 construct with errors 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 (refer to the example given in Figure 4.13). 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 they submit their solution, a simple message indicating whether or not the solution is correct is given. The system contains this basic feedback level 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 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.

### 4.7.2 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 concept (itemised in Table 4.3) 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 constructs with which they experience difficulties. In situations where there is no prominent group of constraint to be targeted, the next problem in the list of available problems, ordered according to increasing complexity, is given.

| Concept id | Problem No | | | |
|---|---|---|---|---|
| | **1** | **2** | **3** | **…** |
| **1** | 4 | 5 | 3 | |
| **2** | 0 | 1 | 1 | |
| **3** | 0 | 0 | 0 | |
| ⋮ | | | | |

Figure 4.19: Problems and concepts matrix

During the initialisation phase, KERMIT inspects all the available problems and forms a matrix that contains the numbers of the constructs of each ideal solution that belong to each concept. An extraction of the matrix is depicted in Figure 4.19. According to the example, problem one has four

constructs that belong to concept number one, i.e. regular entities. In other words the IS for problem one has four regular entities. These values in the matrix are static throughout a problem solving session, but columns in the matrix are removed as problems are completed.

## 4.8 Authoring tool

The ER diagrams are represented internally in an encoded textual representation, as discussed in Section 4.4.1. 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's problems and solutions database. 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 KERMIT 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:

1. Entering the problem text.
2. Composing the ER schema.
3. Specifying the correspondence between the words of the problem text and the constructs of 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 of KERMIT presented to students, in which they can construct the ideal solution to the problem. Once the user models 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 (illustrated in Figure 4.20). 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 to KERMIT's problem and solution database.

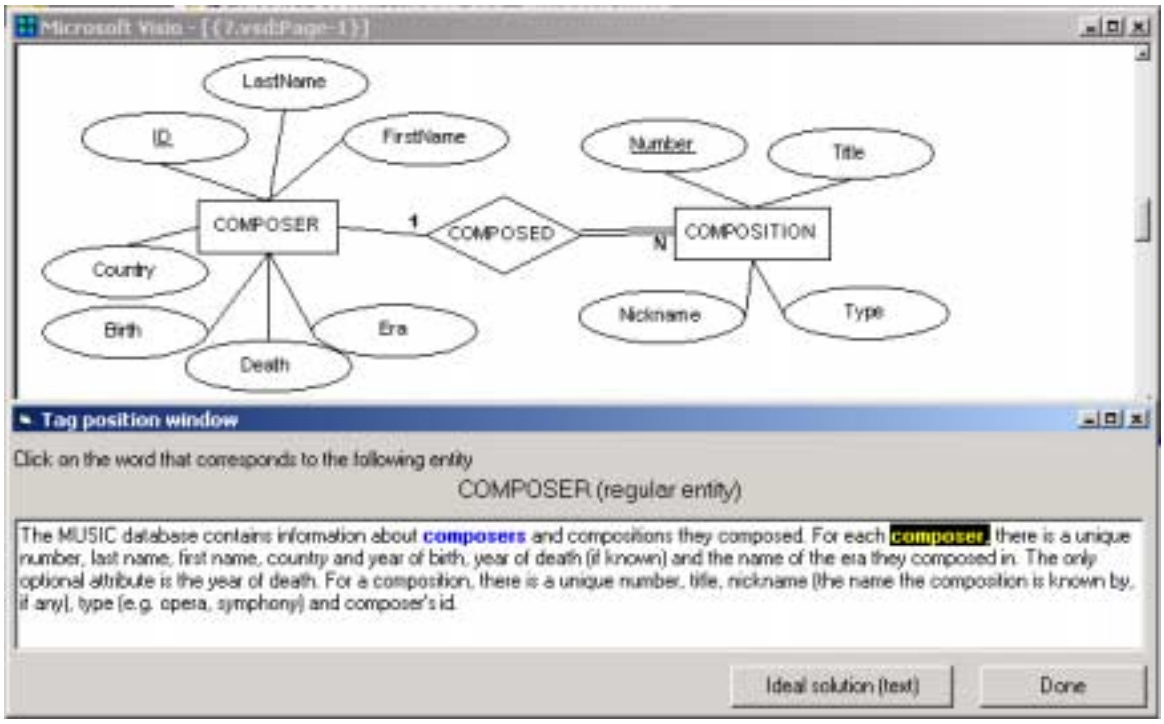



Figure 4.20: Interface of the authoring tool for adding new problems

# Chapter 5

# Evaluation

Evaluation is an integral part of the development process of Artificial Intelligence (AI) based projects. Intelligent teaching systems fit this statement perfectly, being the application of AI methods to educational environments. As the credibility of an ITS can only be gained by proving its effectiveness in a classroom environment or with typical students, we conducted a series of evaluation studies on KERMIT:

1. Preliminary evaluation
2. Classroom Study 1 (Victoria University)
3. Classroom Study 2 (University of Canterbury)

This chapter outlines the evaluation studies and their results. The following section describes the preliminary evaluation and its results. Section two details the first classroom study, which was conducted at Victoria University, and the results can be found in Section three. Sections four and five describe the second evaluation study conducted with students from the University of Canterbury and present the results. Finally the findings of all three studies are discussed.

## 5.1 Preliminary evaluation study

A preliminary evaluation study was carried out in the form of a think-aloud protocol with post-graduate students enrolled in an intelligent teaching systems course. A think-aloud protocol involves a participant and an observer, with the participant being asked to verbalise his/her thoughts while working on a particular task. Initially the observer introduces the task to the participant and begins the observation, while being videotaped. In situations where the participant falls silent, the observer encourages him/her to keep talking.

Two versions of KERMIT were made available to the students, named A and B. Both versions were identical in all ways except for the set of problems. Each version contained a set of three problems installed on a machine accessible to all students.

The think-aloud protocols were performed in pairs. Initially a member of the group played the role of the participant and interacted with version A. Once interaction with the system was completed, the roles were exchanged and the new participant interacted with version B.

Results of the study were collected from the think-aloud protocol videotapes and the student logs. The student logs were analysed to obtain the required statistical data. The videotapes of the think-aloud protocols were also analysed to experience the methodology followed by the students and to obtain an idea of their perception of the system.

### 5.1.1 Interacting with the system

A total of fifteen students interacted with KERMIT. The interaction details were summarised by analysing their logs and are presented in Table 5.1. The mean interaction time with the system was 29 minutes. The interaction times have an extremely high standard deviation of 21 minutes since some students only experimented with the system and others evaluated it thoroughly. The interactions varied from a minimum of 5 minutes to a maximum of 1 hour and 17 minutes.

| | mean | s. d. |
|---|---|---|
| Time spent on problem solving (min.) | 29:41 | 21.40 |
| No. of attempted problems | 2.20 | 0.77 |
| No. of completed problems | 1.73 | 0.80 |
| Time spent per problem (min.) | 13:30 | |

Table 5.1: Interaction details of the preliminary study

During the interaction, the mean number of problems attempted by the students was 2.2. Considering that there were only three problems available from the system, a mean value of 2.2 attempted problems is quite high. However, the high number of attempted problems can be explained by the fact that the students were obliged to complete all three problems during their think-aloud study.

The students completed 1.7 out of the attempted 2.2 problems on average. The completion rate of 79% was expected since KERMIT forced the student to complete a problem before moving on to the next problem.

### 5.1.2 Mastery of constraints

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 constraints [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 [Mitrovic & 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 subject involved in 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 5.1. To reduce individual bias, only the occasions in which at least two thirds of the total population of subjects had a relevant constraint were used. Only the first eight occasions satisfy this criterion.

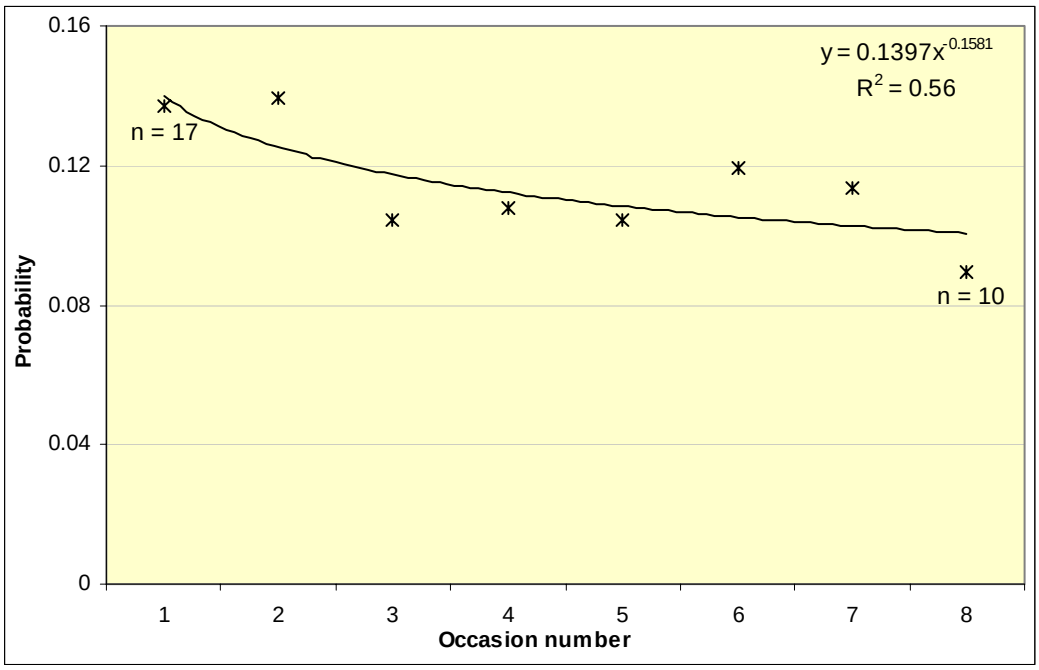



Figure 5.1: Probability of violation a constraint as a function of the number of occasions when that constraint was relevant, averaged over subjects of the preliminary evaluation

The power curve shows a weak relationship between the probability of violating a constraint and the occasion of its application with an $R^2$ power law fit of 0.56. The probability of violating a constraint has slightly decreased as its occasion number increases. This data from the initial preliminary study cannot be treated as an approximation of typical student behaviour since this study involved only three simple problems. For the system to be effective, students would have to spend more time with the system, solving more complex problems.

### 5.1.3 Discussion

Although the preliminary study was not intended as a comprehensive evaluation of KERMIT, students who used the system provided useful feedback. The video footage of the students interacting with KERMIT provided a valuable source for discovering typical student interactions. They also yielded insights into improving the usability of the system. For example, we discovered that students had initially struggled with constructing ER models using the workspace. In the light of this discovery, we added a tutorial that demonstrated how to create ER models by dragging and dropping constructs from the stencil and how words in the problem text need to be highlighted.

The think-aloud protocols also found problems in the knowledge base. In most cases the feedback messages were not very helpful since they were concentrating on minor errors. To ensure that the most important errors are discovered first, we arranged the constraints in the constraint base to follow the traditional ER modelling methodology of modelling entities first, relationships second and attributes last. The interactions also showed that some errors in the students' solutions were left undetected. We enhanced the constraint base to identify these errors by adding more constraints.

Students experienced a number of problems with the system while interacting with KERMIT. The video footage of interactions was very useful in pinpointing the exact bugs of the system. All the bugs identified during the preliminary study were fixed in order to make the system more robust.

## 5.2 Classroom evaluation study 1

An experiment was designed and conducted to evaluate the effectiveness of KERMIT and its contribution to learning. The study involved students learning database modelling, and compared their learning using the fully functional KERMIT against a control group that used a cut-down version of KERMIT. The system used by the control group provided no feedback except for the complete solution upon request. Our intention was to have a cut-down version of the system that

would be similar to a classroom situation where students only get to see the ideal solution. The study also assessed the students' perception of the two systems.

The evaluation 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 [Elmasri & Navathe, 1994]. The students who participated in the study had previously learnt database modelling concepts in the lectures and labs of the course.

During the design phase of the study, we considered other options for the participant population, such as mature students from a continuing education course. Although they can be viewed as a respectable population for the evaluation study since they are generally highly motivated, they are typically low in computer literacy, having limited experience using computers. Therefore, they may struggle with the interface of KERMIT, distracting them from the goal of learning ER modelling. Moreover, since they are not typical students learning ER modelling, the results cannot be generalised to typical students.

### 5.2.1 Process

The study was conducted in the computer laboratories of the School of Mathematical and Computer Sciences at VUW. The experiment involved two versions of KERMIT:

- Experimental group - KERMIT with comprehensive feedback capabilities;
- Control group - a trimmed down version of KERMIT that only offered the complete solution as feedback (named ER-Tutor).

The interfaces of both systems were similar, but with the option of selecting feedback and the feedback textbox missing from the ER-Tutor. The study involved two one-hour sessions in succession. As participants randomly chose a session, the students in the first session were treated as the experimental group and the other as the control group.

Although the study was planned for a two-hour session, each group was only given an hour due to miscommunication between us and the lecturer of COMP302, and resource shortages at VUW. Moreover, the lab allocated to the evaluation study contained only 18 terminals, which is insufficient for a population of 28 students.

Each session proceeded in four distinct phases:

- Pre-test

- Interacting with the system (K*ER*MIT/ER-Tutor)
- Post-test
- Questionnaire

Initially each student was given a document that contained a brief overview of the study and a consent form. After signing the consent form, the students sat a pre-test. At the completion of the pre-test, they began interacting with the system. At the end of the interaction, participants were given a post-test and a questionnaire. The time allowed for completing the tests was not strictly supervised. As soon as the students completed their pre-test they started interacting with the system. 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.

The following subsections explain each of the four phases of the experiment in depth, discussing the design decisions made.

### 5.2.2 Pre- and post-test

A pre- and post-test was included in the study to evaluate the students' knowledge of ER modelling before and after interacting with the system. The pre-test verifies that the experimental and control groups have similar knowledge in ER modelling prior to the study, and that the two groups are comparable. Students' learning during the session can be quantified by comparing the results of both tests. If students, on average, scored higher in the post-test, it can be concluded that they acquired knowledge by interacting with the system. The increases or decreases in score of the two groups (experimental and control) can be compared to approximate the difference in effectiveness of K*ER*MIT and the control system. In other words, if the experimental group show a higher improvement in the post-test compared to the control group, it suggests that students learn more interacting with K*ER*MIT than with ER-Tutor (the control system).

Since the results of the pre- and post-test are compared to assess knowledge acquisition, both the pre-test and the post-test should either be identical or of a similar complexity. Although ideally the pre- and post-tests should be identical for comparison, students may recall test questions in their second attempt. To minimise any prior learning effects we designed two tests (A and B) that contained different questions, but of approximately the same complexity. 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. Therefore the effect of bias from a particular test was reduced.

Designing the tests A and B was complex due to the limited time that was allocated for each test and the goal of evaluating students' knowledge in ER modelling. The tests were designed to be completed in less than ten minutes. They contained two questions: a multiple choice question to choose the ER schema that correctly depicted the given scenario and a question that involved designing an ER schema, asking the students to design a database that satisfied the given set of requirements. The pre- and post-test used are given in Appendix B.

The multiple choice question requires less effort from students and demands less time. Since guessing may be involved in making a selection, a single multiple choice question does not give a good evaluation of the student's knowledge. The rationale behind adding a design question was to comprehensively evaluate the student's knowledge. Moreover, as the question is close to a database design task in the real world, sound assumptions on the student's performance in designing databases in the real world can made. This type of question is also excellent in evaluating the effectiveness of KERMIT in supporting students learning to design databases, as KERMIT offers problems of a similar fashion.

### 5.2.3 Interaction with the system

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 the experimental and control group. The students who were using KERMIT were required to complete the current problem before moving on to the next one, whereas the students in the control group were free to skip problems as they pleased.

A total of six problems were ordered in increasing complexity (see Appendix D for the list of problems and their ideal solutions). The first three in the list were introductory level problems. It was expected that an average student would spend approximately ten minutes each on these. The fourth problem involved constructing a database model of a moderate complexity. We expected students to spend about half an hour on this. The two final problems were challenging and call for a considerable amount of work. They were aimed at more able students, anticipating that they would complete the initial problems quickly.

### 5.2.4 System assessment

The system assessment questionnaire recorded the student's perception of the system. The questionnaire contained a total of 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. They included questions such as *"what did you find frustrating about the system* and *any suggestions for improving the system"*.

The questionnaire is included in Appendix C.

## 5.3 Results and analysis of study 1

This section summarises the results of the data obtained from all four stages of the class experiment at VUW. Initially, data recorded from students interacting with each version of the system are summarised. Then, we present the results of the subjective analysis by summarising students' responses to the user questionnaire. Finally, the scores of the pre- and post-tests are outlined.

### 5.3.1 Interacting with the system

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. Data on students interacting with the system, such as the total time spent interacting with the system and the total number of problems attempted was found by analysing the logs. The data extracted from the student logs are summarised in Table 5.2.

Although both groups were given approximately equal duration for the experiment, students in the control group had spent eight minutes longer interacting with the system. One reason for the difference is that the students who used KERMIT were given an online tutorial, which demonstrated how to construct ER diagrams and how to highlight words of the problem text to indicate each constructs' semantic meaning. Moreover the time limit was not strictly enforced on the second group (the control group) since the laboratory was not reserved for the following hour. Some students in the control group interacted with the system longer.

The students who used KERMIT spent more time per problem. The students in the experimental group spent fourteen minutes while students in the control group spent nine. The difference in the amount of time spent on a problem can be explained by the fact that students using KERMIT were

forced 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 of both groups. Students in the control group had attempted 3.50 problems compared to the 1.64 problems attempted by the experimental group. However, the experimental group managed to complete 1.21 problems out of 1.64 on average with a completion rate of 74%. On the other hand, the control group managed to complete only 0.86 out of the attempted 3.50 problems on average. The control group had a low completion rate of 25%.

| | 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 |

Table 5.2: System interaction details

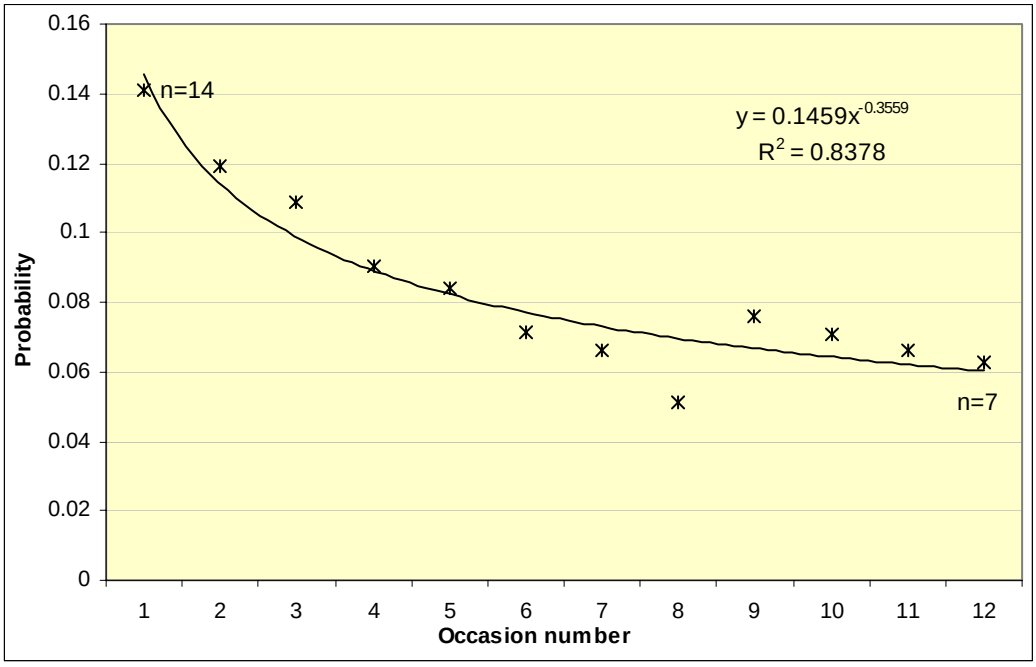



Figure 5.2: Probability of violating a constraint as a function of the occasion when that constraint was relevant, averaged over subjects of Study 1

The student logs were also used to analyse the mastery of constraints as explained in Section 5.1.1. Figure 5.2 illustrates the probability of violating a constraint plotted against the number of occasions when the constraint was relevant averaged over the participants. 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%. The number is low because the students had learnt ER modelling in lectures and practised constructing ER models in tutorials. 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.

### 5.3.2 Subjective analysis

This section summarises the responses to the user questionnaire. All the participants completed a questionnaire at the end of the study. Table 5.3 gives the mean responses of the participants regarding their attitude towards the system.

The results show that the students' mean response to the amount of time required to learn the interface of KERMIT was 13.21 minutes whereas the control group required 10 minutes. This can be explained by the difference in sophistication of the interfaces of KERMIT and ER-Tutor. Since KERMIT evaluates student solutions, 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. Although the students were offered a tutorial that explained and demonstrated highlighting words in the problem text, students required extra time to complete the tutorial.

| | 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 |

Table 5.3: Mean responses from the user questionnaire

When asked to rate the amount they learnt from the system on a scale of 1 to 5 (very poor to very good), the mean response was 2.43 for the group that used KERMIT and 2.64 for the group that

used ER-Tutor. Although the control group has on average ranked the amount learnt 0.21 higher, the difference was found to be statistically insignificant when the data was analysed using a two tailed independent samples t-test. There are a few factors that may have influenced such low mean rankings for the amount learnt from KERMIT. The students in the experimental group had less time to spend on solving problems in comparison to the control group (refer to Table 5.2). Moreover, as a result of the students in the experimental group being forced to complete a problem before moving on to a new one, they had very little opportunity to attempt more than two problems.

Students who used KERMIT had commented that the time they were allowed to interact with the system was too short. Some of the free-form comments are itemised below.

- *Time is very short.*
- *The time provided to learn ER modelling was not enough (which was 1 hour). If time allowed was more could have been able to go through more problems.*
- *I could probably have learnt more with a little more time.*
- *I think if I had more time with the system I'd figure it out.*

The KERMIT group gave a mean rating of 3.64 for the enjoyment and the control group gave a rating of 3.43 for ER-Tutor. The difference is too small to be statistically significant. The experimental group had given a slightly higher mean rating of 3.00 for the usefulness of KERMIT's feedback whereas the control group had given a mean rating of 2.79 for ER-Tutor, which is also a statistically insignificant difference. Moreover, some students had written free-form comments that confirmed the assumption of insufficient help from ER-Tutor. Some comments are itemised below.

- *Would be good to know what you did wrong. i.e. tips on how you can improve your solution based on the system's solution*
- *Only solution? What about some hints?*
- *Solutions didn't help me much without reasons*
- *What feedback? Only feedback I got was a "right" diagram no comments on what I did wrong or even if my diagram was right*
- *Suggestions or any error notifications about the errors made by the users are very helpful to the user*

In response to the question "Would you prefer more details in feedback?" 71% of the group that interacted with KERMIT answered *yes*, in comparison to 86% of the control group who answered *yes* (see Table 5.4). Moreover, 21% of the experimental group said that they did not require more feedback and 14% of the control group made the same response. 7% of the students in the control

group were unsure as to the amount of detail in feedback content. This was not unexpected, as the students did not have a sufficient period of time to become familiar with the system.

| | *Yes* | *No* | *Don't know* |
|---|---|---|---|
| KERMIT | 71.43% | 21.43% | 7.14% |
| ER-Tutor | 85.71% | 14.29% | 0% |

Table 5.4: Responses to whether more detailed feedback was preferred

The student responses to whether they would recommend the system to others are given in Table 5.5. The data shows that 86% of subjects from both groups had noted that they would recommend the respective versions of the system that they used. The remainder of the subjects who used KERMIT were undecided in recommending the system and the remainder who used the control system had rejected the possibility of recommending the system to others. The fact that there were 14% of the student from the experimental group that were undecided illustrates that interacting with KERMIT for only twenty minutes is insufficient to evaluate the quality of the system.

| | *Yes* | *No* | *Don't know* |
|---|---|---|---|
| KERMIT | 86% | 0% | 14% |
| ER-Tutor | 86% | 14% | 0% |

Table 5.5: Responses to whether they would recommend the system to others

Students who used KERMIT awarded a mean rating of 3.50 when asked how easy it was to use the interface, whilst the other group who used the control system awarded a mean rating of 3.71. Since KERMIT's interface is more complex in comparison to the control system, students initially find its interface complicated to use. However, once they become accustomed to ER modelling with KERMIT we expect them to be more productive in learning ER modelling with KERMIT than the control group.

The students who used KERMIT were very enthusiastic about using the system. They had written positive comments about the system:

- *Feedback from KERMIT is very useful*
- *Nice easy to use interface*
- *Highlighting the text and having it appear in the attributes etc. was helpful*
- *I liked the simplicity to use the product*
- *It was straightforward*

- *It has very useful hints that guide you on. Plus the solutions are handy when you're bogged down.*
- *It's a fun & simple way of learning ER. It doesn't seem tedious and boring.*
- *Easy to use easy to understand*

### 5.3.3 Pre- and post-test performance

All students who participated were pre- and post-tested. The mean scores are given in Table 5.6. The experimental group scored a mean of 5.9 out of a possible 12 for the pre-test. The control group scored a mean of 6 for the pre-test. Since the difference of mean scores is statistically insignificant we can conclude that the two groups have similar knowledge in ER modelling and are comparable.

| | Pre-test | s. d. | Post-test | s. d. | Gain score | s. d. |
|---|---|---|---|---|---|---|
| KERMIT | 5.86 | 1.46 | 6.50 | 2.47 | 0.64 | 3.27 |
| ER-Tutor | 6.00 | 2.18 | 6.29 | 2.09 | 0.29 | 2.46 |

Table 5.6: Mean pre- and post-test scores of the experimental and the control group

The group of students who interacted with KERMIT scored 6.50 on average for the post-test, and the group who interacted with the control system (ER-Tutor), scored 6.29. The experimental group scored 0.64 higher in the post-test, whereas the control group gained 0.29. The t- test revealed that the difference between the gain scores is statistically insignificant. A statistically significant gain cannot be expected from such a short interactive session with the system. However, it is promising at least that the value of the mean gain score of KERMIT was higher than the mean gain score of the control system. Moreover, the student's performance did not degrade after interacting with the system, which confirms that the teaching system cannot have any negative impacts on ER modelling knowledge.

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., 2001a]. Both report experiments where students used

the system for a whole two-hour session. Better results on the effect size have been obtained in studies where interactions lasted for a whole semester or an academic year. Bloom [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 [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, power of 0.5 means that half of the repeated experiments would produce non-significant results. Chin [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.

Test A and B, used for the pre- and post-test, were designed to be of similar complexity. We compared the students' performance on test A and B for the pre-test. Only the scores of the pre-test were taken into account to ensure the effects of interacting with different systems were discounted. The analysis revealed that students who sat test A had a mean score of 6.64 with a standard deviation of 1.7. On the other hand, the group who sat test B recorded a mean score of 5.21 with a standard deviation of 1.7. Although we assumed that the students would find test A and test B to be of a similar complexity, the students found test A slightly easier than test B. Kenneth Koedinger refers to this phenomenon of the experts making erroneous assumptions of the students as the 'expert blind spot' [Koedinger, 2001].

### 5.3.4 Discussion

Although the evaluation study was too short, the results from Study 1, conducted at Victoria University, has yielded some promising results. Students who used KERMIT displayed a slightly higher gain score in comparison to the control group. Even though the differences are statistically insignificant, the findings are quite promising for such a short study. Most importantly, the pre- and post-test scores demonstrated that using KERMIT did not hamper students' abilities in ER modelling.

Students using KERMIT experienced a number of system crashes during their interaction. The main reason for the crashes is the inadequate testing of the system running in the environment used for the study. There was little opportunity to test the systems at the computer laboratories of Victoria University where they were running on a Windows 2000 terminal server. Since the system was designed to run as a stand-alone program, a number of unexpected bugs emerged during the study. The control system (ER-Tutor), on the other hand, was more stable in comparison to KERMIT as it was not required to evaluate the student's solution. It would be fair to assume that the system crashes of KERMIT had a detrimental impact on the students' perception of KERMIT.

One other factor that influenced the outcome was the students struggling to familiarise themselves with highlighting words in the problem text to specify each construct's semantic meaning. A typical mistake made by the students was to add a new construct to their workspace, highlight a word from the problem text and rename the construct to have a different semantic meaning. In such cases KERMIT struggles to 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 (see Section 4.3.2 for details). This confusion can be minimised by preventing the users from renaming constructs, and by automatically naming constructs using the highlighted portion of the problem text. The change to the system would not be expected to hamper student's learning, but would reduce the burden of having to type in the name of each construct. A rise in student performance could be expected as a result.

Students who used KERMIT were forced to complete each problem before moving on to the next problem, whereas students in the control group chose a new problem at any instance they were dissatisfied with the current one. 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.

Some students had difficulty in understanding the feedback messages presented by KERMIT. The feedback messages of the version of KERMIT used in the Wellington study were short and less

descriptive. Some messages delivered small hints that novice users struggled to understand. We plan to revise the feedback messages of the constraint base making them more descriptive.

Analysing the pre-test scores of both test A and B revealed that the students found test B harder in comparison to test A. The pre- and post-tests must be of a similar complexity to be comparable. Since the students found two tests to be of different complexities, the gain scores of both groups cannot be trusted to portray true knowledge gains.

## 5.4 Evaluation study 2

An experiment similar to Classroom Study 1 was carried out at the University of Canterbury, Christchurch. Similar to the prior evaluation, this study also involved a comparison of a group of students learning ER modelling by using the fully functional KERMIT against a control group who used a cut-down version of KERMIT, named ER-Tutor. The study involved sixty-two volunteers from students enrolled in the Introduction to Database 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 [Elmasri & Navathe, 1994]. The students had learnt ER modelling concepts during two weeks of lectures and had little practice during two weeks of tutorials.

### 5.4.1 Process

The evaluation study was conducted during two streams of two-hour laboratory sessions. Each session proceeded in four distinct phases identical to the evaluation conducted at the Victoria University. The students started the session by reading the consent form and signing it. At the completion of the pre-test, they started interacting with the system. Students were given the freedom of choosing the amount of time they interacted with the system with a maximum time of approximately one and half hours. At the completion of their interaction the students sat a post-test and completed a questionnaire. To preserve anonymity, the students were asked to logon to the system by using their machine number. They were also asked to indicate their machine number in the pre-test, post-test and questionnaire to be able to match the logs of the student's interaction with the pre-test, post-test and their questionnaire.

The experiment involved two versions of KERMIT:

- KERMIT – Fully functional KERMIT with comprehensive feedback
- ER-Tutor – A trimmed version of KERMIT that only offers the complete solution to the problem.

The group who used KERMIT was treated as the experimental group and the other was treated as the control group. The interfaces of both the systems were similar, with the ER-Tutor only allowing the students to view the final solution, whereas KERMIT allowing students to select feedback from six levels of feedback messages, including the complete solution. The interface of KERMIT also contained a feedback textbox to display the feedback messages, which was absent in the ER-Tutor's interface. The main difference between the two systems, KERMIT and ER-Tutor, was that the students using KERMIT were forced to indicate the semantic meaning of each construct by highlighting a word in the problem text whereas the students using ER-Tutor had no such requirement.

Each session (session A and B) was conducted in two adjacent laboratories (lab A and B). The numbers of participants present during each session is shown in Table 5.7.

| | Lab A | Lab B | Total |
|---|---|---|---|
| Session A | KERMIT (29 students) | ER-Tutor (9 students) | 38 students |
| Session B | ER-Tutor (24 students) | KERMIT (4 students) | 28 students |

Table 5.7: Student participation during each session

The following subsections discuss each phase of the evaluation study.

### 5.4.2 Pre- and post-test

The pre- and post test were used in the evaluation as a measure to obtain an approximation of the student's knowledge in ER modelling before and after interacting with either version (KERMIT / ER-Tutor) of the system. The two tests (A and B) used for Study 1 were modified to be of a more similar complexity since the results from Study 1 showed that students found one test to be harder than the other.

According to the results of Study 1, 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 Study 1 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 by underlining the attribute name with a broken line. Both tests contained similar scenarios that produced similar ER models.

The results of Study 1 showed that students struggled to answer the question that involved composing an ER schema that involved a weak entity found in test B. In order to make the tests more similar, we replaced the question's scenario with a new scenario that yielded an ER diagram similar to the corresponding question in test A.

The new pre- and post-test used in Study 2 are attached in Appendix B.

### 5.4.3 Interaction with the system

Students in the experimental group interacted with KERMIT and students in the control group interacted with ER-Tutor. Each student worked on the system individually solving the problems presented to him or her. Both systems contained a set of six problems ordered according to increasing difficulty. The set of problems and their order was identical to the problems offered during Study 1. The difference from the two versions of KERMIT used in Study 1 was that the students were given more freedom in selecting a problem to work on. Both systems allowed the student to skip to the next problem or go back to a previous problem without completing the current problem.

Since KERMIT's constraint-base was enhanced after close scrutiny, the feedback offered by KERMIT in this study was slightly different to feedback offered by KERMIT during Study 1. The feedback messages were reworded according to the suggestions made by the students who participated in Study 1. The version of KERMIT used in this study was also more stable in comparison to the version used at the Victoria University. This was achieved by identifying and correcting a number of bugs in the system.

### 5.4.4 System assessment

At the completion of the study a questionnaire was given to each student to record his/her perception of the system. The questionnaire contained fourteen questions identical to the questionnaire used for Study 1. It included questions where the student was asked to rank their

perception on different issues on a Likert scale of 1 to 5 and also provided space for the student to make free-form comments.

A copy of the questionnaire can be found in Appendix C.

## 5.5 Results and analysis of study 2

This section summarises the results obtained from each of the four phases of the experiment. The following subsection presents the summary of interaction details of each group. Section 5.5.2 outlines a summary of the data obtained from the questionnaire. Finally, the results of the pre- and post-test are discussed.

### 5.5.1 Interacting with the system

The student logs recorded all the useful events such as logging in, selecting a new problem, submitting a solution, etc. The logs were analysed to discover student interaction details such as the amount of time they spent with the system solving problems, total number of problems solved, etc. The results are summarised in Table 5.8.

The students in the experimental group spent a total of approximately 1 hour and 7 minutes on average interacting with the system and solving problems, and the students in the control group spent 58 minutes. Since the amount of interaction time was not forced upon the participants, although the difference is not significant, it is encouraging to note that students who used KERMIT were more willing to interact with the system. Even though the difference in the interaction times is small in this experiment, we can expect a greater difference for an experiment spanning a longer time period.

During their interaction, students in both groups attempted a similar number of problems. The experimental group attempted a total of 4.36 problems and the control group attempted a total of 4.10 problems on average. The numbers of completed problems out of the attempted problems were also similar. The experimental group had completed 1.75 problems and the control group had completed 1.97 problems. Both the groups had a similar completion rate: 40% for the experimental group and 48% for the control group.

The average times spent on completing a problem for both groups were very similar, with both groups spending approximately 23 minutes. 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.

| | 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 |

Table 5.8: Mean system interaction details

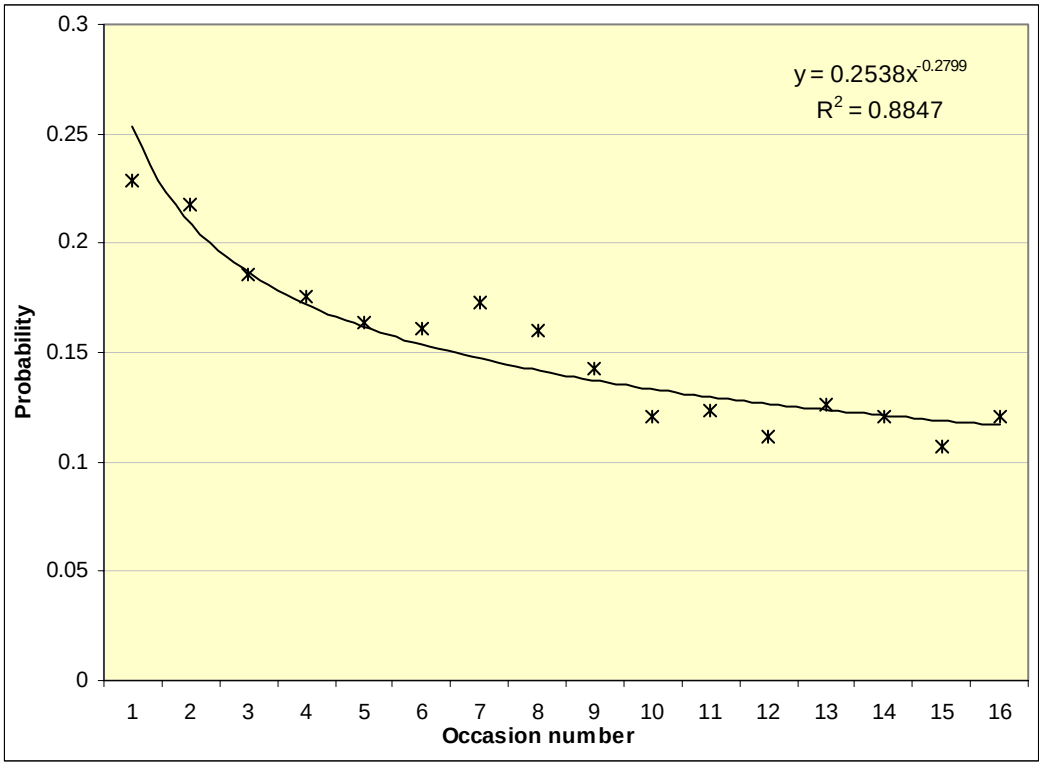



Figure 5.3: Probability of violating a constraint as a function of the occasion when that constraint was relevant, averaged over subjects of the class Study 2

The mastery of constraints was analysed by examining the student logs as explained in Section 5.1.1. Figure 5.3 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 discovered during Study 1, where a

decrease of 45% was displayed. The results of the mastery of constraints for this experiment further strengthen the hypothesis that the students learn ER modelling by interacting with KERMIT.

### 5.5.2 Subjective analysis

All the participants were given a questionnaire at the end of their session to determine their perceptions of the system. Table 5.9 displays a summary of the responses ranking student perceptions on a Likert scale of 1 to 5 where 1 represents strongly disagree and 5 represents strongly agree.

| | 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 |

Table 5.9: Mean responses from the user questionnaire for Study 2

The mean responses show that 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 results show 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 5.8.

The mean response when asked to rate the amount of ER modelling 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 statistically significant when the two data sets were compared using the two tailed independent samples T-test ($t = 0.65$, $p = 0.05$). These results are analogous with our hypothesis that KERMIT's extra functionality, such as offering feedback, aids the students to learn more concepts of ER modelling in comparison to the control system. The comments written in the questionnaire given to the group who used KERMIT further supported our hypothesis. Some free-form comments are itemised below.

- *Without the feedback I wouldn't have learnt from my mistakes*
- *Quick feedback allowed me to learn from my mistakes*

- *The different levels of feedback is a good feature*
- *It was helpful that it gave the user 'detailed hints'*
- *Was able to realise problems in relationships/entities myself through hints rather than being given the solution*
- *The hints that the genie gave were informative and helpful (sometimes) and these helped me learn*

Both groups of students, on average, rated their enjoyment of the system on a similar scale. The difference of the mean responses to the question how *'much did you enjoy learning from the system'* was very small, making it statistically insignificant. However, students who used KERMIT had written some positive comments as itemised below.

- *Not sure why, but really enjoyed it*
- *It was simple and informative*
- *Found it frustrating at first as the interface wasn't very user friendly towards me! But once I got used to it, it was more enjoyable*

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. KERMIT forces the student to highlight a word in the problem text to indicate the semantic meaning of each of his/her constructs. ER-Tutor, on the other hand, gives the student the freedom to type in any name for each construct by double clicking it.

The mean rating for the usefulness of feedback presented by each system is considerably higher for the experimental group who interacted with KERMIT. The students who interacted with KERMIT had rated the usefulness of its feedback as 3.42 on average and the students in the control group had 2.45. The difference of means 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. The comments made by the students who used ER-Tutor (as itemised below) showed their frustration of not getting any further feedback from the system.

- *No feedback given*
- *Brief reasoning might be nice*

- *There is not much feedback*
- *The genie did not give any help*
- *Cardinality checking would be good*
- *Solutions good but no other feedback*

In response to the question '*would you prefer more details in feedback?*', students who used ER-Tutor indicated the need for more detailed help other than the complete solution (detailed results given in Table 5.10). Twenty three out of thirty students (approximately 74%) students indicated that they would prefer more feedback. On the other hand, 61% of the students who used KERMIT also indicated that they would prefer more feedback. Students in the experimental group had indicated in their free-form responses the need for problem specific help or hints that deal with the semantics of the problem. Currently KERMIT's hints are in a general form for them to be used in any problem regardless if the problem's semantics.

| | *Yes* | *No* | *Don't know* |
|---|---|---|---|
| KERMIT | 61.29% | 22.58% | 16.13% |
| ER-Tutor | 74.19% | 9.68% | 12.90% |

Table 5.10: Responses to whether more detailed feedback was preferred

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 as shown in Table 5.11. Approximately 84% of the students who used KERMIT indicated that they would recommend the system to others. The proportion of students in the control group who indicated the willingness to recommend the system to others was lower, approximately 68%. The student's better perception of KERMIT was clearly reflected in the comments made by some students who used KERMIT (itemised below).

- *It was a helpful step-by-step learning process, good for beginners who just started learning ER modelling*
- *The ease of use and highlighting of the text used*
- *Good learning idea – lets you do things at your own speed. Also lets you skip ahead if you wish*
- *Easy to use good GUI*

| | *Yes* | *No* | *Don't know* |
|---|---|---|---|
| KERMIT | 83.87% | 9.68% | 6.45% |
| ER-Tutor | 67.74% | 3.23% | 29.03% |

Table 5.11: Responses to whether the system can be recommended to others

### 5.5.3 Pre- and Post-test performance

The mean scores of the pre- and post-test are detailed in Table 5.12. Both groups had similar pre-test scores of 16.16 and 16.58 out of a possible 22. The difference in scores between both groups is statistically insignificant, confirming that the two groups initially had equal knowledge in ER modelling and that they are comparable.

| | **Pre-test** | **s. d.** | **Post-test** | **s. d.** | **Gain score** | **s. d.** |
|---|---|---|---|---|---|---|
| KERMIT | 16.16 | 1.82 | 17.77 | 1.45 | 1.65 | 1.72 |
| ER-Tutor | 16.58 | 2.86 | 16.48 | 3.08 | -0.10 | 2.76 |

Table 5.12: Mean pre- and post-test scores for Study 2

The mean post-test score of the experimental group was 17.77, with a mean gain score of 1.65. The gain score is statistically significant ($t = 4.91$, $p = 0.01$), suggesting that the students' knowledge increased by using KERMIT. Conversely, the difference in pre- and post-test of the group who used ER-Tutor is statistically insignificant. An independent-samples T test performed on the gain scores of the two groups revealed that the gain scores of both the groups are significantly different ($t = 3.07$, $p = 0.10$). We can conclude from these results that students who used KERMIT learnt more about ER modelling using KERMIT than students who used the control system.

The statistical effect size and power for the experiment were calculated. The common method used in the ITS community to calculate the effect size yields $(1.65 – (– 0.10)) / 2.76 = 0.63$. The resulting effect size of 0.63 is comparable with the effect size of 0.63 published by Albacete and Vanlehn [Albacete & VanLehn, 2000] and 0.66 published in [Mitrovic, et al., 2001a]. 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 was also calculated using the Chin's proposed method of calculating the effect size as an $\omega^2$ value [Chin, 2001]. The $\omega^2$ value was calculated as 0.12. As Chin points out that an effect size of 0.15 is considered large in social sciences [Chin, 2001], 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 or sensitivity of the experiment, which is a measure of how easily the experiment can detect differences. Chin recommends that researchers should strive for a power of 0.8 [Chin, 2001]. The power of this experiment was calculated as 0.75 at significance 0.05, which is an excellent result. The power value of 0.75 signifies that 75% of the repeated experiments would produce significant results.

### 5.5.4 Discussion

The results gathered from classroom Study 2 show that students' knowledge increased by using KERMIT. Students who interacted with KERMIT showed a significantly superior gain in their scores, suggesting that they acquired more knowledge in ER modelling. This hypothesis was further strengthened by the students' responses to the question regarding their perception on the amount of material they learnt by using the system. Students who used KERMIT indicated a higher mean response with a statistically significant difference. The students' free-form comments also suggested that students found KERMIT's hint messages useful.

Although the difference in the mean student ranking for the student's perception on the amount learnt is statistically significant, it is surprising to record a high mean ranking of approximately 3 for the control group. 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.

The students who used KERMIT during the first session (where 29 of the 38 student in the experimental group were involved) found the system was painfully sluggish. The main reason for this behaviour was the extremely high load placed on the Windows 2000 terminal server that ran KERMIT. The first session, which recorded the worst response times of about ten seconds, involved approximately sixty simultaneous users being logged on to the server. Although the system was considerably more stable due to its enhancements after classroom Study 1, the sluggish responses also resulted in system crashes. Students who did not know that the system was evaluating their

solutions in the background attempted to modify their ER diagrams, which resulted in the system crashing. On the other hand, the students who used ER-Tutor did not experience the sluggish response times. This was mainly because ER-Tutor consumes considerably less processing power, as it does not evaluate the student's solution. Moreover, the lower number of users (around forty students) logging on to the terminal server during the evaluation of ER-Tutor also resulted in a system that ran faster. These factors were sure to influence the student's perception of KERMIT. This was evident when the questionnaires were analysed with most students complaining about the slowness of KERMIT.

The major improvement in the results from the previous study at Victoria University can be attributed to a number of reasons. A major factor that influenced significant gains in the student's knowledge in ER modelling can be attributed to the experiment spanning a longer time. The students who participated in Study 2 interacted with KERMIT for approximately an hour, whereas students in Study 1 only interacted for approximately 20 minutes. Students in Study 2 attempted more problems and completed more problems as a result of the longer interaction time.

Other factors that influenced student learning in Study 2 can be attributed to a more stable KERMIT, an enhanced constraint base with reworded feedback messages. The implementation of KERMIT was fine-tuned and a number of bugs were identified and fixed. The resultant stable system was a result of intensive testing of the system before the evaluation. The constraint base was also enhanced by adding a further eight constraints. The feedback messages of the system was also reviewed and reworded to be more descriptive. The enhanced set of feedback messages has influenced the student's perception on the system's feedback as a whole, which was reflected in their responses in the questionnaire. Approximately 71% of the students in Study 1 who used KERMIT indicated that they would prefer more detailed feedback. The percentage of students who indicated the need for more feedback dropped to 61% in Study 2. The percentage of students who indicated that they do not require more detailed feedback had also increased from 7% in Study 1 to 23% in Study 2. Moreover even though the difference of the mean ranking of the amount learnt and usefulness of feedback between the experimental and the control was insignificant in Study 1, the difference shown in Study 2 was statistically significant. In other words, students who used KERMIT in Study 2 indicated that they learnt a significant amount more from the system and they found the feedback more useful in comparison to the control group.

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. The enthusiasm for the system even extended to one student volunteering for two experimental sessions.

# Chapter 6

# Conclusions

This thesis has discussed the design and implementation of an Intelligent Teaching System, named KERMIT, developed to assist students learning the popular database modelling technique, ER modelling. KERMIT's effectiveness in teaching ER modelling to students was evaluated extensively with three evaluation studies, including two classroom experiments. The results of the final classroom evaluation showed that KERMIT is an effective educational tool that can be used to enhance student learning. Participants in the study who used 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 remainder of the chapter is organised as follows. Section 6.1 outlines KERMIT, including a brief overview of its implementation. Section 6.2 presents a summary of findings during the evaluation studies. Finally Section 6.3 discusses future directions of research.

## 6.1 KERMIT

KERMIT [Mitrovic, et al., 2001b; Suraweera & Mitrovic, 2001] is an Intelligent Teaching System developed to assist students learning ER modelling. Since ER modelling, like other design tasks, can only be learnt through extensive practice, KERMIT is developed as a problem-solving environment in which students can practice their skills. The system presents a description of a scenario for which the student has to model a database using ER modelling constructs. When the student requires guidance from the system, KERMIT evaluates the solution and presents hints regarding the errors in the solution. The system adapts to each student, therefore such hints and pedagogical instructions are individualised to guide the student towards the correct solution.

KERMIT is implemented in Microsoft Visual Basic and supports the ER model defined by Elmasri and Navathe [Elmasri & Navathe, 1994]. The main components of KERMIT include the pedagogical module, student modeller and its interface. The pedagogical module acts as the driving engine of the entire teaching system by selecting the instructional messages presented to the student and selecting new problems that best suit the student's knowledge level. The student modeller is implemented using constraint based modelling [Ohlsson, 1994], a student modelling approach that focuses on errors. It evaluates the student solution against the system's knowledge base and records the student's knowledge in the form of a student model. Students interact with the system through the interface and model solutions to the questions using the ER modelling workspace.

The knowledge base of KERMIT is an integral part of the system. The feedback messages are generated from the knowledge base. KERMIT's knowledge base consists of 92 constraints, which deal with both syntactic and semantic errors. The knowledge acquisition phase involves the exploration of a number of avenues, such as analysing the target domain through literature and analysing student's solutions to an exam question.

An authoring tool has also been developed for teachers and other database professionals to add problems and their solutions into the system. The tool supervises the process of adding a new problem and adds the problem and its ideal solution to their respective databases after converting them to their respective internal representations. The authoring tool has made the system able to evolve without the need for programming resources.

## 6.2 Evaluations and results

A total of three studies were conducted to evaluate the effectiveness of KERMIT and to obtain the students' perception of KERMIT. The initial study was a preliminary evaluation, which was conducted in the form of think-aloud protocols. Although this study did not result in any concrete data to formulate any theories on the effectiveness of KERMIT, it yielded some positive comments regarding the students' perception of the system. The students involved in the study also discovered a variety of bugs in the system, which, once corrected, helped to make the system more stable.

The second study was conducted as a classroom experiment where students' learning was monitored by pre and post-testing. The student's perception of the system was attained from the completion of a questionnaire. The results of the study were mainly influenced by the limited amount of time available. The students' perception of the system was also heavily affected by system crashes. Although the results of Study 1 were not significant, the study yielded some

promising results. The students who used KERMIT demonstrated a slightly higher gain in comparison to the control group. Most importantly the study showed that KERMIT was at least as effective as the control system (a conventional drawing tool) and that it did not hamper the student's abilities.

The third evaluation study, conducted in a similar manner to the second study as a classroom experiment, produced a number of significant results, showing that learning increased from using KERMIT. Students who interacted with KERMIT demonstrated a significant improvement in their post-test in comparison to the control group, confirming that they acquired more knowledge in ER modelling. The students' responses to the questionnaire reiterated that they had a good perception of KERMIT. The students who used KERMIT ranked the amount of knowledge they gained from the system, and the usefulness of the system's feedback at a significantly higher scale. Moreover, the free-form comments made by the students in the questionnaire also strongly indicated that they enjoyed using KERMIT than the control system.

The student modelling technique used in KERMIT, Constraint Based Modelling (CBM), has previously been used to represent domain and student knowledge in SQL-Tutor [Mitrovic, 1998a; Mitrovic, 1998c; Mitrovic & Ohlsson, 1999] and in CAPIT [Mayo, et al., 2000]. 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 basic units of knowledge. The research presented in this thesis demonstrated that CBM can also be used to effectively represent knowledge in domains with open-ended tasks such as database modelling. This is an important result of this research that further strengthens the credibility of CBM.

## 6.3 Further research directions

This research has produced an effective intelligent teaching system for ER modelling. There are a number of future avenues that can be explored to further improve KERMIT's effectiveness. KERMIT currently requires the user to indicate the semantic meaning of each construct by highlighting a word from the problem text. Students find this too constraining. 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.

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 problem specific help. Since ER models are very closely linked to the problem's domain, a major portion of the students' problems are specific to the domain of the problem. A further enhancement to KERMIT's feedback would be to guide the student in the process of self-explanation. This could be performed in the form of a set of questions prompted to verify the student's 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 Bayesian network can be used in selecting feedback and selecting new problems for the student.

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.

# Bibliography


[Ahrens & Sankar, 1993] Ahrens, J. D. & Sankar, C. S. Tailoring Database Training for End Users. *Management Information Systems Quarterly*, *17* (4), 1993, pp. 419-439.

[Albacete & VanLehn, 2000] Albacete, P. L. & VanLehn, K. The Conceptual Helper: an Intelligent Tutoring System for Teaching Fundamental Physics Concepts. *In* Gauthier, G., Frasson, C. and VanLehn, K. (eds.). *Proc. of 5th International Conference on Intelligent Tutoring Systems*, Montreal 2000, Springer, pp. 564-573.

[Aleven & Koedinger, 2000] Aleven, V. & Koedinger, K. Limitations of Student Control: Do students know when they need help? *In* Gauthier, G., Frasson, C. and VanLehn, K. (eds.). *Proc. of 5th International Conference on Intelligent Tutoring Systems*, Montreal 2000, Springer, pp. 292-303.

[Anderson, 1993] Anderson, J. R. *Rules of the Mind*. Erlbaum, Hillsdale, NJ, 1993.

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

[Batini, et al., 1986] Batini, C., Lenzerini, M. & Navathe, S. B. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, *18* (2), December 1986, pp. 323-364.

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

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

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

[Brown, et al., 1975] Brown, J. S., Burton, R. R. & Bell, A. G. SOPHIE: A Step Toward Creating a Reactive Learning Environment. *International Journal of Man-Machine Studies*, *7* (5), 1975, pp. 675-696.

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

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

[Choobineh, et al., 1988] Choobineh, J., Konsynski, B. R., Mannino, M. V. & Nunamaker, J. F. An Expert System Based on Forms. *IEEE Software Engineering*, *14* (2), February 1988, pp. 108-120.

[Constantino-Gonzalez & Suthers, 1999] Constantino-Gonzalez, M. d. l. A. & Suthers, D. D. A Coached Computer-Mediated Collaborative Learning Environment for Conceptual Database Design. *In* Lajoie, S. P. and Vivet, M. (eds.). *Proc. of Artificial Intelligence in Education*, Le Mans, France 1999, IOS Press, pp. 645-647.

[Constantino-Gonzalez & Suthers, 2000] Constantino-Gonzalez, M. d. l. A. & Suthers, D. D. A Coached Collaborative Learning Environment for Entity-Relationship Modelling. *In* Gauthier, G., Frasson, C. and VanLehn, K. (eds.). *Proc. of 5th International Conference on Intelligent Tutoring Systems*, Montreal 2000, Springer, pp. 324-333.

[Constantino-Gonzalez, et al., 2001] Constantino-Gonzalez, M. d. l. A., Suthers, D. D. & I., L. J. Designing and Evaluating a Collaboration Coach: Knowledge and Reasoning. *In* Moore, J. D., Redfield, C. L. and Johnson, W. L. (eds.). *Proc. of Tenth International Conference on Artificial Intelligence in Education*, San Antonio, Texas 2001, IOS Press, pp. 176-187.

[Corbett, et al., 1998] Corbett, A. T., Trask, H. J., Scarpinatto, K. C. & Hadley, W. S. A Formative Evaluation of the PACT Algebra II Tutor: Support for Simple Hierarchical Reasoning. *In* Goettl, B. P., Halff, H. M., Redfield, C. L. and Shute, V. J. (eds.). *Proc. of 4th International Conference on Intelligent Tutoring Systems*, San Antonio, Texas 1998, pp. 374-383.

[DDS] DDS, http://chillisource.com/dds/factsheet/index.html

[Dimitrova, et al., 1999] Dimitrova, V., Self, J. & Brna, P. The interactive maintenance of open learner models. *Artificial Intelligence in Education*, *50*, 1999, pp. 405-412.

[Elmasri & Navathe, 1994] Elmasri, R. & Navathe, S. B. *Fundamentals of Database Systems*. Addison Wesley, 1994, *2nd edition*.

[ER/Studio] ER/Studio, http://www.Embarcadero.com/products/Design/erdatasheet.htm

[Gordon & Hall, 1998] Gordon, A. & Hall, L. A Collaborative Learning Environment for Data Modelling. *In Proc. of FLAIRS '98*, Sanibel Island, Florida 1998.

[Hall & Gordon, 1998a] Hall, L. & Gordon, A. An Intelligent Learning Environment for Data Modelling. *In Proc. of 4th International Conference on Intelligent Tutoring Systems*, San Antonio, Texas 1998, pp. 608.

[Hall & Gordon, 1998b] Hall, L. & Gordon, A. Synergy on the Net: Integrating the Web and Intelligent Learning Environments. *In Proc. of WWW-based Tutoring Workshop at 4th International Conference on Intelligent Tutoring Systems*, San Antonio, Texas 1998, pp. 25-29.

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

[Kawaguchi, et al., 1986] Kawaguchi, A., Taoka, N., Mizoguchi, R., Yamaguchi, T. & Kakusho, O. An Intelligent Interview System for Conceptual Design of Database. *In Proc. of 7th European Conference on Artificial Intelligence Conference Services Ltd.*, London 1986, pp. 1-7.

[Kay, 2000] Kay, J. Stereotypes, Student Models and Scrutability. *In* Gauthier, G., Frasson, C. and VanLehn, K. (eds.). *Proc. of 5th International Conference on Intelligent Tutoring Systems*, Montreal 2000, Springer, pp. 19-30.

[Koedinger, 2001] Koedinger, K. COSC 420, Cognitive Modelling and Intelligent Tutoring Systems, University of Canterbury, Christchurch, 2001.

[Koedinger, et al., 1997] Koedinger, K. R., Anderson, J. R., Hadley, W. H. & Mark, M. A. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8* (1), 1997, pp. 30-43.

[Lesgold, et al., 1990] Lesgold, A., Laijoie, S. P., Bunzo, M. & Eggan, G. A Coached Practice Environment for an Electronics Troubleshooting Job. *In* Larkin, J., Chabay, R. and Shefic, C. (eds.). *Proc. of Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration*, Hillsdale, N. J. 1990.

[Lester, et al., 1997a] Lester, J. C., Converse, S., Kahler, S., Barlow, T., Stone, B. & Bhogal, R. The persona effect: Affective impact of animated pedagogical agents. *In Proc. of CHI '97*, Atlanta, March 1997, pp. 359-366.

[Lester, et al., 1997b] Lester, J. C., Converse, S. A., Stone, B. A., Kahler, S. E. & Barlow, T. S. Animated Pedagogical Agents and Problem-Solving Effectiveness: A Large-Scale Empirical Evaluation. *In Proc. of Eighth World Conference of AIED*, Japan, August 1997, pp. 23-30.

[Lim & Hunter, 1992] Lim, B. B. L. & Hunter, R. DBTool: A Graphical Database Design Tool for an Introductory Database Course. *SIGCSE Bulletin*, *24* (1), March 1992, pp. 24-27.

[Mayo, et al., 2000] Mayo, M., Mitrovic, A. & McKenzie, J. CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation. *In* Kinshuk, Jesshope, C. and Okamoto, T.

(eds.). *Proc. of Advanced Learning Technology: Design and Development Issues*, Los Alamitos, CA 2000, IEEE Computer Society, pp. 151-154.

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

[Mitrovic, 1998a] Mitrovic, A. Experiences in Implementing Constraint-Based Modelling in SQL-Tutor. *In* Goettl, B. P., Halff, H. M., Redfield, C. L. and Shute, V. J. (eds.). *Proc. of 4th International Conference on Intelligent Tutoring Systems*, San Francisco 1998, pp. 414-423.

[Mitrovic, 1998b] Mitrovic, A. A Knowledge-Based Teaching System for SQL. *In Proc. of ED-MEDIA/ED-TELECOM'98*, Freiburg 1998, pp. 1027-1032.

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

[Mitrovic, et al., 2001a] Mitrovic, A., Martin, B. & Mayo, M. Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor. *User-Modelling and User Adapted Interaction*, 2001, (in press).

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

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

[Mitrovic & Suraweera, 2000] Mitrovic, A. & Suraweera, P. Evaluating an Animated Pedagogical Agent. *In* Gauthier, G., Frasson, C. and VanLehn, K. (eds.). *Proc. of 5th International Conference on Intelligent Tutoring Systems*, Montreal 2000, Springer, pp. 73-82.

[Noah & Lloyd-Williams, 1995] Noah, S. A. & Lloyd-Williams, M. A Selective Review Of Knowledge-Based Approaches To Database Design. *Information Research: an Electronic Journal*, *1* (2), December 1995, pp. http://informationr.net/ir/1-2/paper4.html.

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

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

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

[Storey & Goldstein, 1993] Storey, V. C. & Goldstein, R. C. Knowledge-Based Approaches to Database Design. *MIS Quarterly*, *17* (1), 1993, pp. 25-46.

[Storey, et al., 1995] Storey, V. C., Thompson, C. B. & Ram, S. Understanding database design expertise. *Data & Knowledge Engineering*, *16* (1), July 1995, pp. 97-124.

[Suraweera & Mitrovic, 2001] Suraweera, P. & Mitrovic, A. Designing an Intelligent Tutoring System for Database Modelling. *In* Smith, M. J. and Salvendy, G. (eds.). *Proc. of 9th International Conference on Human-Computer Interaction (HCII 2001)*, New Orleans, LA, August 2001, vol. 2, pp. 745-749.

[Tauzovic, 1989] Tauzovic, B. An Expert System for Conceptual Data Modelling. *In Proc. of 8th International Conference on the Entity Relationship Approach*, Toronto, Ontario, October 1989, pp. 329-344.

[Towns, et al., 1998] Towns, S. G., Callaway, C. B., Voerman, J. L. & Lester, J. C. Coherent gestures, locomotion and speech in life-like pedagogical agents. *In Proc. of Fourth International Conference on Intelligent User Interfaces*, San Francisco, January 1998, pp. 13-20.

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

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

[Woolf, 1992] Woolf, B. P. AI in Education. *Encyclopaedia of AI*, 1992, pp. 434-444.