Exploring the Design of Computer Supports for Reciprocal Tutoring

Article *in* International Journal of Artificial Intelligence in Education · January 1997

Source: OAI

CITATIONS
79

READS
167

2 authors:

Tak-Wai Chan
National Central University
178 PUBLICATIONS **3,860** CITATIONS
SEE PROFILE

Chih-Yueh Chou
Yuan Ze University
62 PUBLICATIONS **1,487** CITATIONS
SEE PROFILE

International Journal of Artificial Intelligence in Education (1997), 8,1-24

Exploring the Design of Computer Supports for Reciprocal Tutoring

Tak-Wai Chan Chih-Yueh Chou

Institute of Computer Science and Information Engineering
National Central University
Chung-Li, Taiwan 32054, R. O. C.

Abstract

This paper explores the design of a set of system prototypes for supporting a protocol of cooperative learning activity called reciprocal tutoring. During reciprocal tutoring, two or more agents in centralized or distributed environments interact as they take turns at playing the roles of a tutor and a tutee in solving Lisp recursive problems. These agents are either real students or virtual learning companions simulated by the computer. Furthermore, in the design of a virtual tutee, which is one of the roles assumed by the virtual learning companion, we found that the student model plays a critical role in adaptive interaction with the student, thus broadening the applicability of the student model in one-on-one tutoring settings. Preliminary experimental trials of these systems have been conducted.

Keywords: reciprocal tutoring, cooperative learning, learning by teaching, learning companion, student modeling

1. Introduction

In early computer assisted learning research, the question "Should the computer teach the student or vice-versa?" was first raised by Luehrmann (1972) when he argued that, like reading and writing, computing constitutes a new and fundamental intellectual resource that should not be a mere delivery system for instruction. Students should learn how to employ computing as a resource for learning. By teaching or instructing the computer through computing, students may learn more about the process of learning than they do from being tutored by software written by others. Taylor (1980) further elaborated that a computer can be conceived as a *tutor*, a *tutee*, a *tool*, or a *toy*. We should note that sometimes it may not be easy to distinguish these three traits of computers, for example, whether Logo is tool, a toy, or a tutee. However, the differentiation of these traits is useful to designers in identifying their goals and developing the characteristics in computer software that they want to portrait to the users. In addition, the advancement of multimedia and network technology today enables the computer to be an interactive *textbook* with multimedia as well as a *telecommunicator*. Thus computer and network resemble 6T.

In educational psychology studies, the *reciprocal teaching* method (Palincsar & Brown, 1984; Brown, 1992) is a prominent model employed in reading

comprehension. It was modeled after studies of Socratic or inquiry teaching (Collins and Stevens, 1982), but in a peer group setting. Participants take turns as a learning leader, who begins by asking a question and ends by summarizing the gist of what has been read. The group rereads and discusses possible problems of interpretation when necessary. Questioning provides the impetus to get the discussion going. Summarizing at the end of a period of discussion helps students establish where they are in preparation for tackling a new segment of text. Attempts to clarify any comprehension problems that might occur arise opportunistically, and the leaders ask for predictions about future content. These four activities — questioning, clarifying, summarizing, and predicting — bolster the discussion because they are comprehension-monitoring devices. The model is recently being combined with Jigsaw method (Aronson, 1978) in fostering a community of learners (Brown, et al., 1993) that features students as designers of their own learning and engages students in self-reflective learning and critical inquiry in an intentional learning classroom (Scardamalia & Bereiter, 1991).

Intelligent tutoring systems (ITSs) research, as indicated by its name, intends to mimic the process of private tutoring in a computer supported environment. While tutoring is a useful metaphor for describing human computer interaction for learning, Chan and Baskin (1988, 1990) appealed to the computer to impersonate explicitly multiple artificial agents: a teacher and a *learning companion*. Also, researchers like Self (1988) advocate that the student may undertake collaborative learning with the computer. Prototypes of these alternatives to ITSs have been developed. Integration-Kid, a learning companion system, explores various patterns of interactions through different protocols of learning activities among the agents, such as cooperation, competition, and reciprocally working and observing (Chan, 1991). The performance of the learning companion is governed by a subset of problem solving expertise and some faulty knowledge. During the learning process, this problem solving expertise is expanded and the faulty knowledge is tuned by simply deleting and adding knowledge units. People Power and Wombat are collaborative learning systems where a computer takes the role of a collaborator (Dillenbourg and Self, 1992; Blandford, 1994). These systems investigate how the computer and the student co-generate reflective social dialogue through argumentation or negotiation.

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 model is also called *learning by teaching* and was elaborated by Palthepu, 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 'almost knows' the domain and that the system would enhance the student's meta-cognitive reasoning skills. They 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. Following such a *tabula rasa* approach, Nichols (1994) developed a system and found that subjects who tested the system expressed "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 a pedagogically useful direction. Besides being a learning partner of the student, a simulated student can potentially be useful for formative evaluation of an instructional designer's prototypes and teacher training. This task is called 'meta-tutoring' (VanLehn, Ohlsson, and Nason, 1994; VanLehn, 1993). Ur and VanLehn (1994) applied explanation-based machine learning methods to simulate a physics student for the purpose of tutor training and used it to study the cognitive process of learning from a tutor.

Computer support for peer learning is not restricted to centralized systems. Distributed systems that support multiple agents to take different roles to interact over connected machines are being studied (Chan et al., 1992; McManus and Aiken 1993; Jehng et al., 1994). Indeed, distance learning environments in the future will need artificial learning companions to ensure their accessibility if there is no appropriate learning partner available on-line. For some learning activities, the artificial learning companions together with the real students on the network can form a virtual learning group (Chan & Lai, 1995).

This paper first explores the design of a set of reciprocal tutoring environments to practice Lisp recursion. *Reciprocal tutoring is a protocol of learning activity, where two or three agents (an agent is a computer or a student) take turns to play the roles of a 'tutor' and a 'tutee'*. Next, we construct a Petal-Like-System (Bhuiyan, Greer, and McCalla, 1992) to provide a scaffolding tool for the student by exploiting the computing power of the machine to perform parts of the learning task for the students. Finally, like most ITSs, we employ student modeling as the basis to deal with students' difficulties that emerge during problem solving, enabling adaptive interactions between agents. For example, in one of the reciprocal tutoring environments, the student model is used as an intelligent tool for a student to tutor another student through a connected machine. In another environment, we use the student model to implement a virtual learning companion. The rest of this paper is organized in the following way: After discussing the various reciprocal tutoring configurations, we describe different supports for a set of reciprocal tutoring systems. Then we discuss experimental trials of these systems. Finally, after more discussions and future work, we arrive at a few conclusions.

2. Variations of Reciprocal Tutoring

Reciprocal tutoring is a form of cooperative learning by adopting the 'divide and conquer' strategy. A learning task is partitioned into sub-tasks of tutoring and 'tuteeing' where tuteeing is learning by working on the solution while being tutored. These sub-tasks are distributed to different agents and each agent is responsible for a sub-task, while the rest of the work is taken care of by other agents. Integration-Kid (Chan, 1991) supports a close approximation of the reciprocal tutoring protocol where the virtual learning companion and the student reciprocally works and observes each other on different problems. If the observer is more capable or knows the answer to offer helpful comments, then it would become reciprocal tutoring. In contrast to reciprocal teaching in the domain of reading comprehension, the learning task of Lisp

recursion can not be decomposed into many significant sub-tasks without lessening the importance of each agent's role.

In general, we may consider: agents, which are either real (human) or virtual (computer simulated); roles, which are either tutor or tutee (learner); and location, which is either centralized (a single human student interacting with a computer) or distributed (multiple human students in different places connected by a network). If there are only two agents involved in the learning environment and if we let R, V, T, and L denote real, virtual, tutor, and learner, respectively, then we can envisage 10 possible configurations:

RL+RL	RT+RT
RL+VL	RT+VT
RL+VT	VL+VL
RT+RL	VT+VL
RT+VL	VT+VT

Clearly we are not interested in configurations in which all agents are virtual or in which all roles are tutors, therefore the second column of the above list will not be considered. Co-exist collaboration where human students work together using a standalone computer is not within the scope of discussion in this paper. Therefore, if there are multiple real students involved, then we assume a distributed environment. Now let us consider the first configuration, RL+RL, the configuration simply means that the two students collaborate (or compete) and communicate with two connected computers. TurtleGraph (Jehng, et al., 1994) is such a system, when difficulty arises, the two connected students are encouraged to communicate and discuss the solution. The second combination RL+VL is a form of collaborative learning between the student and the computer. This type of system was suggested by Self (1988) and is exemplified by the People Power system (Dillenbourg & Self, 1992). The third combination RL+VT is the usual ITS model. The fourth one, RT+RL, distributes two different roles, tutor and learner, to students. The last one, RT+VL, is the inverted model of the ITS discussed above referred to as the *learning by tutoring* model, which puts the student in the position of a tutor. We shall come back to discuss these last they combinations because they involve T and L roles.

If three agents are involved there will be 20 possible configurations. We shall discuss some of the relevant models. RL+VT+VL is essentially the original learning companion system model. RT+RL+RL involves three students, one of whom serves as a peer tutor who may have more advanced skills, perhaps only because of computer support. RT+VL+VL is an extension of the learning by tutoring model in which the student now instructs or monitors two computer learners. RT+RL+VT is a model in which the VT is a meta-tutor, that is, VT helps RT to tutor RL. But if the RT knows more than the RL and has some good knowledge about the domain, then a VT may not be necessary.

For more than three agents, there are many more combinations. But we can still discuss several relevant models. Three's Company (Chan, 1995), RL+VT+VL+VL, is a direct extension of the learning companion system model. In a further extension of the Three's Company system Glassroom (Chan, 1995), has

configured a scenario which involves the combination RL+VT+VL+VL+VT+VL+VL. The additional virtual learning sub-group VT+VL+VL in Glassroom is observed by another group, RL+VT+VL+VL, which is in fact a Three's Company group. In Distributed West (Chan, et al., 1992), two connected students collaborate to play against the computer (RL+RL+VL) or compete against each other with the help of a private tutor for each student (RL+RL+VT+VT). In the group leader paradigm (McManus & Aiken, 1993), the computer is the leader in coordinating the activities of a group of students, represented by the combination VT+RL+RL+.... In Contest-Kids (Chan & Lai, 1995), a group of students of unlimited number play a competitive game, represented symbolically by RL+RL+RL+....

Since we are interested in the concept of reciprocal tutoring, we will only consider those configurations in which the roles of T and L and agents take turns or reciprocate in their activities. In particular, we shall study seven models. Many of these models involve the use of two tools. The first tool is an intelligent tool, called Diagnosis-Hint-Tree, is developed for the student tutor. It can be used to locate the tutee's error and generate a variety of hints in a menu for the student tutor to choose. Once the student tutor picks a hint, this hint will be sent to the tutee. The second tool, called the Petal-Like-System, is developed for the tutee. It is a nicely designed scaffolding tool to help students to construct the Lisp program and solution. Detailed discussion of the Diagnosis-Hint-Tree and the Petal-Like-System will be in next section.

The first model, called the *distributed reciprocal tutoring* system (DRT), is represented by the combination RT+RL+VT. The system allows two students (a dyad) to work cooperatively but separately on two connected machines to solve problems (see Figure 1). One plays the role of a 'tutee' while the other assumes the role of a 'tutor'. The students take turns in these roles in different problems. While the tutee is working on a problem, the tutor, who knows the answer, is watching how the tutee works. However, knowing the answer is not enough; when the tutee faces difficulties or makes errors, the tutor, who is not necessarily more capable than the tutee, may not be able to offer help. A natural way to handle this situation is to provide an intelligent 'super-tutor', VT, to help the student tutor to assist the tutee in problem solving. The role of the super tutor is to make sure that the student tutor knows where the tutee's difficulty lies and to offer relevant help in the tutoring process. Instead of implementing an 'active' super-tutor agent who informs the student tutor of the tutee's mistakes, an alternative approach is adopted.

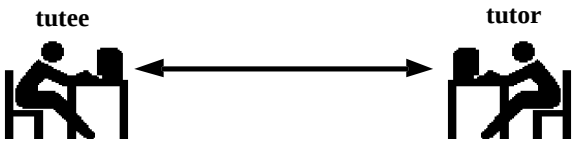

Figure 1. A Learning Dyad in Distributed Tutoring System

The second model is also a distributed system called the *distributed responsibility sharing* system (DRS). It is represented by the combination, RT+RL+RL, which involves a triad of three human students, who take turns playing different roles (see Figure 2). The solution construction task in DRS is shared by two

students. DRS is a twofold reciprocal tutoring system. One student plays the role of 'designer' who uses a Petal-Like-System, another student plays the role of a tutor and, like DRT, uses the Diagnosis-Hint-Tree to assist the designer. After the designer obtains a solution approved by the tutor, the solution is passed to another student, who plays the role of 'translator', to translate the program into Lisp, again, via another Petal-Like-System. The tutor mainly helps the translator use the correct Lisp syntax in the process.

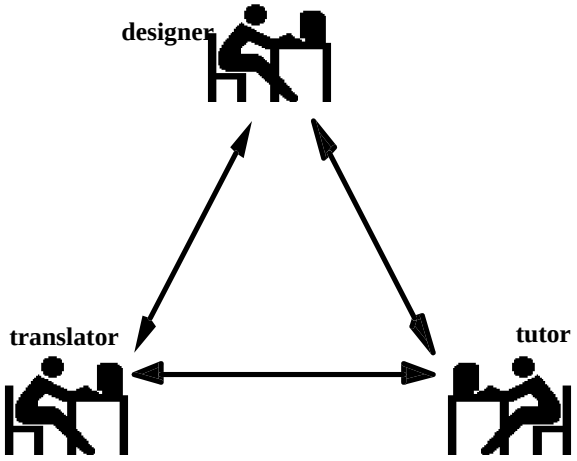

Figure 2. A Learning Triad in the Distributed Responsibility Sharing System

The third model is the combination of ITS and *Learning By Teaching* (LBT), that is, the student and the computer switch their roles of R and L for alternate problems. We call this a *centralized reciprocal tutoring* system (CRT). The ITS and Learning By Teaching models are included as the fourth and fifth models for comparison. In an ITS, students use a Petal-Like-System to construct Lisp programs. When there are mistakes, a student model is used to offer hints directly to the student. The student model lies inside the system and is hidden from the student. In LBT, the computer plays the role of a tutee, being tutored by the student. The configuration of LBT can also be RT+VT+VL, where VT may help RT to tutor VL. In this configuration, a virtual companion, who delineates the sub-optimal performance of a tutee, and a virtual tutor, which helps the student tutor, are used. To tutor the virtual learning companion, like the student tutor in DRT, a Diagnosis-Hint-Tree is used by the student. Indeed, if we were only interested in pure tutoring and pure tuteeing, we could have modified the DRT model by canceling the role-switching part of the activity. However, we are more interested in comparing traditional ITSs with other systems. Therefore, we will study the two sub-systems, ITS and LBT, of the CRT system separately and then compare them with the CRT system itself.

In an ITS, a computer tutor diagnoses and gives hints directly to students. In fact, an interface for the student may be provided to locate and diagnose her mistakes before the system generates hints. So the student is more active and the computer is more passive when the student debugs the program. We shall call this sixth system, a modified version of ITS, the *self-diagnosing* system (SD). In SD, like ITS, students use the Petal-Like-System to construct Lisp programs. When errors are encountered or help is needed, they use Diagnosis-Hint-Tree to spot their own errors and receive hints from the student model. The student model used in SD is similar to the student

model used for teaching Portuguese (Bull, Pain, and Brna, 1993) where the student can inspect and modify the student model through user/system negotiation. Finally, we consider a single agent (RL) model in which the student is a learner who works alone (WA) and the computer only provides the Petal-Like-System tool.

3. Supports for Reciprocal Tutoring

In our observations of mathematics problem solving using paper and pencil (Chan, 1989), we noticed that students go through several learning stages. At the beginning, they acquire relevant unfamiliar information from demonstrated examples, in particular, they focus on the syntax or the form of problem solutions in examples. At a certain point, when the student starts to relieve from the burden of syntactic details, a process of learning to understand more about the essential parts of the problem begins. Slowly, their emphasis shifts from accurate description towards effectiveness. For example, steps which would shorten the solution path are considered. As problems become more complex, students increasingly integrate the newly learned technique with previously learned knowledge. In short, the student's learning evolves from recognizing new knowledge to gradually increasing control over that knowledge. In most problem domains, even learning a simple task at the beginning, there are many elements to master. If students have to handle the whole learning task at the outset, they will inevitably encounter difficulties. Therefore, in computer assisted learning, designing a set of computer or human supports is crucial. These supports serve to enable the learning processes to proceed steadily. Indeed, given a limited amount of a learning resource, such as time, students should be rewarded by maximizing their learning instead of wasting effort by unnecessary struggle at a certain point or because of overwhelming problem complexity. It is the design that leads us to recognize various concerns and put all these concerns together in creative flux to build a learning environment. In this section, we shall explain the details of the implementation of these supports in reciprocal tutoring.

3.1 Cognitive Load Sharing

Cognitive load sharing is a salient feature of most of the systems considered in this paper. In writing a Lisp recursive problem, a student seek an approach and then constructs a program. If errors are found, the program is diagnosed, alternatives are made, possible errors are corrected, the program is modified or reconstructed. If there are errors in a modified program, the diagnostic process is repeated (see Figure 3). Note, in most ITSs, the first two sub-processes and the last two sub-processes are performed by the student while the two sub-processes in the middle are carried out by the computer. In reciprocal tutoring (e.g. DRT), the middle two sub-processes are executed by another student instead. Therefore, both usual human tutoring and reciprocal tutoring can be regarded as a cooperative learning process.

Basically, reciprocal tutoring seeks to add a social element to the spontaneous process of learning by doing and encourages students to reflect and compare as she observes her own and her partner's learning processes. Learning in reciprocal tutoring, therefore, can be characterized as a cyclical process of construction and reflection.

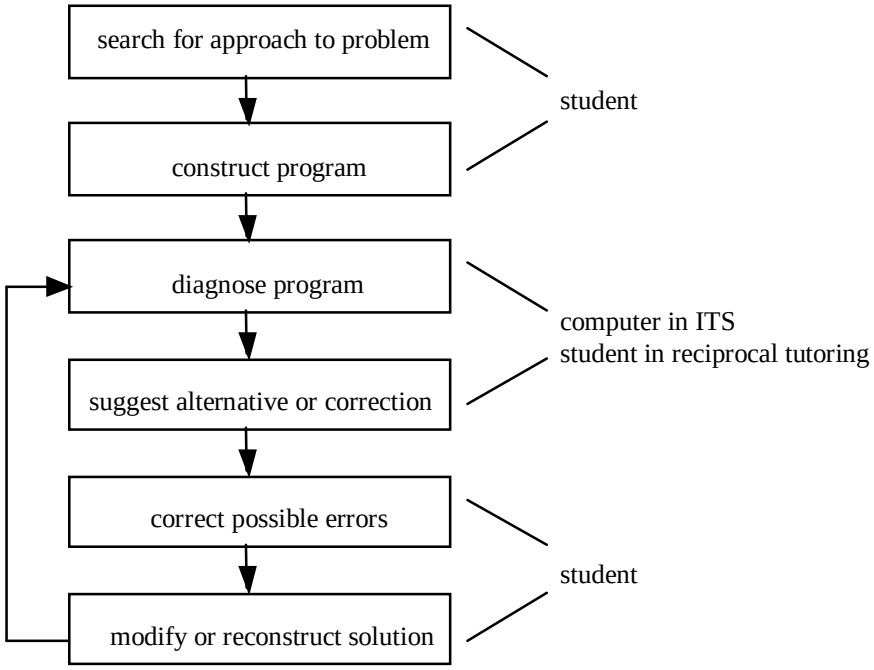

Figure 3. Sub-processes in developing a correct recursive Lisp program

3.2 Petal-Like-System

In apprenticeship learning, scaffolding refers to the assistance offered to students by the expert in the process of accomplishing a complex learning task (Collins, Brown, & Newman, 1989). The expert completes those parts of the task that students have not yet mastered. In other words, students participate in the task only to the degree that they can manage and with the amount of responsibility that they are capable of assuming. Scaffolding is temporary and matches the level of mastery of the students. At a later stage of learning, the process of fading proceeds, that is, the expert's support is gradually removed when students learn to handle more of the task on their own and students are increasingly asked to make up for the missing parts that were being supported. When scaffolding is employed in a computer based learning environment, it refers to the facilities offered by the computer. This is usually done by providing a tool that employs the computing capability of the machine to share a learning sub-task and leaves the rest to the student. This can then reduce the task complexity to the student. We call such a tool a *scaffolding tool*

Most systems described in this paper are supported by a Petal-Like-System (PLS), a typical scaffolding tool. PLS is a part of the Petal system, called PET1 (Bhuiyan, Greer, and McCalla, 1992). PET1 originally provided a mental model-based programming environment to free students from dealing with the syntax problem since there is a set of Lisp 'code chunks' (or expression templates) displayed as buttons on the screen (%t and %f are the Boolean values true and false in our small Lisp language). These code chunks allow the student to construct code without making syntactic errors. Thus, students do not worry about the formal details or legality of the expressions and can pay more attention to the semantics and the heuristic knowledge required for developing a solution. In addition, several advantages of PLS

that were not described in the original paper are revealed. We shall describe them here:

- PLS in fact employs the calculator metaphor which obeys almost all of the good user interface principles (Smith, Cypher, & Spohrer, 1994). It allows students to produce code by invoking a sequence of actions and examining their effects. When the code is what they want, they stop. These actions are not performed by memorizing and typing, but by seeing and pointing – a hand-eye coordinated direct manipulation. The calculator metaphor visualizes information for understanding what is going on and what to do next. Indeed, with proper re-framing of the domain, most problem solving domains can use such a calculator metaphor since they usually require a set of operators to transform the problem state until a solution is reached.

- PLS provides handy problem solving management. The tutee does not need to write the Lisp constructs such as 'defun' and 'cond' as well as the function name and the argument list, but only focuses on filling in the content of the conditional clauses. When the tutee finishes a problem, the system will take up the finished code, obtain the other necessary parts of a complete program from the system and test the program with examples using a Lisp interpreter.

- PLS requires the student to specify which case — base or recursive — is going to be dealt with in the next conditional clause. This reminds the student that in solving a recursive problem, two cases, and only these two cases, have to be tackled, though a problem could have multiple base cases and recursive cases.

- The restriction of using a single 'cond' construct implies that this construct is by itself a schema that suffices to solve a set of recursive problems. We noticed that students using a Lisp language interpreter tended to try more than one 'cond' and sometimes combined a 'cond' with an 'if' construct. However, it turns out that solutions using a single 'cond' construct are usually the most readable and elegant.

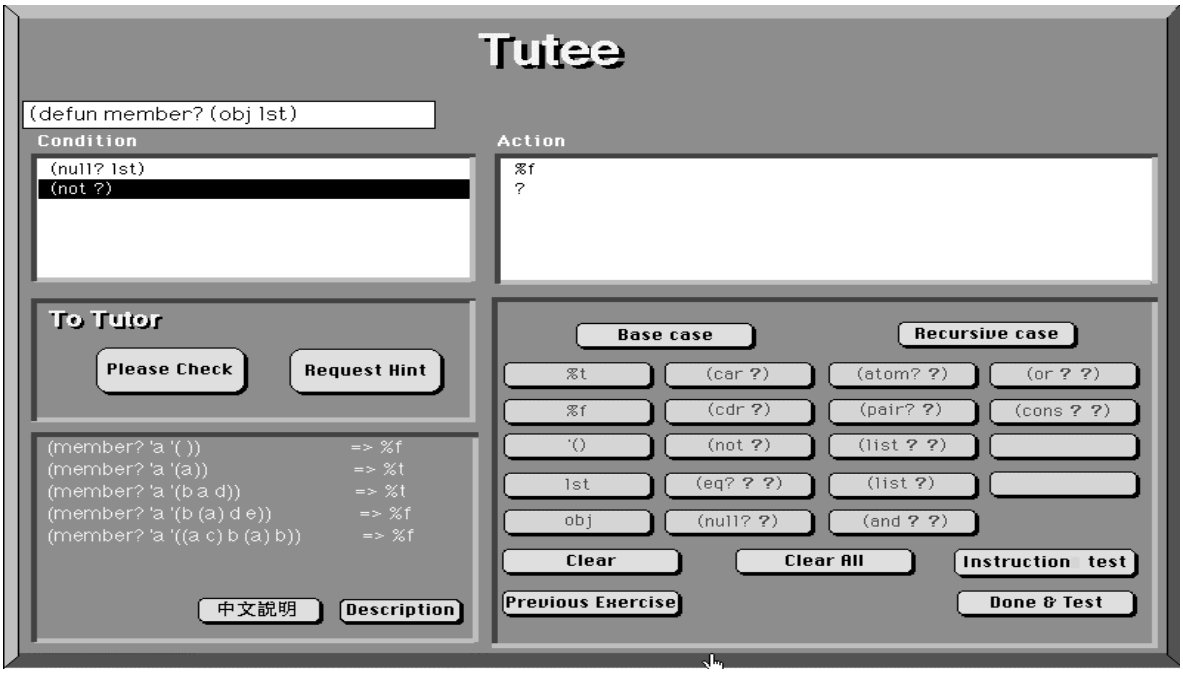

Figure 4. Tutee's Petal-Like Scaffolding Tool

Since PLS handles the syntax of the language as well as some other peripheral support required to manage the learning task, the space of possible errors made by the tutee in writing Lisp recursion code is reduced. This allows the tutee to get on with the important features of Lisp recursion problems more easily and enables the tutor to focus on the tutee's essential errors. First, using the code chunk buttons instead of the keyboard, PLS cuts down dramatically on the space of the student's possible responses. Suppose a Lisp expression is composed of 5 code chunks and assume that there are 20 code chunks on the screen, then the number of possible responses is 20^5. If students use a keyboard, assuming that there are 30 characters on the keyboard and on average 5 characters are required to form a code chunk, then there can be 30^{25} possible responses. To the system, a student's response is either expected or unexpected and either correct or incorrect (see Figure 5). The use of PLS significantly reduces the number of unexpected responses, which consist syntax errors, such as typing or careless mistakes. This permits the system to concentrate most on responses that are expected, no matter whether they are correct or incorrect. Second, by restricting students to the use of a single 'cond' construct, PLS diminishes the number of possible correct solutions to one or two. This, in turn, reduces the complexity of the relationship between the student and the learning companion and simplifies the design of the student model.

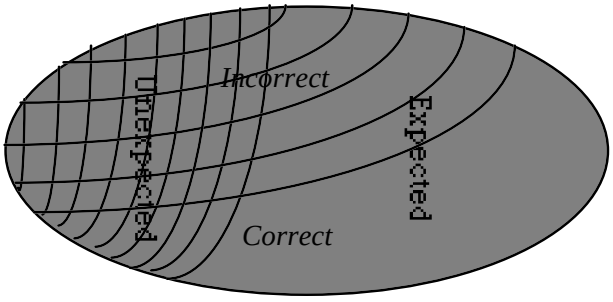

Figure 5. Student Responses Divided by Correctness and System Expectation

The PLS described thus far is used in all systems except the DRS. There are two variations of PLSs used, the PLS-SNL and the PLS-KEY. The designer uses PLS-SNL when the code chunks are expressed in a Semi-Natural Language syntax instead of Lisp syntax. The premise is that this language is closer to the mental model of the student, and thus it is easier to express the problem solution in this language than in the symbolic Lisp language. This is also similar to the idea of PET2 in the original Petal system. The 'translator' uses another Petal-like system, PLS-KEY, where the code chunk has to be KEYed in by the translator. The 'tutor', like tutors in other systems, uses the Diagnosis-Hint-Tree as an interface.

Figure 6 shows the interface of the translator observing the solution of the designer. The third line of code in the top right rectangle of the semi-natural language solution of the designer in Figure 6 is: "if the above are all false, then return[call: findb?, arguments: the tail of lst]."

Figure 6. Semi-Natural Language of the Petal-Like System

3.3 Virtual Tutor

Assistance offered in a cooperative learning environment is usually performed by reducing the cognitive load or by facilitating performance. Instead of deleting learning elements in reducing cognitive load, facilitating performance can be achieved by adding new learning elements for the student. These new elements enhance learning performance and allow the student to proceed more smoothly with the learning task. The advice or suggestions given by a virtual tutor are examples of this kind of assistance.

Tutoring is an adaptive interaction process. By adaptive interaction, we mean interaction that can facilitate changes in the learning environment which correspond to the varying and changing needs of different agents over time. Through this interaction, students receive information to dismiss doubts and overcome difficulties encountered during problem solving. Adaptive information allows students to proceed steadily with the problem solving and prevents cognitive roadblocks — getting stuck. To support such interaction, we must understand the student's problem situation and thus some kind of student model is needed. To simulate a tutor, like other ITSs, we implement a student model. If the computer plays the role of a tutor, advice or hints are generated from the student model. If a student plays the role of a tutor, the Diagnosis-Hint-Tree is provided as an interface on top of the student model. With the Diagnosis-Hint-Tree, the human tutor can explore and locate the current trouble of the tutee and use it to supply relevant hints to the tutee. Thus, the student model inside the computer acts as a 'super-tutor' to the student tutor, but in a rather passive form.

As noted before, the use of the scaffolding tool, PLS, significantly shrinks the space of student responses. In building the student model, we collected error data from many students using the scaffolding tool. A discrimination net was then developed for each problem based on the correct solution and the collected error data. Each node in the net represents a problem situation that might occur in a student's partial program and stores different levels of hints and suggestions. When the student requests a hint, the student's solution is matched with the net. On termination of the matching 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 the PLS, the problem's latitude has been narrowed to one or two possible correct solutions, so the discrimination net is rather linear. The disadvantage 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 with the ability to keep track of the student's actions and point out the tutee's errors and supply hints about how and what to do next in solving a problem.

3.4 Diagnosis-Hint-Tree

The Diagnosis-Hint-Tree (DHT) is an interface enabling the human tutor to interact with the student model in carrying out the tutoring task. Thus DHT is a scaffolding tool for the tutor. The human tutor can see the correct solution and the partial solution as well as every action of the virtual or real tutee (see Figure 7). The tutor cannot interrupt but waits for the tutee to ask for help or check a 'conditional clause' — a line of code including the 'condition' and the 'action' parts — when the tutee finishes writing the clause. When the tutee asks for help, the tutor does two things: locates the tutee's trouble and chooses a hint to give. The tutor expands the Diagnosis-Hint-Tree, starting from the 'base case' or 'recursive case' nodes (that are displayed as buttons on the screen). The tutor is required to pick the correct path in expanding the DHT. Nodes will not expand if the tutor hits the wrong button. When the tutor has successfully spotted the student's trouble, a number of hints (in Chinese)[1]

[1] The English translation of the second hint in Figure 7 is "consider a recursive call here" and that of the third hint is "if the first element is an atom, you can consider (sum-all lst) as the sum of the first element of the list and the result of applying sum-all to the rest of the list."

are available at the end of the path. The system uses a student model to determine whether the tutor has hit the 'right' or 'wrong' button. The 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 encourage the human tutor not to depend too much on the hints in the DHT and to pay more attention to the tutee throughout the tutoring process. Points are scored or deducted according to the performance of the tutor in locating the tutee's errors and penalties are given for using hints that are too close to the answers.

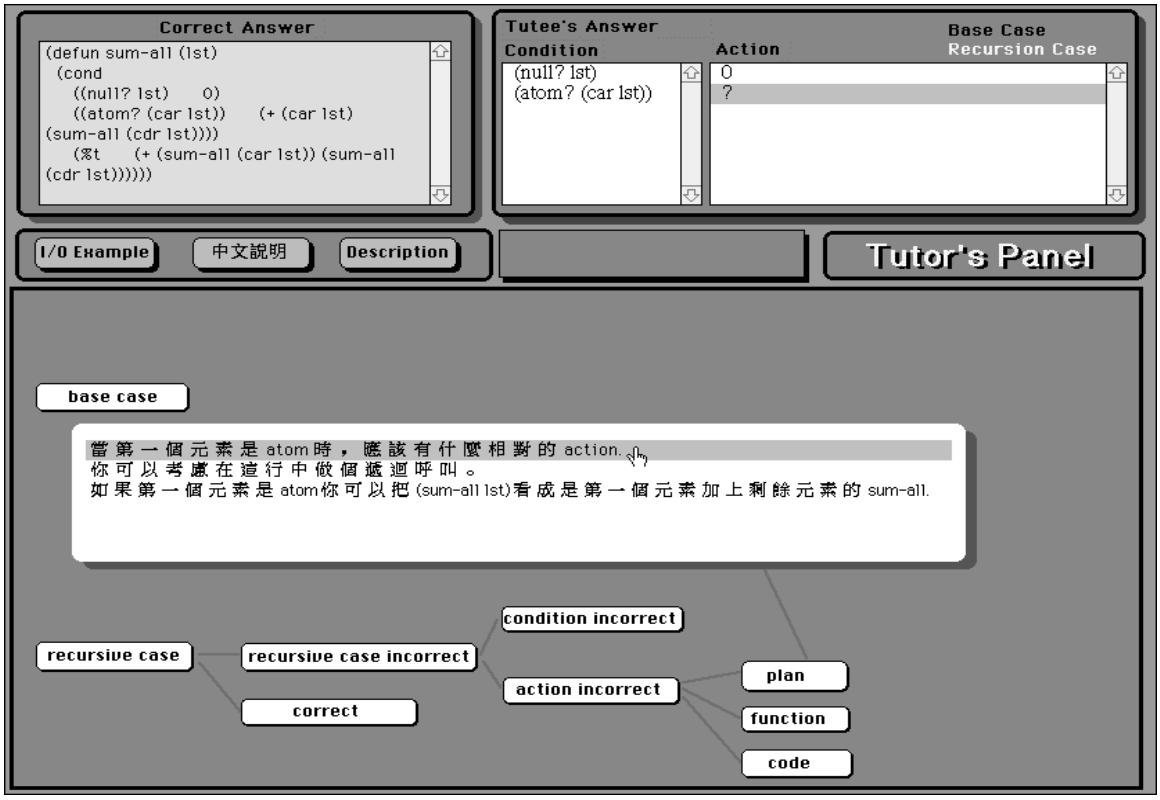

Figure 7. Student Tutor's Interface

3.5 Virtual Tutee (Learner)

To simulate a tutee, we have to model the behavior of a learner whose overall knowledge of the domain appears to be advancing despite occasional mistakes. The tutee's response may be incorrect. Sometimes the tutee may not be able to respond and asks for help. Thus, two objectives have to be fulfilled: the ability to model the evolving knowledge of the tutee and the modeling appearing to be 'psychologically plausible' to the student tutor. By psychological plausibility, we mean that the learning behavior and performance of the tutee must be understandable to the student. Instead of applying machine learning methods to empower the learning ability of the tutee, we take an overlay approach to enable the tutee to pretend to learn at about the level of an average student.

For each problem, the simulation of the tutee will refer to 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. It records the tutee's degree of understanding or competence in the domain. 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. 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 of students in previous empirical experiments. Response rules are heuristic rules that govern the responses of the tutee. Detailed discussion of the simulation of a virtual tutee is out of the scope of this paper (for full description of the simulation of a virtual tutee see Chan & Chou, 1995).

The systems are implemented in SuperCard and Mac Lisp on MacQuadra and PowerMac, using AppleEvent to link the two application programs. A small Lisp interpreter written in Lisp is used for testing a student's program and the student may use it to verify the properties of the Lisp language constructs. Distributed Systems use AppleTalk to connect.

Table 1. Summary of Supports for Reciprocal Tutoring Systems

System	Configuration	Companion(s)	Task(s)	Scaffolding Tool(s)
Distributed Reciprocal Tutoring (DRT)	RT+RL+VT	RT / RL+VT	Tutoring / Tuteeing	PLS / DHT
Distributed Responsibility Sharing (DRS)	RT+RL+RL	RL+RL / RT+RL	Tutoring / Tuteeing	PLS-SNL / PLS-KEY / DHT
Centralized Reciprocal Tutoring (CRT)	RL+VT / RT+VL	VT / VL	Tutoring / Tuteeing	PLS / DHT
Intelligent Tutoring System (ITS)	RL+VT	VT	Tuteeing	PLS
Learning By Tutoring (LBT)	RT+VT+VL	VT+VL	Tutoring	DHT
Self Diagnosing (SD)	RL+VT	VT	Tuteeing	PLS / DHT
Working Alone (WA)	RL	no	Working Alone	PLS

Table 1 is a summary of the supports for human students. The Companion(s) column refers to agents other than the student. The Task(s) column refers to the task(s) that the student may undertake. If the student does both tutoring and tuteeing, then the student will be involved with both spontaneous and reflective learning processes. The student may select only one of these tasks and leave the other to the computer or undergo both types of learning simultaneously as in 'working alone' (WA) model. For the first two systems, DRT and DRS, computers also serve as communicators between students. The DRS system provides the maximum support to the user and WA the minimum. There is slight difference between ITS and SD, the former uses an active virtual tutor and the later uses DHT, a passive virtual tutor. In

the DRT, DRS, CRT, and LBT systems, DHT acts as the super-tutor of the student tutor.

4. Preliminary Experimental Trials

The learning effectiveness of these systems were measured by a posttest after students' use of the system. A class of freshmen learning basic Lisp concepts used the systems to practice Lisp recursion problems for two hours. Their mid-term examination results were taken as a pretest and the students were distributed evenly in different systems according to their pretest result. Each system version was used by at least 5 subjects who all took a posttest with a Lisp language interpreter. Table 2 shows the resulting posttest scores.

Table 2. Posttest Scores of Different Variations of Reciprocal Tutoring Systems

System	Individual Posttest Score	Average Score	Standard Deviation
Distributed Reciprocal Tutoring (DRT)	100, 100, 90, 60, 85	87	14.69
Distributed Responsibility Sharing (DRS)	80, 80, 85, 70, 85	80	5.47
Centralized Reciprocal Tutoring (CRT)	80, 70, 80, 85, 75, 75	78	4.78
Intelligent Tutoring System (ITS)	55, 69, 100, 85, 75, 80	77	13.86
Learning By Tutoring (LBT)	60, 50, 60, 75, 75	64	9.69
Self Diagnosing (SD)	80, 55, 85, 85, 100	81	14.62
Working Alone (WA)	45, 60, 80, 80, 85	70	15.16

Given the limited number of subjects per system, the preliminary evaluation can only provide us with possible hypotheses concerning the effectiveness of reciprocal tutoring with no definite conclusion. Nevertheless, to facilitate our discussion of likely hypotheses, we investigate three perspectives of reciprocal tutoring.

4.1 Virtual vs. Real Companion

The first perspective we are interested in is the possible difference between a virtual learning companion and a real learning companion. For this, we found that DRT students seemed to out-perform CRT students (Table 3). Does this suggest that the virtual learning companion possesses any deficiency when compared to the real learning companion? This may not be the case because we succeeded to fool two students by asking them to use DRT, in fact, they used two CRTs independently (more discussion of this in next section). This indicates that the cognitive benefits due to peer interactions may involve other psychological factors other than pure cognition. For example, real learning companions may draw more of students' attention, creating better performance under peer pressure. The discovery of the attributes that contribute to the difference might help to improve the virtual learning companion's ability to stimulate the students. During the interviews after the test, all CRT and DRT

students felt that reciprocal tutoring is like playing a game. All of the DRT students found that the system was exciting and fun and some even enjoyed seeing their partners struggle in solving a problem. A few of the DRT students said that at times they preferred writing the hints themselves to picking a hint provided by the menu.

Table 3. Real and Virtual Learning Companion Perspective

System	Companion	Average Score	Standard Derivation
Distributed Reciprocal Tutoring (DRT)	Real	87	14.69
Centralized Reciprocal Tutoring (CRT)	Virtual	78	4.78

4.2 Sub-Tasks

Reciprocal tutoring, as mentioned before, can be regarded as a cooperative learning activity where each agent is responsible for a sub-task in accomplishing a complete learning task. Therefore, there are variations of task decomposition and sub-task assignments to agents. Some of these variations employ a student model, possibly in different ways, for adaptive interaction. The nature of the sub-tasks: tutoring, tuteeing, and their combinations, will now be examined.

Table 4 shows that some significant differences in the posttest results. LBT is 64, ITS is 77 and the average of DRT, CRT, and DRS is 82. Low performance of LBT is expected since students only need to watch the computer tutee, expand the Diagnosis-Hint-Tree when needed, and choose a hint for the student. This is a less intellectually demanding process than tuteeing. In our interviews, most students said that they preferred the LBT systems because tutoring is an easy task. They agreed that they would learn more by doing the problems themselves but enjoyed the passive role. Five out of six LBT students preferred teaching a virtual student to teaching a real student because it is easier. All of the students liked the hints provided by the system and some of them thought that there were not enough hints. Does the low performance of ITS imply that social learning systems, or at least reciprocal tutoring systems, are superior to ITSs in general? If so, one possible explanation lies in the dual perspectives of tutoring in reciprocal tutoring. When the student plays the role of tutor, she regards the tutee as her counterpart which involves her in the solution process to a degree much larger than in LBT. At the same time the student benefits from meta-cognitive activity — taking another person's position to observe the problem solving process. More investigation into the cognitive implications of these systems will be required before definitive conclusions can be drawn.

Table 4. Sub-Task Perspective

System	Task(s)	Average Score	Standard Deviation
Distributed Reciprocal Tutoring (DRT)	Tutoring / Tuteeing	87	14.69
Centralized Reciprocal Tutoring (CRT)	Tutoring / Tuteeing	78	4.78
Distributed Responsibility Sharing (DRS)	Tutoring / Tuteeing	80	5.47
Intelligent Tutoring System (ITS)	Tuteeing	77	13.86
Learning By Tutoring (LBT)	Tutoring	64	9.69

4.3 Amount of Support

The final perspective we investigate is the amount of support that is most beneficial to learning. Neither too many learning elements nor too many supports are good for students. If there are too many learning elements, the students will face roadblocks to learning and will be unable to proceed whereas if there are too many supports, students will be inundated with information. Supports should be incremental and built upon each other. The PLS is a scaffolding tool. On top of it, ITS brings forward an additional support where the student receives hints when needed. Another system, SD, is similar to ITS. Instead of hiding the student model inside the system, SD allows students to use DHT to spot their errors and receive hints from it. Finally, in addition to these system supports, the interpersonal supports of peer interactions and possible moral support can be generated in DRT and DRS due to the cyber social presence.

Table 5 shows that the average performance of the pure scaffolding tool (PLS) is 70, adaptive interaction (ITS or SD) is 79, and interpersonal supports (DRT or DRS) is 84. All WA students liked the system and thought that they could focus better on problem solving when the emphasis on syntax was removed and it was convenient to have other facilities, such as checking the solution with examples. Most SD students (5 out of 6) thought that the DHT was useful. Three preferred direct hints provided by the system (like ITS) and three thought that using DHT was helpful. In WA, since there was no hint provided, students sometimes could not proceed to other problems in two hours once they had trouble with one or two of the problems. Unlike WA, students are less likely to get stuck when there is a student model in the system since hints and the needed feedback are provided. Also, students may learn more from these hints. The performance of interpersonal supports is better than adaptive interaction and we suspect that one of the reasons is the effect of interpersonal motivation or peer pressure. When we compare the results of tutoring with ITS and the average of the four other systems, SD, CRT, DRT, and DRS, which involve learning by tutoring, there is some significant difference. This seems to indicate that explicit use of the student model by the student is better than hiding it from the student.

Table 5. Amount of Support Versus Average Test Scores

System	Supports	Average Score	Standard Deviation
Distributed Reciprocal Tutoring (DRT)	Interpersonal, Adaptive Interaction, Scaffolding Tool	87	14.69
Distributed Responsibility Sharing (DRS)	Interpersonal, Adaptive Interaction, Scaffolding Tool	80	5.47
Intelligent Tutoring System (ITS)	Adaptive Interaction, Scaffolding Tool	77	13.86
Self Diagnosing (SD)	Adaptive Interaction, Scaffolding Tool	81	14.62
Working Alone (WA)	Scaffolding Tool	70	15.16

5. Discussion and Future Work

At the outset of this research, we intended to build an individualized LBT system. However, due to the enormous complexities encountered in the early development, we focused on the design of a distributed social learning system, DRS, and selected the domain of acquiring programming skill as the objective of our system. We studied how to divide the programming task into sub-tasks: designing, implementing, and critiquing. Soon we discovered that the system is inevitable to have a student play the role of a critic or teacher unless there is a human or computer tutor or the domain is open-ended so that there is no need to have a teacher justify the answer or solution. Therefore, instead of searching for meaningful ways of decomposing the programming task, we focused on the more fundamental reciprocal tutoring paradigm from which the presented version of DRT naturally emerged. After analysis of data collected from students' using the DRT, we found that a student model could be used as a basis for the implementation of a centralized LBT system. Within that framework the student model enabled us to build an ITS version of the system and hence the CRT. It is interesting to note that the experience of building and using DRT to observe students' behavior before the final actual development of LBT mimicked the Wizard of Oz process.

Besides cooperative learning, *collaborative* learning, where two or more students work on the same task, without decomposition of the task, is also possible for sharing the cognitive load. Collaboration, in the end, may need powerful natural language understanding ability to understand students' dialogues. Comparing to cooperation, collaboration is a free invitation to interact while cooperation is coercive, requiring each member has to complete a part of the task to ensure that the process can move on.

During the experimental trials, two additional small trials were performed. We asked two students to use an interpreter to solve the same set of problems. Both of the students became mired in either syntax or semantics errors for the second or third problems and could not continue. The result is indicative of a more general cognitive difficulty that occurs in the individual learning process without in time feedback. In another trial two students who believed they were being asked to work cooperatively

on DRT were instead placed on two independent CRT systems. Similar tests have been performed using Distributed West (Chan, et al., 1992) and Contest-Kids systems (Chan & Lai, 1995). The results suggest that either the students trusted us and so they did not intend to distinguish or that the restricted and narrow bandwidth of communications made them unable to distinguish between the system with and without human agent. It is also possible that the student tutor does not have a good model of the computer tutee. If this is the case and the tutee is simulated by machine learning methods, the machine learning process must be made explicit and understandable to the student tutor, so that the process can be beneficial to the student's learning.

Tutoring represents a range of actions, including simple 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. Although the tutoring activities discussed here represent only a small part of this range, they are adequate to promote meta-knowledge, such as the strategic knowledge of problem solving and judgment knowledge of the performance. This type of knowledge has the potential to foster knowledge transfer to novel problems or domains. Models of this kind of cognitive development and the measurement of respective learning effects are not easy tasks. Still, the preliminary evaluation seems to indicate that reciprocal tutoring provides cognitive benefits that could possibly differ from other protocols of learning. More data and analysis are needed to answer important questions such and whether a human learning companion is better than a virtual learning companion, whether intelligent reciprocal tutoring systems are superior to intelligent tutoring systems, etc. In such analysis, social presence or absence could possibly be a critical factor.

The process of fading is not represented in our model. A fading process should be coupled with the supports offered by the environment to reflect the changing needs of students. A straightforward way of fading is to have students use systems with full supports, then leave out such supports one by one and finally students use the primitive language interpreter to perform their tasks. Our test and posttest experiments involve only the presence or absence of whatever supports the systems have to the same base environment, the interpreter. To respect the evolving learning repertoire of students, supports should not be singled out to be studied and evaluated without considering how they are faded. Nevertheless, this issue is complicated and intriguing since there are various kinds of supports in a system which are likely to require different fading techniques.

It is difficult to design game-oriented learning activities in which the student can sustain motivation while expending effort to accomplish a learning task. The preliminary evaluation indicates that the reciprocal tutoring learning activity possesses some game-like characteristics. Observation is needed to reveal these elements and how they can be incorporated into the student model to enhance its motivational effects. Indeed, motivation is desirable in all kinds of learning activities.

Reciprocal tutoring interleaves tutoring with tuteeing, or vice versa, allowing immediate use of the learned knowledge and meta-knowledge of the domain. However, learning may also proceed in two phases: tuteeing followed by tutoring. An interesting scenario involves a student who is first tutored by a computer tutor and

then trains an artificial agent so that its performance reaches a level sufficient to defeat another opponent computer agent. Reversing these two phases, that is, tutoring followed by tuteeing, is also possible. This scenario can also be regarded as a fading process since the computer tutee performs a partial modeling process for the student before the student works on the task. Our results indicate that learning by tutoring does not perform well when used alone. Therefore, this may suggest that learning by tutoring should be combined with other forms of learning to generate more effective learning paradigms.

We are currently in the process of transporting DRT and CRT to the Internet. The presence of DRT and CRT on the Internet will make the systems always available to students in their residence halls. A powerful workstation will be used as a server running the main program in Lisp. The clients will run Delphi on PC and/or SuperCard on Mac to perform the interface functions of PLS. A student on line may connect with another student in the same or different campus to play DRT. When CRT is converted to the Internet version, it means that a virtual learning companion is present on the server so that if there is no other student on-line, a virtual learning companion is always available to serve as a virtual agent to cooperate. Thus, the scenario is analogous to playing bridge on the network, where your partner can be a human or a virtual companion.

We can also observe how students' social behavior may affect their learning and the use of the systems. For example, we can examine whether students prefer to work with students they know well or whether weak students will try more than one time but sign on as anonymous users. Perhaps some of the students would have developed differently if allowed to play with the virtual learning companion before with fellow classmates. Data on student preferences can be easily captured by the server automatically without labor intensive experiments. Furthermore, integration of the real-time and interactive network version of these systems on WWW with the usual asynchronous distance learning course with the hypertext-based material for introducing basic Lisp and recursion can be made available in the same system.

We shall also develop a simple follow up version of DRT on the Internet which can also be regarded as a fading phase. Again, the student tutor can see the solution and help the student tutee to diagnose the program or to give advice by typing comments when queried by the tutee. The student tutee, at the other end, only has a menu of the language constructs as reference and key in the program solution. The learning process can be regarded as cycling processes of conceptualization, construction, reflection, and articulation. The DRT system can lead students to learn concepts of recursion in class, experiencing construction and reflection in the first phase of reciprocal tutoring. The students will then undergo articulation, construction, and reflection in the fading phase.

6. Conclusions

In this paper we have discussed the design of a set of different versions of reciprocal tutoring systems. An intelligent tutoring system is a specific type of reciprocal tutoring system. An intelligent reciprocal tutoring system can be viewed as a product

of designing and applying a sequence of devices to support student learning. These computer and human supports constrain the design and learning space, relate various elements of concern during the learning process and place these elements in proper positions in the systems.

A student model plays a critical role in supporting adaptive interaction. In traditional ITS fashion, a student model is used as a supporting component for the computer tutor (usually called the tutoring module) and is hidden from the student. When the student plays as a tutor to teach another student, the student model becomes a tool for the student tutor to manipulate directly in fulfilling the responsibility of being a tutor. Furthermore, when the computer assumes the role of a tutee, the student model serves as an active and autonomous agent inside the system. This means that simulating a typical student or a virtual learning companion is technically equivalent to student modeling; but operated in a proactive way, rather than in a reactive way as in most ITSs. The various uses of student modeling prove that a student model gets to the very soul of cooperative learning systems, dismissing doubts of its importance. (McCalla, 1990; Palthepu, Greer, & McCalla 1991; Chan, 1991). Student modeling will certainly not pass unnoticed in reciprocal tutoring or, more generally, in cooperative learning.

Placing the student in the role of tutor is a methodology common to many distributed cooperative learning in problem solving domains and is potentially applicable to all domains in which advice is needed during the learning process. The model of reciprocal tutoring is a simple and powerful idea and its significance in distributed learning environments will be as important as the intelligent tutoring system model in centralized learning environments.

Acknowledgments: The authors would like to express thanks for the support of the National Science Council of Taiwan and members of the LISA project, in particular, Ming-Hsiu Chang and Ming-Fung Lee for their help in implementing the systems and Wing-Kwong Wong for his editorial suggestions. We would also likely to thank anonymous reviewers for their suggestions and comments which lead to many improvements in the organization of this paper.

References

Aronson, E. (1978). *The jigsaw classroom.* Beverly Hills, CA: Sage.

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.

Blandford, A.E., (1994). Teaching through collaborative problem solving, *Journal of Artificial Intelligence in Education*, Vol. 5. No.1, 51-84.

Brown, A.L., (1992). Design experiments: theoretical and methodological challenges in creating complex interventions in classroom settings, *The Journal of the Learning Sciences*, 2(2), 141-178.

Brown, A.L., Ash, D., Rutherford, M., Nakagama, K., Gordon, A., & Campione, J.C. (1993). Distributed expertise in the classroom. In G. Salomon (ed.), Distributed Cognitions: Psychological and Educational Considerations, 188-288, New York: Cambridge University Press.

Brown, A.L., Campione, J.C. (1994). Guided discovery in a community of learners. In K. McGilly (ed.), *Classroom lessons: Integrating cognitive theory and classroom practice*, 229-270, Cambridge, MA: MIT Press/Bradford Books.

Brown, J.S. and Burton, R.R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 2, 155-191.

Bull, S., Pain, H. & Brna, P. (1993). Student modeling in intelligent computer assisted language learning system: The issues of language transfer and learning strategies, *Proceedings of the International Conference on Computers in Education*, Taiwan, 121-126.

Chan, T.W. (1989). Learning companion systems, Ph.D. Thesis, Computer Science Department, University of Illinois at Urbana-Champaign.

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.

Chan, T.W. (1995). A Tutorial on Social Learning Systems. In *Emerging Technologies in Education*, T.W. Chan & J. Self (eds), AACE.

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.

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.

Chan, T.W. & Chou, C.Y. (1995). Simulating a Learning Companion in Reciprocal Tutoring System, (unpublished manuscript).

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.

Chan, T.W. & Lai, J.A., (1995). Contest-Kids: a competitive distributed social learning environment. *Proceedings of World Conference on Computers in Education*, Birmingham, England, 767-776.

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.

Collins, A. & Stevens, A. (1982). Goals and strategies of inquiry teachers. In R. Glaser (ed.), *Advances in Instructional Psychology* (Vol. 2, pp.65-119). Hillsdale, NJ: Erlbaum.

Dillenbourg, P & Self, J. (1992). People power: a human-computer collaborative learning system. C. Frasson, G. Gauthier & G. McCalla (Eds.). The 2nd International Conference of Intelligent Tutoring Systems, *Lecture Notes in Computer Science*, 608, Springer-Verlag, 651-660.

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.

Luehrmann, A. (1972). Should the computer teach the student or vice-versa? *AFIPS 1972 Spring Joint Computer Conference Proceedings*, Vol. 40, AFIPS, Montvale, N.J.; Also appeared in *The Computer in the School: Tutor, Tool, Tutee*, Taylor, R.P. (ed), 1980, 129-135, Teacher College Press.

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.

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.

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

Palthepu, S., Greer, J., & McCalla, G. (1991). Learning by teaching. *The Proceedings of the International Conference on the Learning Sciences*, AACE, 357-363.

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

Scardamalia, M. & Bereiter, C. (1991). Higher levels of agency for children in knowledge building: A challenge for the deign of new knowledge media. *Journal of Learning Sciences*, 1, 37-68.

Self, J. (1985). A perspective on intelligent computer-assisted learning. *Journal of Computer Assisted Learning*, Vol. 1, 159-166.

Self, J. (1986). The application of machine learning to student modeling. *Instructional Science*, Vol. 14, 327-338.

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

Smith, D.C., Cypher, A., & Spohrer (1994). KIDSIM: Programming agents without a programming language. Communication of ACM, July, Vol 37, No. 7, 55-67.

Taylor, R.P. (1980).*The Computer in the School: Tutor, Tool, Tutee*. Teacher College Press.

Ur, S. & VanLehn, K. (1994). STEPS: A preliminary model of learning from a tutor. *The Proceedings of the 16th Annual Conference of the Cognition Science Society*. Hillsdale, NJ: Erlbaum.

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.

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