A STUDENT MODEL FOR OBJECT-ORIENTED DESIGN AND PROGRAMMING

Fang Wei, Sally H. Moritz, Shahida M. Parvez and Glenn D. Blank
CSE Department
19 Packard Lab
Lehigh University
Bethlehem, PA 18015
(610)-758-4867, (610)-758-4605
faw2@lehigh.edu, sgh2@lehigh.edu, smp9@lehigh.edu, gdb0@lehigh.edu

ABSTRACT

"Objects-first" is an increasingly popular strategy for teaching object-oriented programming by introducing the concepts of objects, classes, and instances before procedural elements of a programming language. Learning object-oriented design and programming is a challenging task for many beginning students. We represent CIMEL ITS, which is an intelligent tutoring system that provides one-on-one tutoring to help beginners learn object-oriented analysis and design, using elements of UML before implementing any code. We also present a three-layered Student Model which supports adaptive tutoring by inferring the problem-specific knowledge state from student solutions, the historical knowledge state of the student and cognitive reasons about why the student makes an error.

1. MOTIVATION

Learning object-oriented programming is a challenging task for many beginning students, let alone object-oriented design. Research by Ratcliffe [19] has shown that lack of comprehension expressed by first year computer science students is a rising concern in academia. McCracken et al. [14] performed a study that suggested that in UK and USA, approximately 30% of students do not understand the basics.

The first few lessons in object-orientation are rich and complex, so that many students get confused, and may withdraw from the course. Many students continue repeating similar errors after teachers tell them the right answers. They often struggle to solve problems after an instructor explains what they need to know. Meanwhile students having difficulties may not want to admit they are having problems or may have difficulties explaining their problems to an instructor. These situations happen for many reasons: 1) It is difficult to explain the problem from the student's perspective—one must understand what the student knows and doesn't know. 2) It is hard to know about let alone overcome preconceived ideas of many students. For example, some students come into the course with experiences in a procedural programming language, such as BASIC, which may actually inhibit learning object-

oriented design and programming. 3) It is hard to trace how many times a student commits similar errors and so observe repeating problem solving patterns. 4) It is hard to remedy a student's deficient problem solving patterns and encourage sound ones—people sometimes refuse to abandon their stubborn ways. 5) It is hard to take all common errors and learned knowledge into account to analyze each individual student. 6) An instructor may not know who is having difficulties until it is too late, may not be able to tell why the student is having these difficulties, may not be able to convince the student to seek help, and may simply not have enough time to look into every student's needs in a large class.

Reiser et al. [20] reported that students working with private tutors can learn given material four times faster than students who attend traditional classroom lectures, study textbooks and work on homework alone. Bloom [6] also reported that students have a better grasp of material working with a private tutor than attending traditional classroom lectures. When a qualified private human tutor is not available, the next best option is an intelligent tutoring system. Anderson and Skwarecki [3] reported that an ITS is a cost-effective means of one-on-one tutoring that provides novices with the individualized attention needed to overcome learning difficulties. Intelligent tutoring systems are not only being used in academia to augment classroom teaching but have also penetrated various industries where companies are using ITSs to train employees to perform their job functions.

As a result, ITSs have been built for various domains such as medicine, engineering, public services, computer science, etc. The application of ITS in computer science has been limited to tutoring database design and specific procedural aspects of programming languages such as Java, C++, LISP and Pascal, and have not kept up with the current technology focus of object-oriented design and programming[21][15][13][20][23].

To help students learn object-oriented design and programming we present CIMEL ITS, an intelligent tutoring system that provides one-on-one tutoring to help beginners with various learning styles in a CS1 course. The ITS supports a design-first curriculum, which subsumes an objects-first pedagogical approach. This paper focuses particularly on a three-layered Student Model in the CIMEL ITS framework which provides adaptive tutoring by inferring the problem-specific knowledge state from student solutions, the historical knowledge state of the student and cognitive reasons that the student makes an error.

2. A "DESIGN-FIRST" CURRICULUM

"Objects-first" is an increasingly popular strategy for teaching object-oriented programming, by introducing the concepts of objects, classes, and instances before procedural elements of a programming language [12]. It exposes students to the concepts of objects, instances and classes early in the course, using integrated development environments (IDEs) such as BlueJ and DrJava, which help students practice using objects and easily visualize the results of their work.

While objects-first is effective in teaching key concepts of object-oriented programming, it does not go far enough in helping students learn problem solving skills. We have developed a curriculum that emphasizes "design-first." Students still learn object concepts very early in the curriculum, but they also learn elements of UML to help them think about a problem before coding. They are introduced to programming from a software engineering point of view, and are invited to compare the development process to building a house or car.

While these skills are important in teaching students how to solve a new problem, they also represent additional, complex concepts that students need to learn. We have built new tools to help. One of those tools is CS1 multimedia courseware developed by the CIMEL (Collaborative, Constructive, Inquiry-based Multimedia E-Learning) project. CIMEL multimedia introduces the breadth of computer science, including introductory concepts in Java and object-oriented programming, complementing a textbook, *The Universal Computer: Introducing Computer Science with Multimedia* [4]. It uses many techniques including audio, video, text, animation, JUST THE FACTS summaries, and interactive, constructive and inquiry-based learning exercises to reach students with a wide variety of learning styles (a web-based demo and documentation is available at www.cse.lehigh.edu/~cimel). A comparison of students who used the CIMEL multimedia and BlueJ with students who used Barnes and Kölling's textbook and BlueJ showed the students learned much better with multimedia [5].

The IDE which students use in the course must also support the curriculum. It should be easy to learn, allow for quick experimentation, and provide an interface in which students can design as well as code their solutions. We chose Eclipse, an open-source IDE widely used by developers in both academia and industry. We also added two "plug-ins," or extensions to the basic environment: Omondo UML, which allows students to enter class diagrams and generate code stubs from their design, and DrJava, a beginner's IDE developed at Rice University [2], which provides an interactive interface in which students can enter and execute code one line at a time.

However, the IDE merely provides the environment. It does not offer students assistance while they are working on a problem. While providing a human tutor to every student who needs one is not practical, an intelligent tutoring system (ITS) that offers customized help within the IDE could fill the gap. We therefore propose CIMEL ITS, an ITS that interfaces with both the Eclipse IDE and CIMEL to give students customized, timely assistance as they are studying the material and applying their knowledge to specific problem solving tasks.

The rest of this paper will describe the overall architecture of the CIMEL ITS and then focus on the Student Model. The Student Model tracks the student's work and develops a picture of the student's knowledge state. An accurate model of what the student knows and doesn't know is crucial in presenting instruction tailored to the student's needs. Determining this knowledge state requires analyzing the student's behaviors on multiple levels.

3. CIMEL ITS ARCHITECTURE

Hartley and Sleeman [10] proposed three components of an ITS: an expert module which contains the domain knowledge of the system, a student module which models student knowledge and behavior, and a pedagogical module which chooses appropriate teaching strategies. These three components work within the framework of a user interface that presents content and interacts with the learner [8]. From this starting point, we developed an architecture for CIMEL ITS, shown in figure 1.

There are four components which comprise the CIMEL ITS itself. The Curriculum Model, at the heart of the architecture, represents the knowledge of the first few lessons in a design-first CS1 course. It is organized as a Curriculum Information Network, or CIN, which links concepts together to show relationships between them. For example, a concept may be identified as having one or more prerequisite concepts, and it may also be a component of another, higher-level

concept. A difficulty measure is also assigned to each concept within the CIN. The Expert Evaluator, the Student Model and the Pedagogical Agent in the CIMEL ITS refer to the CIN to tie the student's learning activities and state of knowledge to the curriculum.

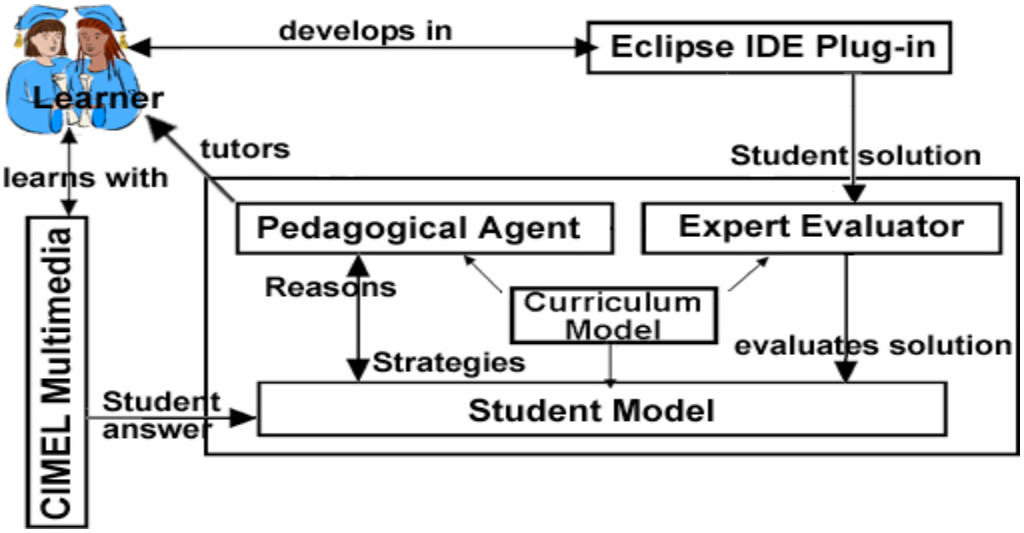

Figure 1. Architecture of CIMEL ITS

The Expert Evaluator interfaces with the Eclipse IDE through a plug-in. It observes the student's work step by step in both the object diagram interface and the code interface of Eclipse, and compares each step to its own solution(s) to the current problem. When a specific error is identified, it is linked to a concept within the CIN, and along with the recommended solution, is passed to the Student Model.

The Student Model maintains a model of the student's current knowledge state based on information from both CIMEL and the Expert Evaluator. From CIMEL, the Student Model gets input on individual student performance based on exercise and quiz data from the object-oriented contents. From the Expert Evaluator, it receives information on both errors made by the student (as described above), and problems which the student completes successfully. The Student Model then performs a diagnosis based on the history of the student's performance to determine the reasons for the student's errors and where there are gaps in his or her knowledge.

The Pedagogical Agent provides feedback to the student and tutoring when he needs help. It consists of a feedback network and tutoring strategies which may be represented by distinct agents. The feedback network is similar to CIN; it also contains feedback for each concept in the domain knowledge. Feedback is assigned a numerical level indicating if the feedback is basic or advanced. For example, the feedback consisting of concept definitions will be assigned level 1. The tutoring strategies under consideration are the traditional tutoring strategy in which an agent plays the role of a tutor and variations of cooperative learning strategies [7]. The "learning by disturbing" strategy has a traditional tutor agent and a companion agent that attempts to test the student's knowledge by misleading him [1]. The "learning by teaching" strategy also has a traditional tutor agent and a companion agent who learns along with the student [17].

When the Student Model indicates that the student needs help, the Pedagogical Agent selects the tutoring strategy and the feedback based on the student profile maintained in the Student Model. Next, the Pedagogical Agent interacts with the

student to provide feedback / tutoring. Depending upon the strategy chosen, the student might interact with one or more agents. The traditional tutoring model consists of a single agent in the role of tutor while other strategies require one agent as a tutor and another as a companion / adversary. The feedback consists of explaining the basic concepts and relationships between various concepts, and referring the student to multimedia, relevant websites and ultimately to a human tutor.

The CIMEL ITS can interact with students either through CIMEL multimedia or the Eclipse IDE, each of which initiate different flows of control through the ITS architecture. The student can learn about object-oriented design and Java programming from CIMEL multimedia. CIMEL records the student's behaviors on the quizzes and exercises into a database. The Student Model uses this data to infer the student's level of knowledge. When the student has difficulties within CIMEL, the Pedagogical Agent[s] may intervene. Possible tutoring actions include a brief explanation of concepts that the student missed or a menu of materials that the student should review. Another possibility is walking the student through a multimedia exercise step by step, making sure the student understands how to do it and highlighting the important concepts. The Pedagogical Agent updates the Student Model with information about the instruction provided.

Another flow of control begins with the Eclipse IDE. The Expert Evaluator observes the student as he enters his design or programming solution, converts the student's solution to a representation language and compares the expert's solution with that of the student and provides the results to the Student Model. The Expert Evaluator assesses what are right answers and errors in the student's solution and the concepts tied to them. After receiving the input from the Expert Evaluator the Student Model performs a diagnosis based on the input and the history of students' records and provides the diagnosis results to the Pedagogical Agent. The Student Model considers what the current state of the student is (struggling or not), what the student needs to know and why the student made those errors. The Pedagogical Agent gives proper instructions to the student based on the diagnosis results and the history of the instructions given to the student from the Student Model. The Expert Evaluator, Student Model and Pedagogical Agent each consult with Curriculum Model to perform their work. From the learner's perspective, if the learner requests help, or appears to be struggling, the Pedagogical Agent may intervene based on both the student's current behavior and his previous history as represented in the Student Model. On successful completion of the assignment, the student receives positive feedback from the Pedagogical Agent.

4. THREE-LAYERED STUDENT MODEL ARCHITECTURE

The Student Model maintains a model of a student's current knowledge state which allows more intelligent pedagogical decisions and actions to happen. After the Expert Evaluator figures out what is wrong in the student's solution the Student Model figures out why those errors are made, which is essential for the Pedagogical Agent to choose the proper instruction action. The Student Model provides fundamental information to understand the specific way the student tries to solve the problem. Student Models have been studied since the beginning of ITS research. Many researchers argue that the main purpose of a Student Model is to guide pedagogical decision-making. Ohlsson [16] called the student modeling problem "cognitive diagnosis."

To help students learn object-oriented design and programming we present a three-layered Student Model which provides adaptive tutoring to each student. The architecture of the Student Model is shown in Figure 2. The Problem-domain Knowledge Model infers how well the student understands relevant concepts, from student solutions annotated by the Expert Evaluator. The Knowledge Model infers the historical knowledge state of the student from a sequence of student solutions to the same problem or multiple problems. Finally, the Cognitive Model infers more general problem solving patterns and antipatterns from student work and errors. The three levels can then provide different kinds of information to the Pedagogical Agent about where the student needs help.

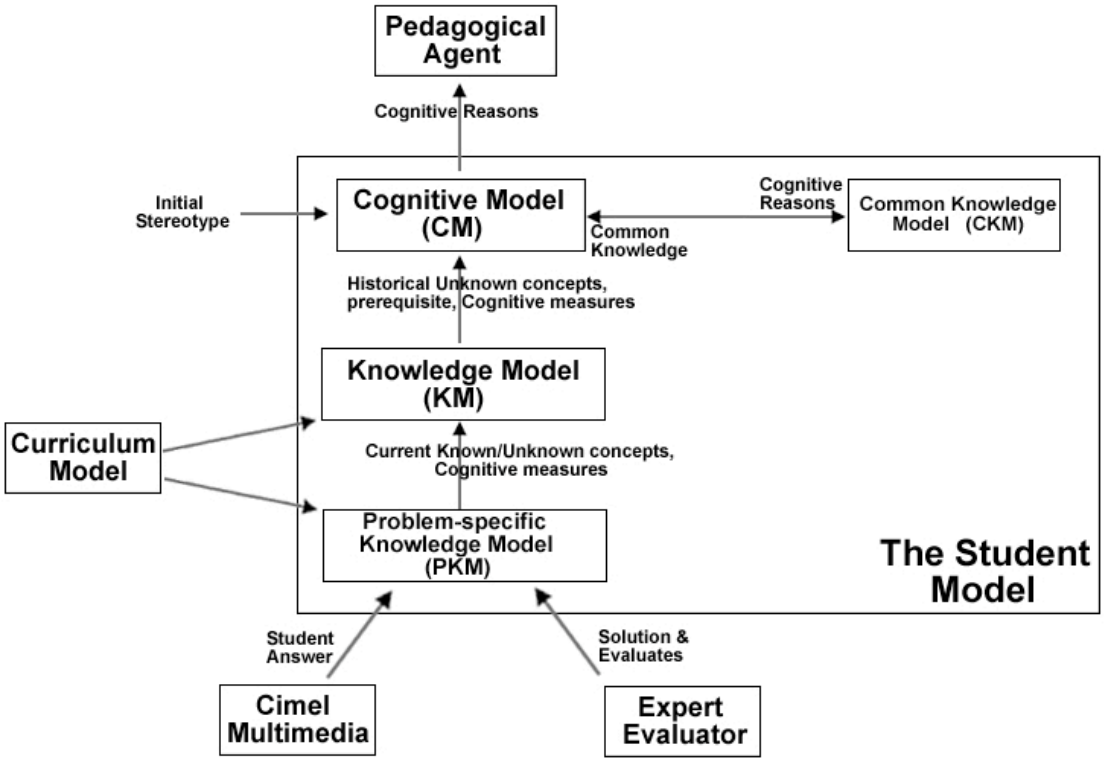

Figure 2. Architecture of the Three-Layered Student Model in CIMEL ITS

4.1 The Cognitive Model

The Cognitive Model (CM) recognizes problem solving patterns that the student is using. During the past 40 years many Student Models attempted to select the appropriate level of advice and explanations, determine readiness for advancement and dynamic planning of the student's curriculum, and give the student feedback on his or her current performance and progress through the curriculum [11]. Few Student Models have incorporated a CM into their approach to find possible reasons of errors or characterize the problem solving patterns that the student is using. Mitrovic et al. [15] argued that feedback with only the correct answer would be sufficient to help the student when he makes an error. But quite often when the student sees the right answer he doesn't know what it means and why it is correct, especially in object-oriented design and programming. So the student's problems are likely to recur. Tu et al. [22] used a mapping technique to map the reason with a specific error and maintained an information map which had all the hard coded mappings. This approach doesn't fit problems in which the reason of an error relates to the wide variety of contexts in which the error is.

Gürer [9] incorporated three CMs to analyze a student's problem solving performance into her Student Model for tutoring physics problems. The experimental results show that her CMs are helpful in diagnosing the student's solution. The "Knowledge type" model determines whether the student is using preconceived notions or actual physics knowledge. The "Focus" model determines whether the student only focuses on the problem's surface features or on physics principles. Finally, the "Approach" model determines whether the student uses a top-down approach or bottom-up approach. Physics problems are more procedurally oriented than object-oriented problem solving. Solving techniques for physics problems involve finding useful equations for given facts and deducting new facts underneath the given facts. Hence the "Focus" model is not pertinent to infer useful cognitive reasons of novices' errors in object-oriented problems because it is a procedural problem solving pattern. We observed that students who have past experience with procedural programming such as BASIC often call a method without putting object name and dot operator before the method's name. The reason is the student incorrectly applied procedural techniques he learned in a prior experience to the new object-oriented domain, which makes it harder remember object-oriented concepts. So the "Knowledge type" model is applicable to object-oriented design and programming. The "Approach" model can also be used in object-orientation because the top-down decomposition is a rational problem solving strategy.

The CM can consist of problem solving patterns/antipatterns. As in the design patterns literature, experts know that there are patterns of problem solving that are effective, high-quality solutions to recurring problems while antipatterns produce ineffective, low-quality solutions or none at all. Patterns are often non-obvious to beginners; antipatterns may often seem more obvious but turn out to be misleading blind alleys. We would like the problem solving patterns of a student to be sound which means the student can solve the object-oriented problems correctly and efficiently because of the problem solving patterns he is using. But students often use antipatterns of problem solving which causes inefficient and even wrong solutions.

"Hacking" is a common antipattern of novices. "Hacking" has three forms: "No Design," "Not Finish Design," and "Not Apply Design." They determine whether the student starts to write code without doing any design, without finishing design and not applying the design he made respectively. We observed the three forms of "Hacking" quite often among novice students when they first solve object-oriented problems. We incorporate "Hacking" in the CM to emphasize that students must finish design completely first and then do coding from the design.

"Analogy" is useful in general problem solving, and especially object-oriented design. The analogy pattern has three cases: "If Analogy," "Right Analogy," and "Analogy Adaptation." "If Analogy" determines whether the student used analogy by copying existing design or code of classes in his solution. "Right Analogy" determines whether the student makes appropriate analogies between classes. Only similar classes can be candidates for analogy. "Analogy Adaptation" determines whether the student adapts the copied design or code into the proper class. For example there are three similar classes, Circle, Square and Triangle in project "Shape." The Square Class has a draw method that draws a square object on a canvas. Students are requested to create a draw method for the Circle Class. The right pattern for students is to copy the draw method in the Square Class instead of coding from scratch.

We incorporate "Knowledge type", "Approach", "Hacking", and "Analogy" models, into the CM. They represent and diagnose different problem solving patterns and antipatterns for object-oriented design and programming problems. The CM enables the Pedagogical Agent to remedy a student's deficient problem solving antipatterns and encourage sound ones.

4.2 Other Knowledge Models

The Problem-specific Knowledge Model (PKM) maintains a student's knowledge state and his solution for each question. A question can be an exercise or a quiz in the CIMEL multimedia or a programming exercise in the Eclipse IDE. The PKM specifies how the student's knowledge state for the question is tied to CIN (Curriculum Information Network) in terms of concepts he knows and doesn't know. Each concept is associated with a probabilistic value to show how well the student knows the concept and a time stamp to show when the learning action about the concept happens. The PKM helps to maintain a history of when, what and how well the student learns on each question. The history enables tracing how many times the student commits similar errors and figuring out the student's repeating problem solving patterns. PKM uses a probabilistic Bayesian network to represent how well student knows the CIN concepts.

The Knowledge Model (KM) maintains a history of changes in the student's general conceptual knowledge. It generalizes an overall student's knowledge state from what observed in the PKM, by tying it to CIN in terms of *concepts* he knows and doesn't know. The KM helps to maintain a history of learning effect of the student. This history enables explaining the problem from the student's perspective by understanding what the student knows and doesn't know. We can also get a learning effect/time curve of each student to understand the difficulties students are having for future research purposes.

The Common Knowledge Model (CKM) maintains a common knowledge state inferred from *all* students. The common knowledge state is a cross reference between common errors and concepts, problem-solving strategies or other reasons that lead to these errors. The CKM can provide hints to the CM, which are the popular reasons that tied to the same error made previously by other students.

4.3 Flow of Control in the Student Model

The PKM receives student performance data from the CIMEL multimedia and the Expert Evaluator in the Eclipse IDE. In CIMEL multimedia the student performance data consists of student's performance on each quiz or exercise. In the Eclipse IDE the student performance data includes problem presented to the student, the student's solution, the expert's solution, the CIN concepts used to create the expert solution, and the gaps between the expert and student solutions as identified by the expert evaluator. From the data the PKM infers concepts for each question the student learns and how well he knows them. Each concept in the inferred results has a time stamp and a probabilistic value and associates to one question. All concepts are organized according to the CIN. PKM also measures cognitive measurement flags such as IfDesignFirst for the "No Design" antipattern. When the student starts to write code without doing any design in Eclipse, IfDesignFirst is set to 1. Cognitive measurement flags also associate with a time stamp. The problem-specific known/unknown concepts along with the cognitive measurement flags are sent to the KM.

The KM calculates how well the student knows all concepts in the CIN from the input of the PKM. Each concept in the result has a probabilistic value and a time stamp. These concepts comprise the student's knowledge state along his learning history. If we find a prerequisite concept with the newest time stamp for the missing concepts in the student's solution has very low probabilistic value, the possible reason for the student's error can be the student doesn't know the prerequisite. The KM records the cognitive measurement flags from the PKM to keep track of the progress on students' problem solving strategies. The unknown prerequisites and current cognitive measurement flags are sent to the CM to let it infer the reasons for the student's error.

The CM has three sources of input, initial stereotype [18], "unknown" prerequisites, and popular reasons among students for the error in the solution along the history. The initial stereotype is (a) initial knowledge level (novice, beginner, intermediate and advanced) (b) computing language experience (such as Basic). The Cognitive Model directly obtains the initial stereotype from the student when he first accesses CIMEL ITS. The "unknown" prerequisites and the cognitive measurement flags are from the Knowledge Model. The popular reasons among students for the error along the history are from the CKM. The CM synthesizes and analyzes all the inputs to get cognitive reasons of the error. These reasons are incorporated into the CKM finally.

4.4 A Concrete Example

The first exercise in which students work with objects presents a group of classes which define shapes: circle, square, and triangle. The class diagram is given in the EclipseUML plug-in. Students create instances of these shapes, which they see drawn on a canvas. They then manipulate the shapes by calling methods to move them or change their size or color. Each exercise specifies exactly what the student is to do. For example, an exercise may require the student to create a square, change its color to magenta, and move it 100 pixels right and 20 pixels down from its initial location.

Suppose the student is working on the second step of this problem: changing the square's color to magenta. To formulate a correct solution, the student would observe that there is a method called changeColor, which has one parameter, a String that specifies the new color. Next, the student must call changeColor for the instance of Square he has already created. The complete correct code would be:

Square s = new Square();
s.changeColor ("magenta");

Now, suppose the student enters this solution:

Square s = new Square();
changeColor("magenta");

The Expert Evaluator would recognize that the second line is in error, and would pass this information to the Student Model:

Problem presented to the student;
Student solution: changeColor("magenta");
Correct solution: s.changeColor("magenta");
CIN clause: Comp(invokeMethod: nameObject, period, methodName, parameter-list)
Missing: nameObject, period

The Student Model interprets what this error tells us about the student. Why did the student make this particular mistake? The reason could be any one of these:

1. The student made a typographical error.
2. The student doesn't remember the syntax of calling methods.
3. The student doesn't understand the concept that non-static methods are called for a specific instance of a class.
4. The student has misconception that he is programming in a procedural language such as BASIC.
5. The student doesn't understand that methods are how object behaviors are implemented in Java (a deeper concept than #3 – WHY non-static methods are called for a specific instance).

These five reasons reflect five different levels of understanding. The Student Model looks at the student's performance so far for clues as to which reason is most likely the actual cause of the error. How many similar exercises has the student already completed? What was his performance on those? What related concepts has the student already studied, and how deeply do we believe the student understands them?

Let us suppose that the student has completed three other exercises which included the task of calling a method for an instance. In those exercises, the student committed similar errors twice. The student was given reminders on the syntax of calling a method, and in one case was given a brief review of the concept of manipulating individual instances through calling methods. The student had viewed the section in CIMEL related to calling methods, and did well on the exercises at that time. The PKM (Problem-specific Knowledge Model) reflects a probability that the student understands the concept. However, if this is the third time the student has made this same error, and was recently given a brief review of the concept, the KM decides that the student probably does not understand the prerequisite concepts of calling methods.

If a student continually tries to call a method without an instance it may be because the student has the misconception that he is programming in a procedural language such as BASIC. Students having prior experience with a procedural programming language tend to have the misconception which is making it more difficult for him to grasp object concepts. The Cognitive Model's "Knowledge type" pattern infers that a misconception is the main reason that the student keeps repeating the error.

This decision is now used in two ways. First, the verdict, along with the information sent by the Expert Evaluator, is passed by the CM to the Pedagogical Agent. Second, it is used to update all components in the Student Model to reflect the change in our estimate of the student's knowledge of the concept. In this case, the Student Model records that the student understands the concept of calling methods and with only a low certainty. The Student Model also records the error, along with general information about the exercise, in a history of the student's completed work.

5. CONCLUSION AND FUTURE WORK

Learning object-oriented design and programming is challenging for novices. Many students have learning difficulties which cannot be entirely solved by teachers. To help student learning, we have designed an ITS which works through both the CIMEL multimedia and the Eclipse IDE. The Student Model will help the ITS determine which concepts and operations a student understands so that he or she can apply them in design and programming problems. Our three-layered Student Model provides a more accurate profile of a student so that the ITS can support adaptive

tutoring. It infers the problem-specific knowledge state from a student's work, the historical knowledge state of the student, and cognitive reasons that the student makes an error. A critical component of the Student Model is the Cognitive Model. It represents and diagnoses different general problem solving patterns and antipattern, such as "Knowledge type," "Approach," "Hacking," and "Analogy", which characterize the problem solving strategies that the student is using.

We plan to refine the cognitive model from student data and continue developing details of the Student Model. We will have a working prototype of CIMEL ITS including the Student Model by the summer of 2005. This will allow us to use CIMEL ITS to augment CS1 courses in the Fall 2005 semester, and to gather experimental data on the effectiveness of CIMEL ITS and the Student Model.

6. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants No. EIA-0087977 and 0231768 and PITA (Pennsylvania Infrastructure Technology Association).

7. REFERENCES

[1] Aïmeur, E. and Frasson, C. Analyzing a New Learning Strategy According to Different Knowledge Levels. *Computers in Education*, vol. 27, no. 2, 1996, pp. 115-127.

[2] Allen, Eric, Cartwright, Robert, and Stoler, Brian. DrJava: A Lightweight Pedagogic Environment for Java. In *Proceedings of the SIGSCE Conference on Computer Science Education*, March, 2002.

[3] Anderson, J.R. and Skwarecki, E..The Automated Tutoring of Introductory Computer Programming. *Communications of the ACM*, vol. 29, September 1986, pp 842-849, ACM Press.

[4] Blank, G. D., Barnes, R. F. and Kay, E. J.. *The Universal Computer: Introducing Computer Science with Multimedia* (McGraw-Hill/Primis, 2003/2004). Sample material at www.cse.lehigh.edu/~glennb/um/ and http://cimel.cse.lehigh.edu.

[5] Blank, G. D., Pottenger, W. M., Sahasrabudhe, S. A., Li, S., Wei, F., and Odi, H. Multimedia for computer science: from CS0 to grades 7-12, *EdMedia*, Honolulu, HI, June 2003.

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

[7] Chan T. W. and Baskin A. B.. Learning Companion Systems. In *Intelligent Tutoring Systems: At the crossroads of Artificial Intelligence and Education* (Edited by Frasson, C. and Gauthier, G.), Chap 1. Ablex, N.J., 1990.

[8] Dag, F. and Erkan, K.. Realizing of Optimal Curriculum Sequences for a Web Based General Purpose Intelligent Tutoring System, *The IJCI Proceedings* (ISSN 1304-2386, vol. 1, No 1, July 2003, International XII. *Turkish Symposium of Artificial Intelligence and Neural Networks (TAINN'2003)*.

[9] Gürer, W. D. (1993) *A Bi-level Physics Student Diagnostic Utilizing Cognitive Models for an Intelligent Tutoring System,* PhD Dissertation

[10] Hartley, J.R. & Sleeman, D.H. Towards more intelligent teaching systems. *International Journal of Man-Machine Studies 2,* 1973, pp. 215-236.

[11] Katz, S., Lesgold, A, Eggan, G & Gordin, M. (1992). Modelling the student in Sherlock II. *International Journal of Artificial Intelligence in Education* 3(4):495-518.

[12] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., The BlueJ System and its Pedagogy, *Journal of Computer Science Education,* Special issue on Learning and Teaching Object Technology, vol. 13, no. 4, Dec 2003.

[13] Kumar, A.. Model-Based Reasoning for Domain Modeling in a Web-Based Intelligent Tutoring System to Help Students Learn to Debug C++ Programs, *6th International ITS Conference*, Biarritz, France and San Sebastian, Spain, June 2002.

[14] McCracken M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Yifat Kolikant, Y., Laxer, C., Thomas, L., Utting, I., Wilusz, T., A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, December 01, 2001, Canterbury, UK

[15] Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B.. Constraint-Based Tutors: A Success Story. In *Proceedings of the 14th Industrial and Engineering Applications of AI and Expert Systems Conference (IEA/AIE-2001),* Budapest, Hungary, June 2001,pp. 931-940.

[16] Ohlsson, S. (1986) *Some Principles of Intelligent Tutorinig,* Instructional Science, Vol.14, pp. 293-326.

[17] Palthepu, S., Greer, J., and McCalla, G.. Learning by Teaching. *The Proceedings of the International Conference on the Learning Sciences*, AACE, 1991.

[18] Prentzas, J., Hatzilygeroudis, I., and Garofalakis, J., A Web-Based Intelligent Tutoring System Using Hybrid Rules as Its Representational Basis. *The 6th International Conference, ITS 2002, Biarritz, France and San Sebastian, Spain, June 2002, Proceedings, pp. 119-128.*

[19] Ratcliffe, M. B.. Improving the Teaching of Introductory Programming by Assisting the Strugglers. *The 33rd ACM Technical Symposium on Computer Science Education*, Cincinnati, USA, March, 2002.

[20] Reiser, B.J., Anderson, J.R., Farrell, R.G.. Dynamic Student Modeling in an Intelligent Tutor for LISP Programming, *Proc. of the Eighth Int'l Joint Conf. on Artificial Intelligence*, pp. 8-14, Los Angeles, 1985.

[21] Sykes, E.R. and Franek, F.. A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. In *Advanced Technology for Learning,* vol. 1, no. 1, 2004.

[22] Tu, L., Hsu, W. and Wu, S.. Cognitive Student Model – An Ontological Approach. *International Conference on Computers in Education (ICCE'02) December 03 - 06, 2002, Auckland, New Zealand*

[23] Woolf, B. and McDonald, D., Human-Computer Discourse in the Design of a PASCAL Tutor*, Proceedings of Conference on Human Factors in Computing Systems*, Boston, MA, 1983.