Simulating a Learning Companion in Reciprocal Tutoring Systems

Tak-Wai Chan and Chih-Yueh Chou

Institute of Computer Science and Information Engineering, National Central University

Abstract

This paper describes how student modeling plays a role in the implementation of a cooperative learning system, in particular, the simulation of a virtual learning companion. A scaffolding tool for solving Lisp recursive problems is used to support a protocol of cooperative learning, called reciprocal tutoring. During reciprocal tutoring, two agents, where an agent is either a real student or a virtual learning companion simulated by the computer, interact and take turns for playing the roles of a tutor and a tutee. We describe a method that takes the overlay approach of student modeling to simulate a learning companion. The method constructs the behavior of the companion, illustrating its advancing knowledge and mistakes. A preliminary evaluation of the system has been conducted.

Keywords — Collaborative composition, problem solving, multi-user simulation, cooperative learning, learning by teaching, learning companion, reciprocal tutoring, student modeling.

1. Introduction

As an alternative to one-on-one intelligent tutoring systems (ITSs), Chan and Baskin (1988, 1990) proposed a three-agent model, *learning companion systems*. They suggested that the computer can be simulated as two co-existing agents: a teacher and a learning companion. The learning companion is an artificial student who interacts with the student and learns under the guidance of the computer teacher. Thus, the learning companion performs the learning task at about the same level as the student, and both the student and the companion exchange ideas while being taught by the computer teacher. Integration-Kid, a learning companion system, explores various patterns of interactions through different protocols of learning activities of the agents (or the triad), such as, cooperation, competition, modeling, and observing (Chan, 1991). The performance of the learning companion in Integration-Kid is governed by a subset of problem solving expertise and some faulty knowledge. In the process of learning, this problem solving expertise is being expanded and the faulty knowledge is being tuned by simply deleting and adding knowledge units.

Social learning systems are emerging learning environments that involve multiple agents, working at the same computer or across connected machines. These agents are either computer simulated or real human beings, taking various roles via different protocols of learning activity (Chan et al., 1992; McManus and Aiken 1993; Jehng et al., 1994). These systems are sometimes called or closely related to collaborative, distributed, or distance learning systems. In some learning activities, the artificial learning companions together with the real students on the network can form a virtual learning group (Lai, 1994).

Of particular interest to this work is a protocol of learning activity where the student "learns how to learn by teaching the learning companion" (Chan and Baskin, 1988, p.199). This inverted model of ITS, which puts the student in the position of a tutor, instead of a tutee, is termed as *learning by teaching* and elaborated by Palethepu, Greer, and McCalla (1991) who describe a system architecture in a declarative domain represented by a semantic network. The system acts as an interactive knowledge acquisition tool and asks the student questions in order to complete the inheritance hierarchies. They speculate that such a system would be useful to a student who already 'almost knows' the domain and enhance the student's meta-cognitive reasoning skills. Following such a *tabular rasa* approach, Nichols (1994) develops a system. Subjects tested the system "express discomfort at having to feed a 'knowledge-hungry' agent."

Instead of using such *pure* co-learners, VanLehn, Ohlsson, and Nason (1994) suggested that a more complex system could use covert experts and pedagogical modules to move the dialogue in pedagogically useful direction. Besides being a learning partner of the student, a simulated student can potentially be useful for formative evaluation of instructional designer's prototypes and teacher training, thus the name 'meta-tutoring' (VanLehn, Ohlsson, and Nason, 1994; VanLehn, 1993). Ur and VanLehn (1994) apply explanation-based machine learning method to simulate a physics student for the purpose of tutor training and use it to study the cognitive process of learning from a tutor.

To emphasize individualization, student modeling is usually at the core of the research of ITSs. However, there are some objections of using detailed student model because of the practical difficulty in building it. Besides, since students may have different prior knowledge, learning experience, particular interests, social background, motivation, or personality characteristics, if all were represented in the student model, it may embrace almost all the problems in cognitive science. For more discussion of the arguments and counter-arguments of using student model, see Self (1988) and McCalla (1990).

There has been hesitation of using student models in social learning environments. For example, Palethepu, Greer, and McCalla (1991) found that student modeling seemed to be less useful and put it as an interesting open question whether student modeling is critical in learning by teaching. Also, Newman (1989) views learning interaction in a social context as a successive exchange of interpretations of the actions on the tasks by different agents. An agent does not have, or apparently need, an understanding of exactly how other agents approaching the task. Quite the contrary, it is necessarily somewhat ignorant of each other's mental state. All an agent has to do is to produce some moves that in some way contribute (or can be understood as an attempt to contribute) to the task. The other agents do not have to know exactly what the agent thinks he or she is doing as long as the other agents can regard that what the agent does contributes a part to the joint accomplishment of the task. The primary requirement of an agent's utterance is whether it makes sense to the other agents. By a sense making response, it means that the response possesses some relation with the past interaction experience that represents a range of expectations as well as the specificity of the current problem context. Therefore, if one's response is unexpected to others, others can express their unexpectancy explicitly.

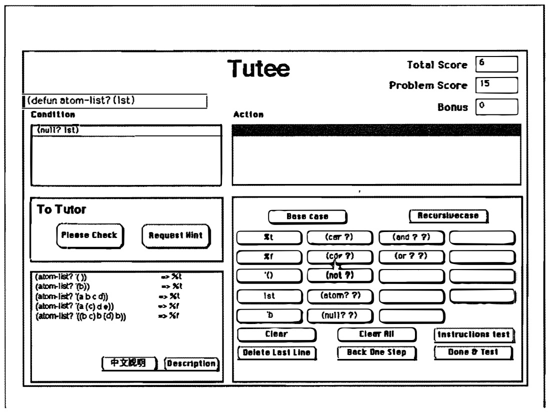
This paper discusses how to use student model to construct a cooperative learning environment and to simulate a virtual learning companion for practicing Lisp recursion. The rest of this paper is organized as follows: Following a description of the design of the reciprocal tutoring system, we deliberate the system implementation, in particular, the modeling of a learning companion. After that is a preliminary evaluation of the system, and finally, a discussion.

2. The System

Cooperative learning is a form of learning activity to lessen cognitive load by partitioning the learning task into sub-tasks undertaken by different agents. Each agent is only responsible for a sub-task while the rest is being taken care by other agents. In the domain of learning to write programs, students have to understand the problem first, then formulate the problem in the conceptual model supported by the programming language. Followed from that are the code generation and debugging where the student iteratively generates and tests code. If there is an error, the student diagnoses the code, looks for a way to fix the error, receives advice if available, then revises the code. Code generation and debugging, being decoupled programming sub-processes, can be taken on by a tutee and a tutor respectively. In general, this view suggests that tutoring in problem solving in an ITS is a cooperative process between the student and the computer, undertaking two sub-tasks: tuteeing (learning by being tutored) and tutoring, respectively. If two agents perform these two sub-tasks alternatively, then we shall obtain a protocol of cooperative learning activity called reciprocal tutoring (cf. Palincsar & Brown, 1984). Having a student play the role of a tutor (or a critic) is perhaps inevitable for cooperative learning in problem solving domains, since problem solving usually requires feedback in solution development, unless there is a human or computer tutor. Furthermore, the conceptualization of reciprocal tutoring, we believe, offers broad applicability to other knowledge domains.

Learning in the Reciprocal Tutoring System (RTS), a student plays the role of a 'tutee' whilst the computer assumes the role of a 'tutor' and they take turns for different problems. When the student plays the role of tutee, the system is a typical ITS. A student model is kept inside the system to provide hints and feedback. The student uses a nicely designed scaffolding tool (see Figure 1) which is originally a part of the Petal system (Bhuiyan, Greer, and McCalla, 1992). Following the 'calculator metaphor', this Petal-like scaffolding tool (PLS) provides a set of Lisp 'code chunks' (or Lisp expression templates) displayed as buttons on the screen. These code chunks allow the student to construct code without making syntax errors. Thus the student does not worry about the legality of the expressions and can pay more attention to the semantics and the heuristic knowledge required for getting a solution. Also, the restriction of using single 'cond' construct (the 'cond' construct in Lisp is similar to the usual 'if' construct in most programming languages, but in a structured form rather than nested 'if") implies that this construct is by itself a schema that suffices to solve a set of recursive problems. For the system development, PLS provides two advantages. First, using the code chunk buttons instead of the keyboard, PLS cuts down dramatically the space of the student's possible responses. This permits the system to concentrate on expected responses, no matter whether they are correct or incorrect. Also, when the student is restricted to use a single 'cond' construct, the number of possible correct solutions diminishes to one or two solutions only. This, in turn, affects the design of internal student modeling when considering how to model the student and the learning companion.

The student tutor can see the correct answer and every action of the computer tutee, while playing the role of a tutor (see Figure 2). When the computer tutee is working on a problem, the student tutor, who knows the answer, is watching how the tutee works. However, knowing the answer is not enough; when the computer faces difficulties or makes errors, the student, who is not necessarily more capable than the computer, may not be able to offer help. A natural way to handle this situation is that the system provides an intelligent 'super-tutor' to help the student tutor tutor the computer. This super-tutor at least has to make sure that the student tutor knows where the tutee's trouble is and is able to offer relevant help. Instead of implementing an 'active' super-tutor agent, we take an alternative approach to satisfy these objectives. We develop an intelligent tool, called the Diagnosis-Tree (DT), to be used by the student tutor in the process of tutoring.

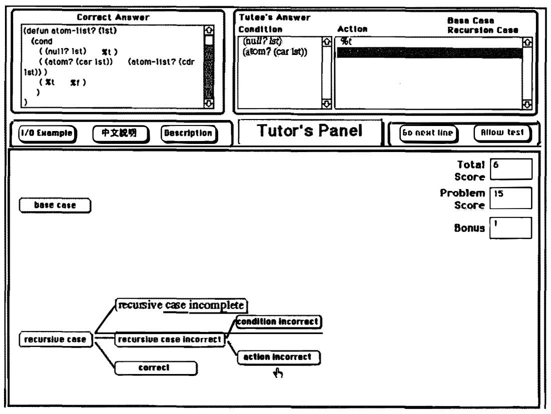


```
(defun atom-list? (lst)
(null? lst)
```

```
(atom-list? '())              => %t
(atom-list? '(b))             => %t
(atom-list? '(a b c d))       => %t
(atom-list? '(a (c) d e))     => %f
(atom-list? '((b c) b (d) b)) => %f
```

Figure 1. Tutee's Petal-Like Scaffolding Tool.

DT helps the student tutor identify where the error is and construe why it is an error When the computer tutee 'presses' the button of 'Request hint' to ask for help, the tutor does two things: locates the tutee's trouble and chooses a hint to give. The tutor expands the DT, starting from the 'base case' or 'recursive case' nodes (displayed as buttons on the screen). The tutor is required to pick up the correct path in expanding the DT in locating the error. First, the tutor has to judge whether the current clause is a base case or recursive case. The highlighted part in Figure 2 is the action part of a recursive clause. When the tutor presses the 'recursive case' button, the DT expands and displays three buttons - 'recursive case incomplete', 'recursive case incorrect', and 'correct'. If the tutor selects 'recursive case incorrect' button, the DT will further expand and generate two buttons - 'condition incorrect' and 'action incorrect'. Nodes will not expand if the tutor hits the wrong button. When the tutor has successfully spotted the student's trouble at the end of the diagnosis path expanded (in this case, the 'action incorrect' button), a number of hints in Chinese (can be easily anglicized, of course) are available (see Figure 3). These hints explain errors, suggest possible approaches or Lisp functions to be used, or offer answers directly. The tutor will pick a hint and send it to the tutee. If 'more hint' is requested by the tutee, the tutor will send another one. A point system is used to make sure that the dyad will not depend too much on the DT and will pay more attention to each other throughout the process. Points will be scored or deducted according to the performance of the tutor in locating the tutee's errors and there will be penalty of using hints that are too close to the answers.

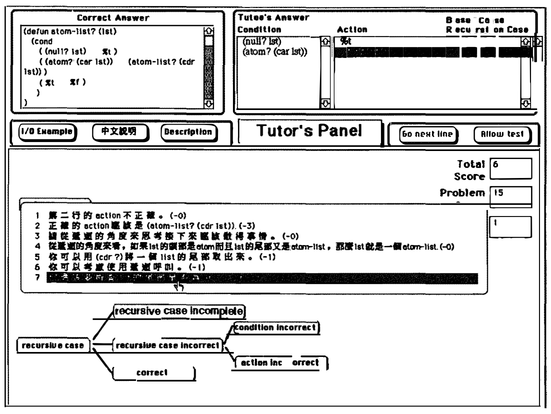


```
(defun atom-list? (lst)
  (cond
    ( (null? lst)    %t )
    ( (atom? (car lst))    (atom-list? (cdr
lst)) )
    ( %t    %f )
  )
)
```

```
(null? lst)
(atom? (car lst))
```

```
%t
```

Figure 2. Student Tutor's Interface (I).



```
(defun atom-list? (lst)
  (cond
    ( (null? lst)    %t )
    ( (atom? (car lst))    (atom-list? (cdr
lst)) )
    ( %t    %f )
  )
)
```

```
(null? lst)
(atom? (car lst))
```

```
%t
```

1 第二行的 action 不正確。(-0)
2 正確的 action 應該是 (atom-list? (cdr lst)). (-3)
3 請從遞迴的角度來思考接下來應該做得事情。(-0)
4 從遞迴的角度來看，如果lst的頭部是atom而且lst的尾部又是atom-list，那麼lst就是一個atom-list. (-0)
5 你可以用 (cdr ?)將一個 list 的尾部取出來。(-1)
6 你可以考慮使用遞迴呼叫。(-1)
7 [illegible]

Figure 3. Student Tutor's Interface (II).

3. Simulating the Learning Companion

Reciprocal tutoring system is implemented with SuperCard and Lisp on MacQuadra and PowerMac, using AppleEvent to link the two application programs. A small Lisp interpreter written in Lisp is used for testing student's program and the student may use it to verify the properties of the Lisp language constructs.

3.1. Modeling Tutor

To simulate a tutor, like ITS, we implement a student model, and use that model to build the DT. We compare the student's solution with a discrimination net, a simple artificial intelligence technique, to keep track the student's actions. For each problem, based on the correct solution and the error data collected for this problem (in a separate experiment where students used PLS), we set up a discrimination net. Each node in the net represents a possible situation of the student's program and stores different levels of hints and suggestions. When the student requests hint, the student's program is matched with the net. On termination at a certain node, the current situation of the student's program is recognized and the associated hints are then available for the tutor to deliver. Using the schema (a single 'cond') constrained by PLS, the problem's latitude has been narrowed to one or two possible correct solutions, so the discrimination net is rather linear. The disadvantages of this approach is that each problem needs a discrimination net. The size of each net is rather large and is thus labor intensive to build. Nevertheless, this student model suffices to provide the computer the ability to point out the tutee's errors and supply hints about how and what to do next in solving a problem.

3.2. Modeling Tutee

To simulate a tutee, we have to model the behavior of a learner whose knowledge of the domain appears to be advancing, even making mistakes from time to time. The tutee's response may be incorrect. The tutee sometimes even may not be able to give a response and asks for help. Thus, two objectives have to be fulfilled: able to model the evolving knowledge of the tutee and be 'psychologically plausible' to the student tutor. By psychologically plausible, we mean that the learning behavior and performance of the tutee can be conceived by the student. Instead of applying machine learning methods to empower the learning ability of the tutee, we take an overlay approach to make the tutee pretend to learn at about the level of an average student.

We take an example to illustrate how the simulated tutee works. Let us consider the solution of a simple recursion that finds the length of a list:

```
(defun length (lst)
    (cond   ((null? lst)        0)
            (%t       (+ 1 (length (cdr lst))))))
```

If the tutee knows how to do this problem, then using the code chunk buttons on PLS interface, the tutee has only to construct '(null? lst)' and '0' for the first conditional clause and '%t' and '(+ 1 (length (cdr lst)))' for the second. For '(null? lst)', it is composed of two code chunks, '(null? ?)' and 'lst'. The tutee is able to use the code chunk '(null? ?)' means that the tutee understands the semantics of 'null?' (syntax has already been taken care by PLS) and knows that it is applied to the base case of this problem. So the use of '(null? ?)' in this problem is associated with two *concepts*: 'null?' and 'base-case'.

Now, we may assign a number from 0 to 1 for each concept to represent the tutee's proficiency of that concept. For example, if the tutee is very good at understanding 'null?', but not 'base-case', then we may assign the proficiency value of 'null?' to be 0.7 and that of 'base-case' to be 0.4. Knowing the proficiency value of the associated concepts of '(null? ?)', 'null?' and 'base-case', we take their minimum value, that is, 0.4, to be the proficiency value of the code chunk. So if a proficiency value of a code chunk is not high, it means that there is a deficiency of understanding this code chunk due to at least one of the associated concepts. Between 0 and 1, we may use four intervals to represent the grade of a code chunk: [0, 0.3), [0.3, 0.5), [0.5, 0.7), and [0.7, 1] for poor, fair, good, and excellent, respectively. Thus, the grade of '(null? ?)' is fair. Now suppose that the tutee is in the situation to use the code chunk '(null? ?)', then since the grade is fair, the tutee may ask the human tutor for hints. If the first hint is too general, then the tutee will ask for a second hint. If unfortunately, the second hint is related to the concept 'null?' that the tutee understand well, then the tutee will ask for another hint until there is a hint which is related to the concept 'base-case'.

For each problem, the simulation of the tutee will refer five components of the system: (i) a *tutee model*, (ii) a decomposed solution, (iii) the tutor's hints, (iv) the error code base, and (v) a set of response rules. The tutee model is a set of all the concepts of the domain and their proficiency values. The concepts range from simple such as 'null?' and 'car' to complex such as 'deep-recursion'. Thus the tutee model records the tutee's degree of understanding of the domain and hence its competence.

A decomposed solution is a list of code chunks decomposed from a solution of the problem. Each code chunk is associated with the required concepts. For example, a decomposed solution of the 'length' problem is [('(null? ?)'; 'null?', 'base-case'), ('lst'; 'null?-arg', 'atom'), ...]. Note that, these required concepts are context dependent, that is, different solution contexts may require different required concepts for the same code chunk. Also, there are no proficiency values of the concepts or the code chunks in the decomposed solution.

A tutor's hint, though in natural language form, can be analyzed to determine whether it indicates a concept of that code chunk. The error code base is formed by collecting mistakes such as the mixing up of 'car' and 'cdr', 'and' and 'or', '%t' and '%f', and error data from previous empirical experiments.

Response rules are heuristic rules that govern the responses of the tutee. There are four rules based on the computed grade of a code chunk and one for checking when the tutee finishes a condition clause:

If the grade of the code chunk is excellent, the tutee will display the correct code chunk.

If the grade of the code chunk is good, the tutee will display the correct code chunk, but the responding time is longer than when the grade is excellent.

If the grade is fair, there will be 25% possibility that the solution is correct and the responding time is longer than when the grade is good. The other 75% possibility is to respond with incorrect code chunk or ask for help.

If the grade is poor, the response is either showing incorrect code chunk or asking for help.

If the tutor completes a condition clause, the tutee will ask the tutor to check the clause.

At the beginning, the initial proficiency values of the concepts in the tutee model are taken by trial and error method through testing the behavior of the tutee to see whether they are reasonable for an average novice learner. Figure 4 depicts the data flow between different components in simulating the tutee and the algorithm works as follows:

(I) From the decomposed solution (a list of code chunks), get the next code chunk and compute the grade of this code chunk as follows: from the tutee model, retrieve the proficiency values of all the required concepts associated to that code chunk, get the minimum of these values and transform it as the grade of the code chunk. If there is no more code chunk to be tried in the decomposed solution (problem has done), terminate.

(II) Use the response rules to determine what the tutee would do. Four possible responses:

(IIa) Display the code chunk (correct response), then go to (I).

(IIb) Ask for help. Keep asking for hint until the hint indicates to use the required concept of lowest proficiency value of the current code chunk. Display the code chunk and update the proficiency value of that concept, say, add 0.05, in the tutee model, then go to (I).

(IIc) Make mistakes. Get an error datum from the error code base to display, then go to (I).

(IId) Ask tutor for checking. The tutor uses the DT to check. If the clause is correct, the tutor notifies the computer tutee to continue, then go to (I). Otherwise, the tutor points out the mistaken code chunk and then the tutee has to backtrack to it and re-try the whole clause, then go to (I).

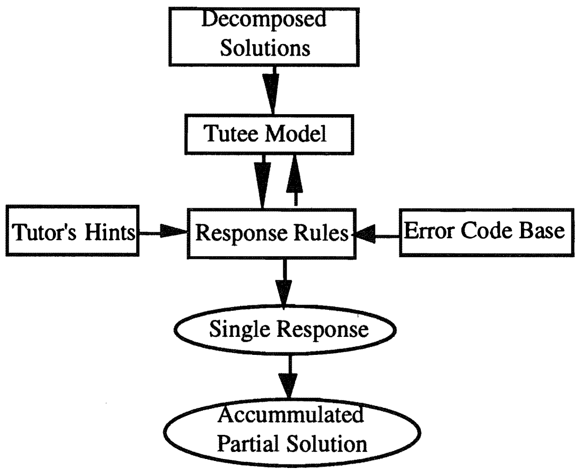


Figure 4. Data Flow Between Components in Simulating a Tutee.

Note that the consistently advancing competence of the tutee is due to the decoupling of the tutee model from the data (decomposed solutions and the error data base) to construct the responses. Moreover, this decoupling allows different problems to be independent to the tutee's behavior and thus new problems can be easily added into the system. Also, notice that the transferring of the proficiency knowledge from the student tutor's hints to the computer tutee is in a rather straight forward manner. Finally, if a problem has more than one solution, the system will choose one of the decomposed solutions for reference.

4. Preliminary Evaluation

As previous description, Reciprocal Tutoring System (RTS) enables the student and the computer to switch the roles of a tutor and a tutee, thus, if there is no role reversal, it essentially consists of two sub-systems:

Computer Tutoring system (CTS) and Human Tutoring system (HTS). With these two sub-systems, we can study the sub-tasks, tutoring and 'tuteeing' (working on the task with the help of a tutor), separately. Together with the PLS (no hints provided), we shall compare the learning effects on students when using RTS, CTS, HTS, and PLS in our evaluation.

Twenty two freshmen who learned basic Lisp concepts before used the systems to practice Lisp recursion problems for two hours. Their mid-term examination was taken as a pretest and students were distributed evenly in different systems according to their pretest result. Each system version has 5 or 6 subjects to use. Then all took a posttest with Lisp language interpreter. Table 1 is the average scores of the students and their standard derivations in the posttest.

Table 1. Posttest Scores After Using Different Systems.

Systems	RTS	CTS	HTS	PLS
aver. / std. der.	78 / 4.79	77 / 13.9	64 / 9.7	70 / 15.2

Given the limited number of subjects per system, the result of these preliminary trials are not significant and can only provide us a picture of how to draw hypotheses about reciprocal tutoring. All PLS students like the system and think that they can focus better without caring about the syntax and it is convenient to have other facilities, such as checking the solution with examples. The low performance of PLS is apparently due to the fact that many PLS students could not proceed to other problems within two hours once they had trouble with one or two of the problems, because of the lacking of the hints and the needed feedback. Moreover, students using other systems may learn more from the hints provided.

About the sub-task nature of tutoring and tuteeing, there are some significant differences in the posttest results. Low performance of HTS is expected since students only need to watch the computer tutee, expand the DT when needed, and choose a hint for the student, thus is less intellectually demanding than tuteeing. In our interviews, most students said that they like tutoring the computer because it is an easy task and they agree that they should have learnt more if they had to do the problems themselves. Five out of six HTS students prefer teaching computer to a real student because it is easier. CTS students like having hints and some of them think that there are not enough hints.

Unlike other systems, all RTS students felt that reciprocal tutoring is like playing a game. Now, if CTS represents a typical ITS, then the performance of RTS is about the same as an ITS. One possible explanation for this is that in reciprocal tutoring, when the student plays a tutor, he or she regards the tutee as a counterpart of himself or herself. So the tutor also involves solving the problem, in a degree much larger than when in HTS, while at the same time being benefited by meta-cognitive activity — taking another person's position to watch throughout the process. Certainly, we need more investigation in this direction.

We successfully fooled two students who were asked to work on a 'distributed' RTS on two connected machines in different locations cooperatively; in fact, they used two independent RTS systems. This either implies that our approach of modeling a virtual tutee is psychologically plausible enough to the student tutor or students believed what we told them simply because they trusted us. The narrow bandwidth of the information between the communicating agents clearly contributes a part of this psychological plausibility. It is also possible that the student tutor does not have a good model of the computer tutee.

5. Discussion

Besides the centralized systems, we are also investigating two distributed reciprocal tutoring systems. One simply consists of two connected machines with a dyad taking turns to play the roles of tutor and tutee. Another one is a twofold reciprocal tutoring system. It consists of three connected machines with a triad participating three different sub-tasks. Detailed description of these distributed systems is beyond the scope of this paper. A very preliminary empirical trial indicates that there is possibility that the distributed systems may surpass the centralized systems. A reason for this perhaps is the interpersonal motivation or peer pressure generated in the cooperative learning process. Currently, we are moving these systems to Internet where the artificial learning companion is an option for the student to choose as a partner to cooperate, especially when there is no other student on-line.

Our current implementation of the simulated tutee can be improved in several aspects. For example, we can employ Bayesian net to represent the tutee model, instead of using the current simple concept list. Bayesian net provides a representation of more complex concepts and relationships and a mechanism of updating and making reference of the belief values. All the decomposed solutions can be represented more systematically in a single hierarchy of strategies in solving Lisp recursions (Greer & McCalla, 1989). To enhance the psychological plausibility of a virtual learning companion, task-related motivation may be modeled to reflect the tutee's feelings and attitude, for example, confidence and effort. Also, it is technically desirable to adopt a single student model to simulate both the tutor and the tutee.

Tutoring represents a range of actions, from simply lecturing, evaluating and diagnosing the tutee's answers, offering opinions, hints, advice, suggestions, and strategies, motivating the student, and even monitoring all parts of the learning activities. The tutoring activities discussed here only represent a small part of this range; yet, they promote meta-knowledge, such as strategic knowledge, judgment knowledge of the performance. This type of knowledge potentially fosters knowledge transfer, that is, learned knowledge to be applied to novel problems or domains. However, how to model this kind of knowledge and measure the learning effect are not easy tasks. Still, the preliminary evaluation indicates that reciprocal tutoring provides cognitive benefits that could possibly be different from other protocols of learning. More data and analysis are needed to answer important questions such as whether human learning companion is better than virtual learning companion or whether intelligent cooperative learning systems are superior to intelligent tutoring systems.

It is difficult to design game-oriented learning activities where the student can sustain motivation while paying effort to accomplish a learning task. The preliminary evaluation indicates that the reciprocal tutoring learning activity possesses some game elements. More observation is needed to reveal these elements and how they can be incorporated with the student model to enhance the motivational effects of reciprocal tutoring. Indeed, it is most desirable that all kinds of learning activities are motivational.

Reciprocal tutoring interleaves tutoring with tuteeing, or the vice versa, allowing immediate use of the learned knowledge and meta-knowledge of the domain. However, learning may also be proceeded in two phases: tuteeing followed by tutoring. An interesting scenario is that a student is being tutored by a computer tutor first, then trains an artificial agent so that its performance can reach to a level to defeat another opponent computer agent. Reversing these two phases, that is, tutoring followed by tuteeing, is also possible. This case can be regarded as a 'fading' process since the computer tutee partially does the modeling process for the student before the student works on the task (Collins, Brown, & Newman, 1989). In general, if learning by tutoring cannot perform well when stands alone, how it combines with other general forms of learning to generate effective learning paradigms remains a research question.

Finally, putting the computer in the position of a tutee and being taught by the student make the simulation of a virtual companion a salient feature of the system, and, as can be seen, approximating a typical student is technically equivalent to student modeling. Thus, the student model plays a critical role in designing reciprocal tutoring system, and, therefore, many cooperative learning activities. It dismisses doubts of the importance of the student model in cooperative learning (McCalla, 1990; Chan, 1991). Also, this work unfolds some different uses of the student model: When student model is used in traditional ITS fashion, it is a supporting component for the computer tutor (usually called the tutoring module) and is hidden from the student. When the student needs to teach another student or the computer, student model is a tool manipulated by the student to fulfill the responsibility of being a tutor. When the computer simulates a virtual learning companion, it plays as an active and autonomous agent inside the system. However, how the student model can become a general 'powerful' engine to be used faithfully remains to be a long term research problem.

Acknowledgments

We would like to thank for the support of the National Science Council of Taiwan and the members of the LISA project. We also thank for the anonymous reviewers' invaluable comments for the improvement of this manuscript.

References

1. Bhuiyan, S., Greer, J.E., & McCalla, G.I. (1992). Learning recursion through the use of a mental model-based programming environment, The 2nd International Conference of Intelligent Tutoring Systems, *Lecture Notes in Computer Science*, 608, Springer-Verlag, 50-57.

2. Chan, T.W. (1991). Integration-kid: a learning companion system. *The Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, Morgan Kaufmann Publishers, Inc., 1094-1099.

3. Chan, T.W. & Baskin, A.B. (1988). Studying with the prince: The Computer as a Learning Companion. *The Proceedings of International Conference of Intelligent Tutoring Systems*, 1988, June, Montreal, Canada, 194-200.

4. Chan, T.W. & Baskin, A.B. (1990). Learning companion systems. In C. Frasson & G. Gauthier (Eds.) *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Chapter 1, New Jersey: Ablex Publishing Corporation.

5. Chan, T.W., Chung, Y.L., Ho, R.G., Hou, W.J. & Lin, G.L. (1992). Distributed learning companion systems -- WEST revisited. The 2nd International Conference of Intelligent Tutoring Systems, C. Frasson, G. Gauthier & G. McCalla (Eds.). *Lecture Notes in Computer Science*, 608, Springer-Verlag, 643-650.

6. Collins, A., Brown, J. S., & Newman, S. E. (1989) Cognitive apprenticeship: teaching the

craft of reading, writing and mathematics. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser,* Hillsdale, NJ: Lawrence Erlbaum Associates Publishers.

7. Greer, J., & McCalla, G. (1989). A computational framework for granularity and its application to educational diagnosis. *The Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, Michigan, Morgan Kaufmann Publishers, Inc., 477-482.

8. Jehng, J.C., Shih, Y.F., Liang, S. & Chan, T.W. (1994). TurtleGraph: a computer supported cooperative learning environment, *The Proceedings of the World Conference on Educational Multimedia and Hypermedia*, Vancouver, Canada, AACE, 293-298.

9. Lai, J.A. (1994). Contest-Kid: a distributed competitive learning environment. Master Thesis, Institute of Computer Science and Electronic Engineering, National Central University, Taiwan.

10. McCalla, G. (1990). The central importance of student modeling to intelligent tutoring, Research Report, 90-7, ARIES Laboratory, Computational Science Department, University of Saskatchewan, Saskatoon, Canada.

11. McManus, M. M., & Aiken, R. M. (1993). The group leader paradigm in an intelligent collaborative learning system. In S. Ohlsson, P. Brna, and H. Pain (Eds.), *Proceedings of the World Conference on Artificial Intelligence in Education*. Charlottesville, VA: Association for the Advancement of Computing in Education, 249-256.

12. Newman, D. (1989). Is a student model necessary ? Apprenticeship as a model for ITS. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Artificial Intelligence and Education*, Amsterdam: IOS, 177-184.

13. Nichols, D. (1994). Issues in designing learning by teaching systems, AAI/AI-ED Technical Report No. 107, Computing Department, Lancaster University, Lancaster, United Kingdom.

14. Palthepu, S., Greer, J., & McCalla, G. (1991). Learning by teaching. *The Proceedings of the International Conference on the Learning Sciences*, AACE.

15. Palincsar, A. S. & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and monitoring activities. *Cognition and Instruction*, 1, 117-175.

16. Self, J. (1988). Bypassing the intractable problem of student modeling. *International Conference of Intelligent Tutoring Systems*, Montreal, Canada, 18-24.

17. Ur, S. & VanLehn, K. (to appear), STEPS: A preliminary model of learning from a tutor. Proceedings of the 16th Annual Conference of the Cognition Science Society. Hillsdale, NJ: Erlbaum.

18. VanLehn, K., Ohlsson, S. & Nason, R. (1994). Applications of simulated students: an exploration, *Journal of Artificial Intelligence in Education*, Vol. 5 No. 2, 135-175.

19. VanLehn, K. (1993). Keynote speech, *World Conference on Artificial Intelligence in Education*, Edinburgh, Scotland, August.

Authors' Addresses

Tak-Wai Chan and Chih-Yueh Chou: Institute of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan 32054, R. O. C., email: chan@sparc15.src.ncu.edu.tw.