Knowledge-Based Systems 15 (2002) 407–412
www.elsevier.com/locate/knosys

An approach to developing computational supports for reciprocal tutoring[☆]

Chih-Yueh Chou[*], Chi-Jen Lin, Tak-Wai Chan

Institute of Computer Science and Information Engineering, National Central University, Chung-Li 32054, Taiwan, ROC

Received 3 July 2001; accepted 7 January 2002

Abstract

This study presents a novel approach to developing computational supports for reciprocal tutoring. Reciprocal tutoring is a collaborative learning activity, where two participants take turns to play the role of a tutor and a tutee. The computational supports include scaffolding tools for the tutor, and a computer-simulated virtual participant. The approach, including system architecture, implementations of scaffolding tools for the tutor and of a virtual participant is presented herein. Furthermore, a system for reciprocal tutoring is implemented as an example of the approach. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Reciprocal tutoring; Learning companion; Collaborative learning

1. Introduction

Reciprocal tutoring, a modified form of reciprocal teaching [1], is a collaborative learning activity, where two participants take turns playing the roles of tutor and tutee. The tutor guides the tutee in doing or learning a specific task. The participants may be a student or a computer-simulated virtual student. The computer-simulated virtual student, who collaborates with or competes against students, is known as a learning companion [2,3]. It has been proven that learning in reciprocal tutoring with a real participant is more effective than doing so in an intelligent tutoring system (ITS) [4]. It also indicates that learning in reciprocal tutoring with a learning companion is nearly as effective as learning in an ITS. In a similar study, reciprocal tutoring with a learning companion is 'found to be nearly as effective as individual tutoring by expert teachers—and considerably more effective the instruction provided in a well-taught physics class' [5]. Although, until now, the learning companion was not similar to an actual student, it had two advantages. First, students could collaborate with a learning companion any time in a reciprocal tutoring activity, without waiting or looking for another participant. Second, the system could adapt the learning companion to an individual student's status and particular tutoring strategy.

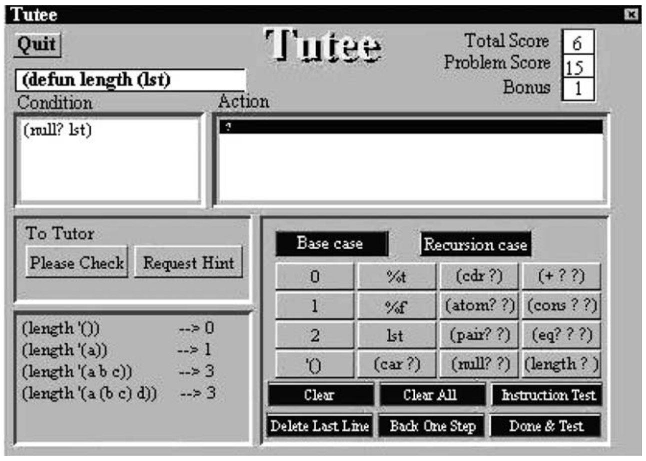
Chan and Chou also recommended that computer supports for reciprocal tutoring should include scaffolding tools for the tutor, and a computer-simulated learning companion to collaborate with the student [4]. They also implemented a system, named reciprocal tutoring system (RTS), which supports reciprocal tutoring in learning Lisp recursive program. The scaffolding tools of the tutor are interfaces that enable the human tutor to perform their required task, such as diagnosing the tutee's status and providing hints. The learning companion can perform as a tutor to guide the student according to their status. The learning companion can also act as a tutee to solve problems and to adaptively respond to various hints from a tutor.

However, how to implement computational supports for reciprocal tutoring has seldom been addressed. The implementation of learning companion within RTS is composed of two parts: virtual tutor and virtual tutee [6]. These two parts are totally independent and do not contain a sharing component. To implement the virtual tutor, a discrimination net, which contains possible solutions and hints, is employed to trace the tutee's actions and as well as to guide them. To implement the virtual tutee, decomposed solutions and an error code base are composed to simulate the solutions and responses. However, it is possible to share

[☆] This article is based on parts of a doctoral dissertation at National Central University [9].

[*] Corresponding author. Tel.: +886-3-422-7151-4805; fax: +886-3-426-1931.

E-mail addresses: yueh@src.ncu.edu.tw (C.Y. Chou), zen@src.ncu.edu.tw (C.J. Lin), chan@src.ncu.edu.tw (T.W. Chan).



```
(defun length (lst)
(null? lst)

(length '())        --> 0
(length '(a))       --> 1
(length '(a b c))   --> 3
(length '(a (b c) d)) --> 3
```

Fig. 1. The tutee interface of RTS II.

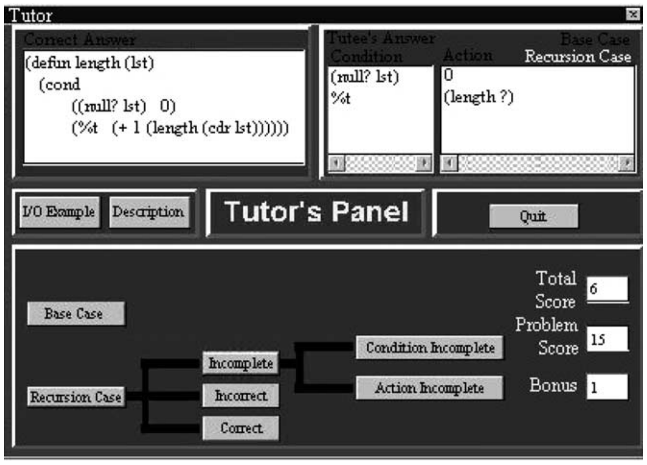
domain knowledge component between the tutor and tutee implementation. The domain knowledge includes possible solutions, errors, and hints. Building the domain knowledge into a computer is a labor-intensive work of the implementation. Thus, sharing the domain knowledge component among several implementations could reduce the cost of computational supports for reciprocal tutoring.

An approach to developing computational supports for reciprocal tutoring is proposed in this study. The approach integrates implementation of scaffolding tools for a tutor, as well as a virtual tutor and virtual tutee. These implementations share and, in different ways, make use of domain knowledge component. The approach is applied to implement second version of RTS—RTS II. The rest of this paper is organized as follows: RTS II is described, then the approach, including implementation architecture, and the method of implementing the virtual tutor and the virtual tutee are presented. Finally, a conclusion and discussion is given.

2. RTS II

The second reciprocal tutoring system (RTS II) is similar to the original version RTS. RTS is implemented with SuperCard and Lisp on MacQuadra, using AppleTalk peer-to-peer connection to link two computers. However, RTS II is implemented with Dephi, C and Lisp on IBM PC, using client–server architecture to match two computers on Internet. After logging into RTS II, a student chooses to collaborate with either a learning companion or a real student. If the student chooses a real student, the system searches online for available students and assists in matching two students. If the student chooses a learning companion, the system simulates one to collaborate with the student.

When playing the role of a tutee, the student uses a calculator-like interface [7], that is, pushes code chunk buttons to compose the recursive program (Fig. 1). If having



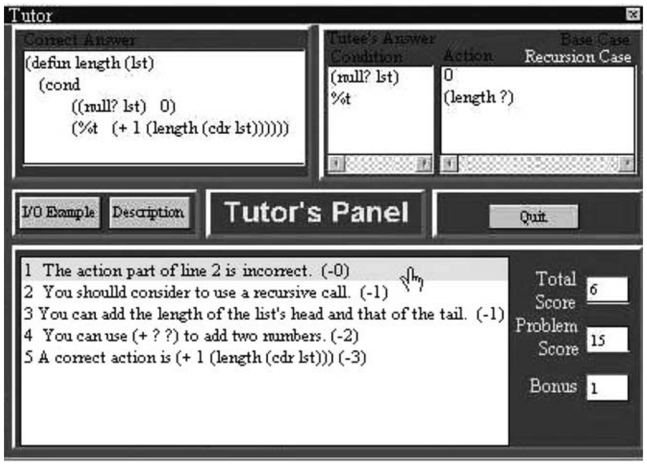
```
(defun length (lst)
  (cond
     ((null? lst)  0)
     (%t  (+ 1 (length (cdr lst))))))

(null? lst)   0
%t            (length ?)
```

Fig. 2. The tutor interface of RTS II (I).

some question in solving problem, the student can ask the tutor for hints. After composing one line of program, the tutee must ask the tutor for guidance. The design prevents the student from going too far in inaccurate solving way. When the program is completed, the student can run it to verify whether the solution is correct or not.

When acting as a tutor, a student can observe the tutee's solution and correct solution (see Fig. 2). If the tutee requests for hints, the student must diagnose the tutee's situation through a scaffolding tool, diagnosis-hint-tree (DHT). DHT is a series of branches with several mutually exclusive situation buttons. Each situation button presents a specific recursive program situation. If the tutor pushes the situation button that matches the tutee's situation, DHT will expand to the next branch and reveal relevant situation buttons. Otherwise, the system notifies the tutor having an inaccurate diagnosis. After expanding the DHT to identify the tutee's program situation, the tutor chooses a hint from a provided list and then forwards the hint to the tutee (Fig. 3). Notably, a score system was designed to prevent the tutor from depending on DHT too much. The tutor receives bonus scores if DHT is expanded without inaccurate diagnosis.



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

(null? lst)   0
%t            (length ?)
```

1 The action part of line 2 is incorrect. (-0)
2 You shoulld consider to use a recursive call. (-1)
3 You can add the length of the list's head and that of the tail. (-1)
4 You can use (+ ? ?) to add two numbers. (-2)
5 A correct action is (+ 1 (length (cdr lst))) (-3)

Fig. 3. The tutor interface of RTS II (II).

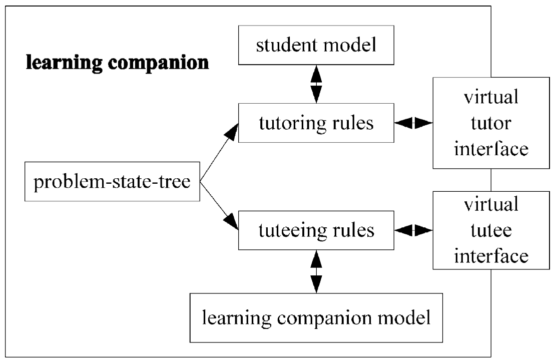


Fig. 4. Implementation architecture of learning companion.

The score system also prevents the tutee from depending on the tutor too much. Hints with more detail decrease more score.

3. An approach of developing computational supports

3.1. Implementation architecture

In the implementation of the learning companion, a problem-state-tree component is shared in simulating the tutor role and the tutee role (see Fig. 4). A problem-state-tree contains domain knowledge about a recursive program, including possible solutions, error, and relevant hints. To implement the virtual tutor, the system uses the problem-state-tree to trace the student's status, to construct student model, and to tutor the student adaptively. Tutoring rules component controls the earlier tutoring simulation. The tutoring rules component receives the tutee's action from interface component, the domain knowledge from problem-state-tree component, and previous status from student model component. Then, the system estimates the tutee's current status, updates the student model and determines a response according to the tutoring rules. The scaffolding tool for tutor, DHT, is a by-product of implementing the virtual tutor. One implementing requirement of the virtual tutor is to diagnose the tutee's status and provide adaptive hints. The implementation can be employed to develop a scaffolding tool, which assists student playing the role of a tutor.

To implement the virtual tutee, the system makes use of the problem-state-tree to compose the program, to request hints and to adaptively respond to various hints. Tuteeing rules component controls the earlier simulation by receiving hints form interface, domain knowledge from problem-state-tree, and learning companion's domain proficiencies from learning companion model as parameters. The detailed implementation of virtual tutor and virtual tutee is described in the following sections.

3.2. Implementing the virtual tutor

In RTS II, the virtual tutor is an expert. Therefore, implementing the virtual tutor is similar to implementing one in an ITS. Implementing the virtual tutor for each problem requires three system components: (1) a problem-state-tree, (2) a student model, and (3) a set of tutoring rules. These three components trace the tutee, build a student model and DHT, and provide adaptive hints.

For each problem, a problem-state-tree is established that is based on the correct solution and the errors that were collected previously where students used PLS without a tutor. The problem-state-tree is a typical node-link data structure that stores problem data (Fig. 5). Furthermore, it contains three kinds of nodes: problem-state-node, hints-node, and operations-node. Each problem-node represents a possible situation of the student's program. Each hints-node relates to a certain problem-state-node. The hints-node stores numerous hints related to the problem-state-node, including each hint's type, content, related domain concepts, and score. An operations-node stores several operations, their related domain concepts, as well as their connected problem-state-node.

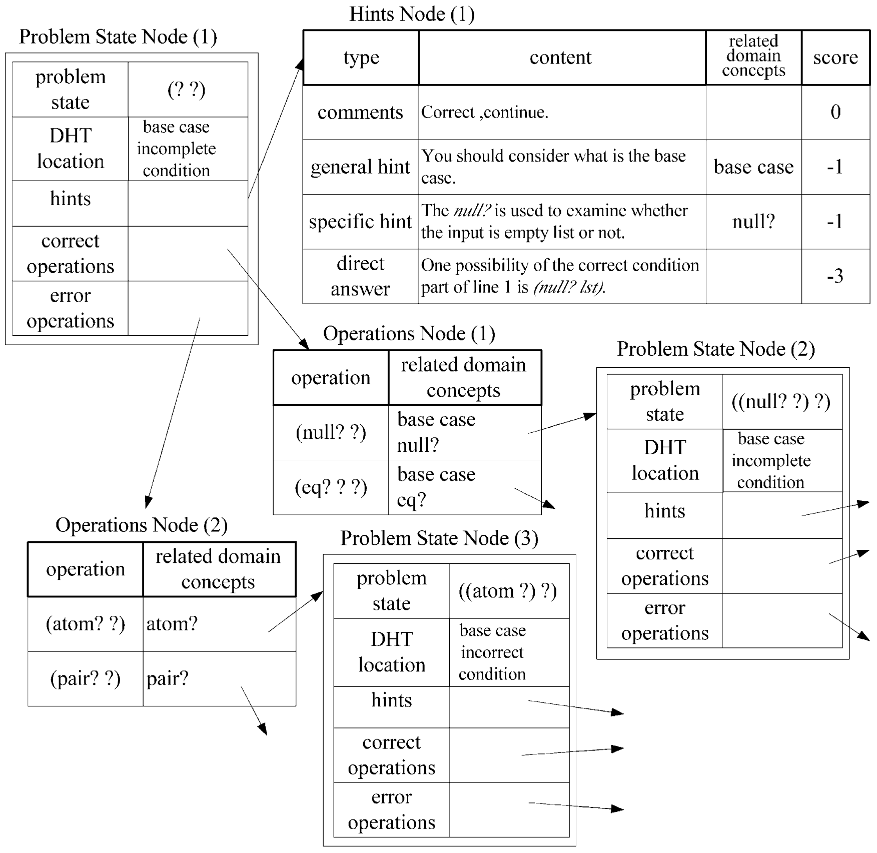
3.2.1. Tracing the tutee and building student model

To implement a virtual tutor, like an ITS, the RTS II traces the tutee's solution and builds the tutee's student model. This model stores the system-estimated proficiencies that are based on the domain concepts of the tutee. The system compares the tutee's solution with a problem-state-tree to interpret the tutee's actions. This is a simple artificial intelligence technique, which is known as differential modeling [8]. For example, in Fig. 5, the tutee's initial program situation is (? ?), and the system identifies that the tutee's state is at problem-state-node (1). If the tutee pushes the code chunk (null? ?), the system traces that the tutee go through operations-node (1) to reach the problem-state-node (2). In addition, the tutee applies the operation (null? ?) correctly, and thus the system increases the proficiency of related domain concepts, *base case* and null?, in the tutee's student model. To represent the student's degree of proficiency in a particular concept, the student model uses a number between 0 and 1 for each domain concept.

The next possible error operations are (atom? ?) and (pair? ?) and the related domain concepts are pair? and atom?. When the correct solution is not chosen and code chunk (pair? ?) is used, it implies that the tutee lacks the proficiency related to null?, eq?, and *base-case*. Therefore, the system increases the degree of proficiency for related domain concepts, pair? and decreases it for domain concepts null?, eq?, and *base-case* in the tutee's student model. Through this method, the system can trace a tutee and develop the tutee's student model.

3.2.2. Building a diagnosis-hint-tree

When a student requests a hint, the tutee's program is


Problem State Node (1)

problem state	(? ?)
DHT location	base case incomplete condition
hints	
correct operations	
error operations	

Hints Node (1)

type	content	related domain concepts	score
comments	Correct ,continue.		0
general hint	You should consider what is the base case.	base case	-1
specific hint	The *null?* is used to examine whether the input is empty list or not.	null?	-1
direct answer	One possibility of the correct condition part of line 1 is *(null? lst).*		-3

Operations Node (1)

operation	related domain concepts
(null? ?)	base case null?
(eq? ? ?)	base case eq?

Problem State Node (2)

problem state	((null? ?) ?)
DHT location	base case incomplete condition
hints	
correct operations	
error operations	

Operations Node (2)

operation	related domain concepts
(atom? ?)	atom?
(pair? ?)	pair?

Problem State Node (3)

problem state	((atom ?) ?)
DHT location	base case incorrect condition
hints	
correct operations	
error operations	

Fig. 5. A problem-state-tree.

matched with the problem-state-tree. On termination, the current situation of the tutee's program is recognized and the relevant hints are presented to the tutor. Each node stores hints and as well as the DHT location for the tutor to use. For example, according to the DHT location of problem-state-node (1) in Fig. 5, the hints are located in 'base case'–'incomplete'–'condition' (Fig. 6). The system provides hints for the human tutor who then chooses one for the tutee. Via the schema (a single 'cond') that the user's interface constrains, the latitude of the problem is narrowed to one or two possible solutions, so the problem-state-tree is rather linear. However, this approach is limited in that each problem requires a problem-state-tree, which is labor intensive. Nevertheless, this problem-state-tree enables the computer to indicate errors and supply hints about how to proceed to solve a particular problem.

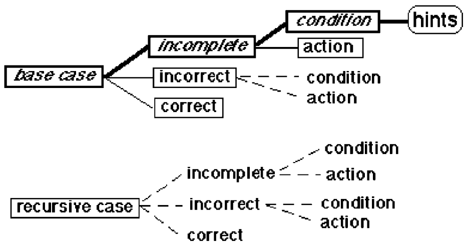


Fig. 6. The diagnosis-hint-tree structure.

3.2.3. Giving adaptive hints

A hint in DHT, although in natural language form, can be classified according to its content type and related concept. In RTS II, there are four types of DHT hints: comments, general hints, specific hints, and direct answers. Comments are remarks on a current status's accuracy. For example, 'Correct, continue' or 'The action part of line 2 is incorrect'. General hints are generic guides to the error or directions for subsequent steps. For instance, "An endless recursion call will occur" and "You should consider what is the base case in this problem". Specific hints suggest a specific domain concept. That is, 'The null? is used to examine whether the input is empty list or not'. Direct answers simply provide one of many possible correct solutions. For example, 'One possibility of the correct condition part of line 1 is (null? lst)'. Every hint has four properties: type, content, related domain concepts, and score. The hint's related domain concepts can help determine whether the hints are appropriate or not. If a hint is concerned with the domain concept null? and the tutee's proficiency degree on null? is high according to the student model, the hint is not appropriate. Section 3.3 illustrates that the hint's related domain concepts are also effective in simulating a virtual tutee's response to the tutor's hint. When a tutee asks for help and the tutor provides hint, their score decreases. Each hint shows the amount it decreases the score. As stated more detailed hints further decrease a score. This score system

Table 1
Components for implementing virtual tutor and virtual tutee

Component	Implementing the virtual tutor	Implementing the virtual tutee
Problem-state-tree	(1) Provide information for student modeling. (2) Provide information to give tutee hints	(1) Provide information for problem solving. (2) Provide information to respond to tutor's hint
Student model	Store detected student's status	–
Learning companion model	–	Represent companion's degree
Set of tutoring rules	Simulate the tutor's action	–
Set of tuteeing rules	–	Simulate the tutee's action

prevents the tutor from providing a direct answer too quickly.

The principle that the tutoring rule of the virtual tutor in RTS II adopts is to provide hints that are initially general, but, if required, become specific. That is, the hints intensify from comment, general hint, and specific hint, to direct answer. If the hint has related domain concepts, the system will consult the tutee's student model to determine whether the hint is appropriate or not. Therefore, the virtual tutor can provide adaptive hints.

3.3. Implementing the virtual tutee

For each problem, virtual tutee implementation involves three components: (1) a problem-state-tree, (2) a learning companion model, and (3) a set of tuteeing rules. The problem-state-tree supports both for implementing the virtual tutee and the virtual tutor. The learning companion model stores the learning companion's degree of proficiency in domain concepts and, hence, the companion's competence level. Each proficient degree of domain concept is a number between 0 and 1. A degree of 0 implies that the virtual tutee knows nothing about the concept. However, a degree of 1 implies that the virtual tutee has mastered this concept. The data structure and stored domain concepts in the learning companion model are the same as that of student model. Rather than applying machine learning methods to empower the learning ability of the virtual tutee, an overlay approach was adopted to pretend the virtual tutee to learn at roughly the level of an average student.

Initially, the proficiency degree of domain concepts in the learning companion model is performed on a trial and error method. That is, through testing the behavior of the virtual tutee, responses are determined reasonable for an average novice learner. Tuteeing rules are heuristic rules that govern the responses of the virtual tutee. The rules list as follows:

(I) From the problem-state-tree, locate the virtual tutee's program state (initial situation is (? ?)). Then locate all the correct operations that can enable passage to the next problem state, and compute these operation's grade according to the learning companion model. If the problem is completed, terminate. For example, if the virtual tutee's proficiency degree of 'null?' is 0.8 and that of 'base-case' is 0.4, the proficiency degree of operations (null? ?) may be calculated by the average of 'null?' and 'base-case'; that is, 0.6.

(II) Use a correct-threshold number to verify the correct operation with the highest grade to determine what the virtual tutee would do. There are four possible responses:

(IIa) Select the correct operation and display the correct code chunk. The system locates the correct operation with the highest proficiency grade from the problem-state-tree. If the proficiency degree of the operation is higher than the correct-threshold number, the virtual tutee selects the correct operation. Then go to (I).

(IIb) Make mistakes. The system locates the error operation with lowest proficiency degree. If the proficiency degree of the operation is lower than an error-threshold number, the virtual tutee make mistakes and selects the error code chunk, then return to (I).

(IIc) Ask tutor to verify results. If the virtual tutee completes a condition clause, a human tutor is asked to check the clause, for which DHT is used. If the clause is correct, the tutor instructs the virtual tutee to continue, and then return to (I). Otherwise, the tutor identifies the error code chunk, causing virtual tutee to backtrack, revise the whole clause, then return to (I).

(IId) Request assistance. Keep asking for hints until the hint indicates to the related concept with lowest proficiency degree. Display the code chunk and increase the proficiency degree of that concept in the learning companion model, then return to (I).

4. Conclusion and discussion

Herein, an approach to developing computational supports for reciprocal tutoring was presented. The computational supports include scaffolding tools for a tutor and a virtual participant, which can in turn be a tutor and a tutee to collaborate with the student in a reciprocal tutoring activity. The approach includes system architecture, scaffolding tools for the tutor, and a virtual participant. RTS II, a system for reciprocal tutoring, is implemented in the approach and provides an example thereof.

Implementing the virtual tutor and virtual tutee require several components (see Table 1) and share the problem-state-tree component, which integrates domain knowledge of each program. However, they employ the problem-state-tree to obtain the required information for various purposes. Implementing the virtual tutor, the system obtains information from the problem-state-tree to create a student model and provide hints. Alternately, in implementing the virtual tutee, the system acquires information from the problem-state-tree for problem solving and to respond to hints. With the same data format of storing domain proficiencies, the system estimates the student model data to understand the student, however, the system designer establishes the data of the learning companion model to represent the learning companion. The data corrected in student model can be employed in establishing the learning companion model. Tutoring rules and tuteeing rules are core of implementation of virtual tutor and virtual tutee, respectively. However, designing these two rules both requires knowledge of tutoring strategy to design interaction between tutor and tutee.

Notably, the virtual tutee consistently advances due to decoupling the learning companion model from the data (problem-state-tree) to construct the responses. Adjusting domain proficiencies in the learning companion model can alter the behavior of the virtual tutee. Moreover, this decoupling allows various problems to be independent of the virtual tutee's behavior and thus allows new problems to be added easily.

Acknowledgments

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract no. NSC89-2520-S-008-011. Shi-Sheng Yang and Ruei-Liou Huang are also appreciated for their assistance in implementing the proposed system.

References

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

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

[3] T.W. Chan, Integration-kid: A Learning Companion System, The Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, Morgan Kaufmann Publishers, Los Altos, CA, 1991, pp. 1094–1099.

[4] T.W. Chan, C.Y. Chou, Exploring the design of computer supports for reciprocal tutoring, International Journal of Artificial Intelligence in Education 8 (1997) 1–29.

[5] L.A. Scott, F. Reif, Teaching scientific thinking skills: students and computers coaching each other, The 9th International Conference on Artificial Intelligence in Education (AI-ED 99), Le Mans, France, 1999, pp. 285–293.

[6] T.W. Chan, C.Y. Chou, Simulating a learning companion in reciprocal tutoring system, The Proceedings of the First International Conference on Computer-Supported Collaborative Learning, 1995, pp. 49–56.

[7] S. Bhuiyan, J.E. Greer, G.I. McCalla, Learning Recursion through the use of a Mental Model-Based Programming Environment, The Second International Conference of Intelligent Tutoring Systems, Lecture Notes in Computer Science, vol. 608, Springer, Berlin, 1992, pp. 50–57.

[8] R.R. Burton, J.S. Brown, An investigation of computer coaching for informal learning activities, International Journal of Man–Machine Studies 11 (1979) 5–24.

[9] C.Y. Chou, An approach to modeling multiple learning companions in problem solving activities, Doctoral Thesis, Institute of Computer Science and Information Engineering, National Central University, Taiwan, 2000.