INSTRUCTIONAL PLANNING IN AN INTELLIGENT TUTORING SYSTEM: COMBINING GLOBAL LESSON PLANS WITH LOCAL DISCOURSE CONTROL

BY

CHONG WOO WOO

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
in the Graduate School of the
Illinois Institute of Technology

Approved____________________
Advisor

Chicago, Illinois
December, 1991

ACKNOWLEDGEMENT

I am deeply grateful to my thesis advisor, Dr. Martha Evens, for her intellectual support, endless encouragement, understanding and patience throughout the course of this study. I feel extremely fortunate to have her as my advisor. Thank you Martha. I also would like to express my sincere appreciation to Dr. Allen Rovick and Dr. Joel Michael of Rush Medical College for their expert advice and guidance during this work.

To my thesis committee members, Dr. C. Robert Carlson, Dr. James Kenevan, and Dr. Clayton Hall, thank you for the attention and suggestions you gave on my thesis.

I would like to say thanks to my colleagues in the project group for their invaluable contributions, comments, and cooperations. Thank you all.

Finally, I would like to thank my parents for their love, inspiration, and support that they have provided. I will be eternally grateful. To my fiancee, SungEun, I would like to express deepest affection and love for her understanding and confidence in me.

This work was supported by the Cognitive Science Program, Office of Naval Research under Grant No. N00014-89-J-1952, Grant Authority Identification Number NR4422554 to Illinois Institute of Technology. The content does not reflect the position or policy of the government and no official endorsement should be inferred.

C. W.

TABLE OF CONTENTS

		Page
ACKNOWLEDGEMENT		iii
LIST OF FIGURES		vi
CHAPTER		
I.	INTRODUCTION	1
	1.1 An Overview	1
	1.2 Evolution of Computer-Based Instruction at Rush	3
	1.3 Goals of the Thesis	4
	1.4 Organization of the Thesis	6
II.	THE BACKGROUND	9
	2.1 Qualitative Reasoning	9
	2.2 Subject Area	10
	2.3 Organization	12
	2.4 System Constraints	14
	2.5 Multiple Simultaneous Inputs	15
III.	ORGANIZATION OF CIRCSIM-TUTOR	18
	3.1 Intelligent Tutoring Systems	18
	3.2 Domain Expertise	23
	3.3 Input-Understander	26
	3.4 Student Modeler	27
	3.5 Instructional Planner	28
	3.6 Text Generator	29
	3.7 Screen Manager	29
	3.8 Summary	30
IV.	SURVEY OF PLANNING IN TUTORING SYSTEMS	32
	4.1 Approaches to Planning	33
	4.2 Approaches to Instructional Planning	39
	4.3 Review of ITSs	43
	4.4 Summary	49
V.	PLANNING INSTRUCTION	51
	5.1 Design Issues	52
	5.2 Scenario	55
	5.3 Organization of the Instructional Planner	61
	5.4 Summary	66

CHAPTER Page

VI. THE LESSON PLANNER 67

6.1 Implementation Goals 68
6.2 Lesson Planning Rules 69
6.3 Lesson Planning 72
6.4 An Example 77
6.5 Summary 81

VII. THE DISCOURSE PLANNER 82

7.1 Implementation Goals 82
7.2 Architecture of the Discourse Planner . . 84
7.3 Discourse Planning 91
7.4 Trace of Discourse Transitions 96
7.5 Summary 98

VIII. CONCLUSION 100

8.1 Significance of this Research 100
8.2 Future Research 102

APPENDIX

A. TUTORING RULES IN ENGLISH AND IN LISP CODE . . 104

B. TRACE OF A TUTORING SESSION 120

C. DIALOGUE PRODUCED BY THE SYSTEM 125

BIBLIOGRAPHY 132

LIST OF FIGURES

Figure Page

1. The Concept Map 11
2. List of Available Procedures 13
3. The Prediction Table 13
4. ICAI Domains 19
5. The Structure of Our System 21
6. A Frame from the Domain Knowledge Base 25
7. Planning System 33
8. Organization of a Classical Planning System . . . 33
9. INTERPLAN Backtracking Problem 35
10. Organization of a Dynamic Planning System . . . 38
11. Instructional Planner 63
12. Structure of the Lesson Planner 67
13. Goal Generation Rules 73
14. Generated Lesson Goals in the Goal Stack 73
15. The Strategy Rules 75
16. The Tactical Rules 76
17. Generated Plan for "causal_relation (RAP,SV)" . . 76
18. The Subgoal Stack 77
19. An Example of Lesson Planning 78
20. A Pseudo Code for Lesson Planning 80
21. The Flow Chart for Tutoring Non-Neural Variables in DR 86
22. The Discourse Network 87
23. The Pedagogic Default Rules 89

Figure Page

24. Some Pedagogical Meta Rules 89

25. Tactical Default Rules 90

26. Tactical Meta Rules 90

27. Pseudo Code for Discourse Planning 93

28. Trace of Discourse Transition Process 97

CHAPTER I

INTRODUCTION

1.1 An Overview

The goal of this research is the design and development of an instructional planner that is responsible for determining what to do next at each point during a tutorial session. The planner is a central component of an intelligent tutoring system (ITS) being developed as a joint project of Rush Medical College and Illinois Institute of Technology. The goal of this research is to develop an ITS, CIRCSIM-TUTOR, that assists first year medical students to learn the the behavior of the cardiovascular reflex system that stabilizes blood pressure. Since the students have already attended lectures about the domain, CIRCSIM-TUTOR assumes prerequisite knowledge and assists them to correct their misconceptions in the problem solving.

At any time in the tutoring session, the planner has to decide what subject matter to focus on, how to present it to the student and when to interrupt the student's problem-solving activity [Dede, 1986; Kearsley, 1987]. For example, the planner has to decide *whether to ask a question, introduce a new topic, remediate a misconception, etc.*, during the tutoring session. This pedagogical decision making is very complex and there is no one correct choice due to the dynamic changes in the student's learning state. Hence, the decision must be based on many different knowledge sources; knowledge about the domain, knowledge about the student, and pedagogical knowledge.

Recent approaches to designing tutoring systems view the decision making process as a planning problem [Peachey and McCalla, 1986; Macmillan et al., 1988; Brecht et al., 1989; Murray, 1990]. Adaptive planning techniques in the tutoring domain enable the generation of customized plans for individualized instruction. Among the recent research systems, MENO-TUTOR [Woolf, 1984] represents an important attempt at planning the discourse strategies observed in human tutors, but it lacks global lesson goals [Murray, 1988]. This lack of global lesson goals limits the ability of a system to generate globally coherent and consistent instruction during the tutoring session. IDE-INTERPRETER [Russell, 1988] is another attempt at planning the lesson goals at various levels of abstraction, but this system lacks power at the local diagnostic level. Thus, there is a need to build an instructional planner that combines globally coherent lesson goals with flexible local discourse plans.

In this research, I am building a planner that integrates opportunistic control with sophisticated planning methods; combining capabilities of lesson planning with discourse planning. This planner is a dynamic instructional planner that supports customized, globally coherent planned instruction, supports mixed initiative strategy, and has the capability for replanning. This has required the invention of multi-level instructional planning.

1.2 Evolution of Computer-Based Instruction at Rush

Computer Aided Instruction (CAI) in the cardiovascular domain at Rush Medical College has evolved from HEARTSIM [Rovick and Brenner, 1983], to CIRCSIM [Rovick and Michael, 1986], to the CIRCSIM-TUTOR prototype [Kim et al., 1989] and finally to CIRCSIM-TUTOR over the last ten years.

HEARTSIM was a Plato program and CIRCSIM is a stand-alone Basic program. The CIRCSIM-TUTOR prototype is a Prolog prototype of our ITS designed and implemented by Kim [1989]. Its design is based on major ITS architecture, which includes an expert module, a student model, a planner, and a communication module. However, the prototype system still does not possess all of the capabilities needed for an ITS. It lacks natural language capabilities, it does not analyze the student's misconceptions, and the instructional planner is very primitive; a discourse planner could not be implemented since complete discourse strategies for all the primitive actions had not been developed, planning knowledge is not explicitly represented as a separate module, and there was no replanning capability so that the system could not respond to student initiatives.

CIRCSIM-TUTOR uses the same architecture as Kim's prototype but includes complete student modelling, instructional planning, and natural language understanding and generation facilities. The role of my research is to design and build a sophisticated instructional planner for CIRCSIM-TUTOR, which can support all of the limitations of the earlier systems.

1.3 Goals of the Thesis

The instructional planner in CIRCSIM-TUTOR has several novel features.

First, the planner employs two different kinds of instructional planning mechanisms: lesson planning and discourse planning. Lesson planning is further divided into goal generation, planning of strategies, and planning of tactics to refine the goal into subgoals. Discourse planning is implemented using a two level approach: pedagogical decision making at the upper level and tactical discourse state-based planning at the lower level. By combining these two planning mechanisms, the planner can provide both globally coherent instruction and flexible discourse response to the student throughout the tutoring session.

Second, the planner has a dynamic planning capability; it can generate plans, monitor the execution of the plans, and replan when the student interrupts with a question during the tutoring session. The planner is dynamic; it generates new plans and replans when necessary. By planning instruction dynamically based on the inferred student model, the planner can provide more adaptive instruction than the unplanned instruction produced by CAI systems.

Third, the pedagogic knowledge is represented explicitly as a set of rules, which allow the planner to fine tune the plans dynamically and to modify the plans easily, rather than requiring the human author to anticipate the plans. These rules are used to generate lesson goals, strategies, and tactics, and discourse management. The system interprets the rules and comes back with an appropriate response to interact with the students.

Fourth, the planner plans at different levels of the hierarchy. This hierarchical planning technique reduces the complexity of the planning process. This top-down plan expansion technique has been implemented in several ITS systems [Murray, 1990; Russell, 1988].

Fifth, the planner supports a mixed-initiative strategy by allowing *student initiatives* during the tutoring session. The planner needs to do replanning after it carries out the student's request. We are currently investigating strategies for responding to the student's initiative, and implementing somewhat primitive responses.

Finally, the planner is based on cognitive science research into transcripts of human tutoring interactions. From these transcripts, we extracted some possible strategies and tactics, which we employ as heuristics in generating the content of lesson plans and discourse strategies.

This thesis describes the implementation of the above features in detail. CIRCSIM-TUTOR is written in Procyon Common Lisp and runs on a Macintosh IIci computer.

1.4 Organization of the Thesis

This thesis is divided into eight chapters. Chapter II describes the background of the system. The subject area of CIRCSIM-TUTOR is cardiovascular physiology and the system assists students to understand the behavior of the complex negative feedback system. Then the chapter explains the overall organization of the system, some important constraints, and the effect of simultaneous student inputs.

Chapter III begins with a brief introduction to ITS: the general structure and the issues involved in each module of the ITS. Then I describe each component of CIRCSIM-TUTOR briefly introducing functions and data structures.

Chapter IV presents a survey of literature related to the study of Artificial Intelligence planning techniques and the application of planning techniques in ITSs. Teaching strategies, control mechanisms and chronological progress are the main foci of this discussion. I also review some well known ITSs with their contributions and limitations from the planner's point of view.

Chapter V presents design issues for building the planner: levels of planning and tutoring strategies. A short tutoring session is displayed, which came from a transcript of human tutor and student interaction. And then a short scenario is described to explain how the system works. The chapter concludes with a discussion of the overall organization of the planner: lesson planning, discourse planning, and plan monitoring.

Chapter VI discusses the lesson planner. It first discusses the main features of the planner: goal generation and plan generation. Each phase uses its own lesson planning rules: goal generation rules and plan expansion rules. The results of applying the rules are saved in stacks: a goal stack and a subgoal stack. This chapter explains the generation of the content of lesson plans in detail.

Chapter VII discusses the discourse planner. The discourse planner controls the interaction between the tutor and the student. The structure of the planner is a two level discourse management network, which consists of a set of states that represent tutorial actions. The control mechanism is separated into default and meta-rule transitions. The chapter ends with a short trace of discourse transitions.

The thesis concludes in Chapter VIII with a discussion of the significance of the planner, describes some of its limitations, and gives suggestions for future research.

CHAPTER II

THE BACKGROUND

2.1 Qualitative Reasoning

Qualitative reasoning or simulation [deKleer and Brown, 1984; Forbus, 1984; Kuipers, 1984] is an approach to problem solving that reasons about the causal relationships that structure our world. Qualitative simulation is a kind of qualitative reasoning and qualitative reasoning operates on a qualitative model. Forbus [1984] explains qualitative reasoning with a steam boiler example: If the input temperature is increased, what happens to the output temperature? Anderson [1988] argues that qualitative reasoning is the most demanding approach and essential to produce a high performance tutoring system. He states that qualitative modelling can maximize the pedagogical effectiveness since it is human-like reasoning, although the implementation effort is much larger than that required for the traditional black box models or glass box models.

Among the recent research efforts, deKleer and Brown's approach is interesting because it evolved within the last phase of the SOPHIE project [deKleer and Brown, 1984]. Their approach is referred to as a component centered approach [Cohn, 1987], where a system is modelled by instantiating components from a library, which are then connected together explicitly. The relationships between the components are called confluences, and the entire system is modeled by a set of confluences; cause and effect relationships or constraints among the components. For example, assume a system that consists of a set of components, such as a valve, an amount of water, and a pressure. The system is originally in an equilibrium state, and a disturbance is introduced (water flow), then constraints are propagated until a new equilibrium is reached.

This causal process may construct reasonable causal explanations of how the system works. These problems are similar to those of CIRCSIM-TUTOR, where a perturbation occurs in a component of the system and the qualitative changes propagate until the system again reaches an equilibrium state. Another similarity is in its qualitative quantity space, where {-, 0, +} are used to represent the qualitative values (- represents a drop in the parameter value, 0 no change, + an increase).

2.2 Subject Area

CIRCSIM-TUTOR is an approach to qualitative simulation in cardiovascular physiology [Michael et al., 1990]. It is designed to teach first year medical students about the negative feedback system that controls the blood pressure. The cardiovascular system consists of many mutually interacting components, and the student must understand the cause and effect relationships for each individual component of the system. Figure 1 shows a causal model of CIRCSIM-TUTOR, called the *Concept Map*, designed by Michael and Rovick [Kim et al., 1989]. Each box in the map represents a physiological variable, such as SV for Stroke Volume and RAP for Right Atrial Pressure. An arrow with "+, -" sign between two boxes tells the direction of the causal effects and whether the causal relationship between the connected variables is direct or inverse. For example, a qualitative change in one component of the system, a decrease in RAP, directly causes a decrease in SV. This qualitative change propagates to other adjacent components of the system according to the propagation rule.

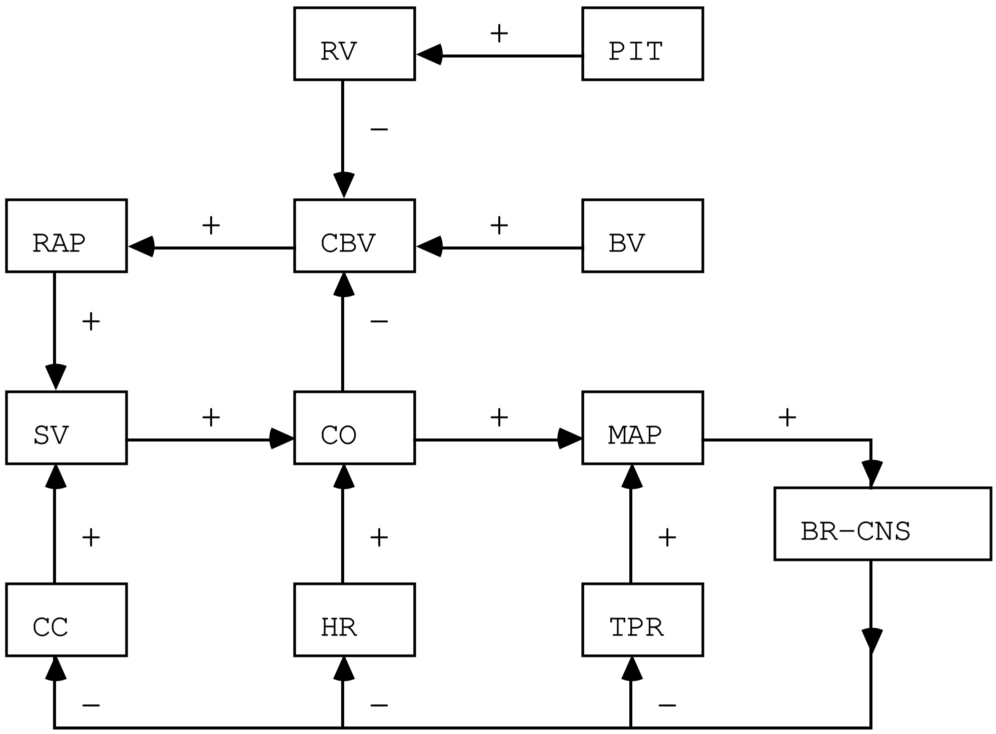

Figure 1. The Concept Map

There are three stages in the human body's response to a perturbation in the system that controls blood pressure. The first stage is the Direct Response (DR) in which a perturbation in the system will physically affect many other parameters. The second stage is the Reflex Response (RR), in which other parameters are affected by the negative feedback mechanism to stabilize the blood pressure. The final stage is the Steady State (SS), which is achieved as a balance between the changes directly caused by the initial perturbation and the further changes induced by negative feedback.

2.3 Organization

CIRCSIM-TUTOR begins with a brief introductory message and then asks the student to choose any procedure from the curriculum list. The curriculum (Figure 2) is stored as a set of seven different experimental procedures designed by our expert human tutors (JAM and AAR). Each procedure begins by describing a perturbation of the cardiovascular system, and asking the student to predict how the system variables will respond to the perturbation by making qualitative entries in the Prediction Table (see Figure 3); using a "+" sign to represent an increase, a "-" for a decrease, and "0" to indicate no change. The first column of the table is used to predict the Direct Response (DR) of each variable to the perturbation, the second is used for the Reflex Responses (RR), and the third for the Steady State (SS).

List of Available Procedures

1. Decrease Arterial Resistance (Ra) to 50% of Normal.
2. Denervate the Baroreceptors.
3. Decrease Ra to 50% of Normal in a Denervated Preparation.
4. Hemorrhage: Remove 1.0 Liter of Blood.
5. Decrease Cardiac Contractility (CC) to 50% of Normal.
6. Increase Venous Resistance (RV) to 200% of Normal.
7. Increase Intrathoracic Pressure (PIT) to 2 mmHg.
8. Quit.

Figure 2. List of Available Procedures

Parameters	DR	RR	SS
Cardiac Contractility	0		
Right Atrial Pressure	–		
Stroke Volume	–		
Heart Rate	0		
Cardiac Output	–		
Total Peripheral Resistance	0		
Mean Arterial Pressure	–		

Figure 3. The Prediction Table

When the student finishes predicting all seven parameters in one column of the table, for example the DR stage, the student's answers are compared with the correct answers. If the student has made any errors, a natural language tutoring session will begin, based on the result of this evaluation in order to correct the student's misconceptions.

2.4 System Constraints

There are some system variables that need to be described; the procedure variable is the variable changed by the perturbation; the primary variable is the first variable in the Prediction Table affected by the procedure variable, (in some cases the procedure variable is the primary variable); the neural variables are the variables directly under nervous system control. The rest of the variables we call physical variables. The students are not allowed to predict the variables in any arbitrary order, since there are some constraints that they must follow. For example, the constraints for DR are fairly complex:

Constraint DR1: The student must predict the primary variable first, and the value must be correct.

Constraint DR2: The student must predict the physical variables in the correct causal sequence.

Constraint DR3: The student may predict the neural variables at any time and in any order.

The student receives a canned error message, when either of the first two constraints is violated, and is told what to do next. The purpose of forcing the student into the correct sequence is to make sure the causal behavior of the system is followed correctly. Neural variables can be entered at any time since neural variables do not change during the DR period except when one is a primary variable. The constraints for the RR stage are designed to teach the students about the effect of the baroreceptor reflex:

Constraint RR1: The student must predict either the neural variables or MAP first.

Constraint RR2: The student must finish predicting all the neural variables before predicting other physical variables.

Constraint RR3: The student must predict the physical variables in the correct causal sequence.

Finally, when predicting the SS stage, the student is allowed to enter predictions in any arbitrary order since there are no specific constraints for this stage.

2.5 Multiple Simultaneous Inputs

In a mixed-initiative type of ITS, the tutor and the student share control over what happens next during a tutoring session. Generally, in these systems the tutor begins by posing a question and the student either responds to the question or takes the initiative. Sometimes this style of tutoring leaves students confused and frustrated if they do not have enough background in the domain knowledge, even though there exists some type of tutoring strategy that prevents students from getting too far off the track [Reiser, 1989]. Rather than blindly walking through the domain, it is much more effective if the tutor provides a simulated problem situation in the domain for the student before the actual interactive tutoring begins.

CIRCSIM-TUTOR begins with a Prediction Table, in which the student is asked to make qualitative predictions about the behavior of the system given a particular perturbation. After the student finishes all the predictions, the tutor analyzes the student's answers and shows what errors were made if any. Based on a careful analysis of these errors, the tutor can generate a global lesson plan, and interactive tutoring begins by using a mixed-initiative Socratic strategy in natural language. Thus, the Prediction Table provides a qualitative simulation environment for the student by requiring multiple simultaneous inputs (multiple responses to different aspects of a problem provided by the student in a single uninterrupted turn) before interactive tutoring begins.

There are several benefits of adapting this kind of design strategy. First, the tutor receives enough initial knowledge about the student so that it can narrow the focus for tutoring. It can also detect some common student misconceptions [Michael et al., 1991] or *bugs*. Second, the students can see a simple mental model of the entire domain at the start, which prevents the students from getting too far off the track. Elsom-Cook [1988] argues that using multiple pedagogic strategies can provide a very powerful learning environment. CIRCSIM-TUTOR begins with a coach-like environment during the Prediction Table entry, and then moves to Socratic tutoring for the interactive tutoring session. This flexibility in adapting to the student's needs at different stages provides another benefit.

CHAPTER III

ORGANIZATION OF CIRCSIM-TUTOR

The goal of constructing an Intelligent Computer Aided Instruction (ICAI) system or an Intelligent Tutoring System (ITS) is to develop an adaptive instructional system by applying Artificial Intelligence principles and techniques. Traditionally ICAI systems have been separated into four major components: the domain knowledge base, a collection of instructional strategies and an algorithm for applying them, a student modeler, and an interface. Since a major goal of CIRCSIM-TUTOR is to carry on a natural language dialogue, we have divided the interface into three pieces, an input understander, a text generator, and a screen manager. As a result, CIRCSIM-TUTOR has seven submodules.

3.1 Intelligent Tutoring Systems

Computer Aided Instruction (CAI) systems were first developed by educational researchers and widely used during the 1950's and 1960's. Carbonell [1970] defined a second type of CAI, Intelligent Computer Aided Instruction, initiated by computer scientists (see Figure 4). It aims to teach the individual student more effectively and adaptively than traditional CAI systems, by applying AI principles and techniques. Typical ICAI systems consist of four main components. For each component of the system, various AI techniques have been applied to improve the performance of the system.

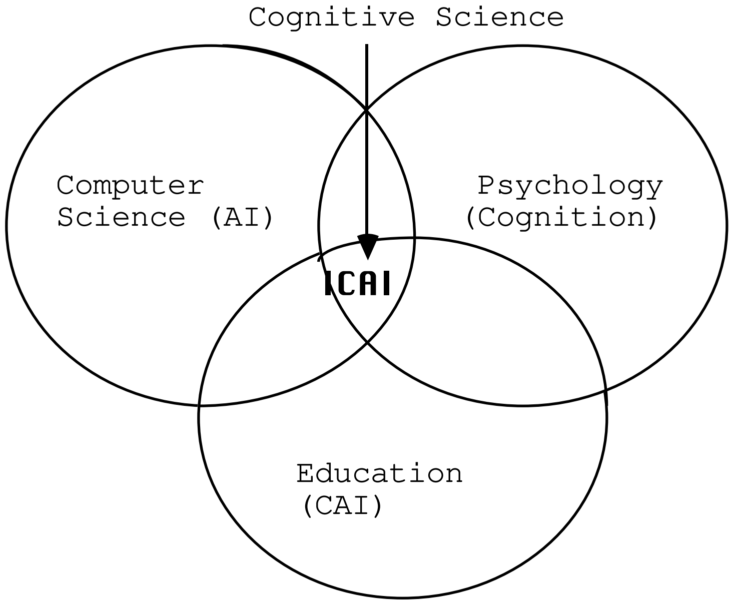

Figure 4. ICAI Domains (adapted from [Kearsley, 1987] p. 4)

3.1.1 Components of an ICAI. In the early form of CAI, all he components were combined in a single structure. This combined structure caused a number of problems when the system was modified. It was sometimes necessary to restructure the whole system. Thus, there was a need to divide the system into separate components to represent the way the tutor and the student act in a learning situation: the knowledge to be taught, the instructional module, the communication method, and a mechanism for modeling the student. A number of researchers [Carr and Goldstein, 1977; Sleeman and Brown, 1982; Barr and Feigenbaum, 1982] separated the system into four different modules: the domain expertise module, the student model module, the tutoring module, and the communication module.

As we designed CIRCSIM-TUTOR we adapted these four major components of an ITS and divided the domain expertise into a domain knowledge base and a problem solver. We also separated the communication module into three interface submodules, so that CIRCSIM-TUTOR consists of seven major modules: a domain knowledge base, a problem solver, a student modeler, an instructional planner, an input understander, a text generator, and a screen manager. Figure 5 shows the overall architecture of CIRCSIM-TUTOR. Details of each component will be explained in the following sections.

3.1.2 Applying AI Techniques in ICAI. Early efforts to apply AI techniques in ICAI systems focused on the representation of the subject matter, which was implicitly encoded in the program in early CAI systems. Various knowledge representation techniques, such as semantic networks, rules, and scripts, have been applied. But the most important progress is to create an explicit and separate domain knowledge base. This development allows easy modification of the domain knowledge without reformulating the whole system.

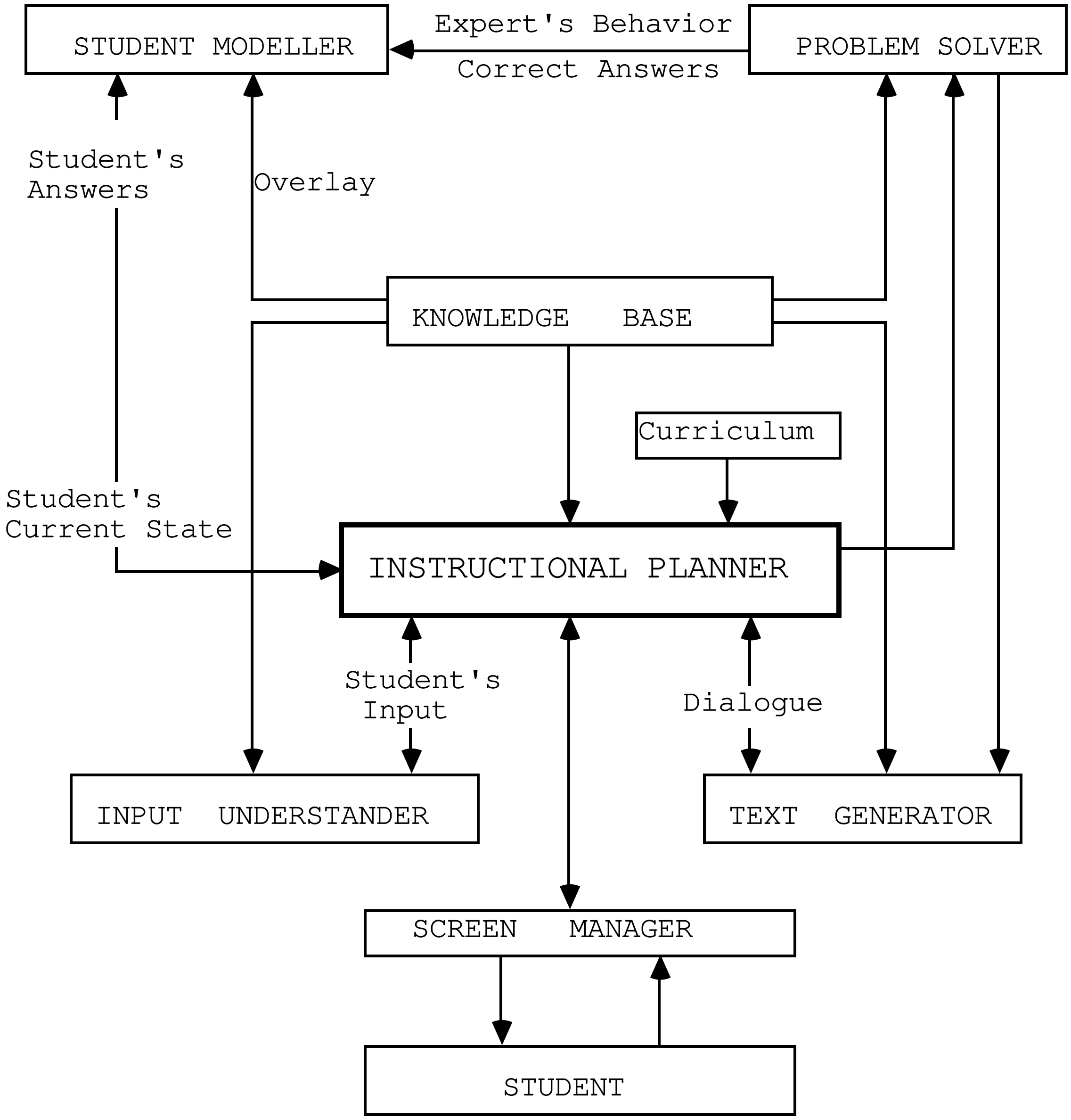

Figure 5. The Structure of our System

Another AI technique, natural language generation, plays a very important role in ICAI systems. Users can communicate with the system by asking a question and answering the question in a language they already know. SCHOLAR [Carbonell, 1970], SOPHIE [Brown et al., 1982], and WHY [Stevens et al., 1982] adapted natural language interaction. In addition, it can provide context-dependent tutoring [Woolf and McDonald, 1984].

From the mid-1970's, ICAI systems focused on the analysis of the student's learning status. Generally well-defined expertise does not guarantee an expert teacher. Without exact knowledge about what the student knows and does not know for a given problem, the system cannot provide adaptive instruction. Hence, AI techniques were used to evaluate the knowledge status of the student [Carr and Goldstein, 1977; Brown and Burton, 1978]. This model can be used effectively for deciding the next appropriate tutoring strategy by the tutoring module.

Finally, AI techniques are applied to represent the expert's pedagogical knowledge. In traditional CAI systems, the tutoring strategy is procedurally hard-coded in the program. It is structured as a branching program; if the student answers A then go to this section, if the student answers B then move to the next section. If the system needs to contain all the possible answers from the student, the system may become very complex. Hence, the system is expensive to build and hard to modify, because the system is greatly anticipated by the effort of a human author. Using AI techniques, such as instructional planning, the tutorial strategies can be represented explicitly and automatically.

3.2 Domain Expertise

3.2.1 Domain Knowledge Base. The builder of a domain knowledge base faces two very important issues; what knowledge should it contain and how should that knowledge be encoded [Wenger, 1987]. There are three different categories of knowledge encoding [Anderson, 1988]: the black box model, the glass box model, and the cognitive model. The cognitive model is the approach that CIRCSIM-TUTOR is attempting to implement. The domain knowledge is decomposed into meaningful, human-like components and a causal reasoning mechanism is applied to it, so that the system can teach the student to solve problems in a human-like manner. For a detailed discussion of this problem see Wielinga and Breuker [1990].

Domain knowledge can be divided into three different types of knowledge to be tutored: declarative knowledge, procedural knowledge, and knowledge of tutoring heuristics. Declarative knowledge includes domain concepts and causal relationships between them. Procedural knowledge involves the rules for using the concepts in solving problems. For example, in CIRCSIM-TUTOR, a rule that figures out the *actual determinant of SV* is *if the primary variable is RAP, then RAP is the actual determinant of SV.* Knowledge of tutoring heuristics must be extracted from the experience of domain experts; it involves ways of teaching the student about the particularly difficult points in the domain.

We have built a small domain knowledge base encoded as a network of frames (see Figure 6). Each frame represents domain concepts and how they relate to each other causally. There are three conceptual levels in the domain knowledge; level 0 consists of the definitions and static facts, level 1 consists of the cause-effect relationships between the parameters of the cardiovascular system, and level 2 contains a deeper knowledge of underlying physiology. The level 2 knowledge is used when the tutor needs to give a hint to the student. Currently, the level 2 knowledge is under refinement and development. Hence, in the present program the domain knowledge base is constructed as a set of components that is used for both problem solving and causal explanation. This is the most important and the basic knowledge that constitutes the domain expertise.

3.2.2 Problem Solver. According to Clancey [1989], the intelligence of an ITS comes from its ability to solve the problems. The problem solver solves the problems presented to the student or asked by the student. If the problem solver solves the problems but can not explain how it solves them, it may just as well retrieve stored answers. The ability to solve the problem, using the expert's problem solving behavior, can be used to identify the student's misconceptions, to give an explanation, and to provide a basis for tutoring strategies.

```
(frame SV

   (frame-type           variable
    var-type             physically-affected
    frame-name           SV
    class                instance
    instance-of          variable
    name                 Stroke Volume
    definition           volume of blood ejected each
                         heart beat
    part-of              heart
    anatomy              ventricle
    causal-relation-in   causal-RAP-SV causal-CC-SV
    causal-relation-out  causal-SV-CO))
```

Figure 6. A Frame from the Domain Knowledge Base

Problem solving in CIRCSIM-TUTOR is carried out by two problem solvers: the main problem solver and the subproblem solver. The main problem solver solves the problem, generates correct answers, and produces the same problem solving path as an expert in the domain. This solution path can be used to monitor the student's problem solving behavior while the student is making entries in the predictions table. The subproblem solver solves current problems generated by the planner, such as *determinant of X, relationship between X and Y*, and also problems coming from the student questions. The other modules of the system may consult these problem solvers to get any information they need. For instance, the student modeler needs to consult the problem solver to evaluate the student's answer.

3.3 Input Understander

The input understander is responsible for understanding the student's natural language input. It handles not only well-formed but also ill-formed student inputs [Lee et al., 1990; Lee, 1990]. The student input may be either an answer to the tutor's question, or a question from the student. If the student's answer is *The actual determinant of SV is RAP*, then the planner will pass the sentence to the understander along with the current lesson topic in logical form, *(actual-determinant SV)*. Then the input understander parses the sentence, checks its coherence with the current topic, and returns the logic form, *(answer (actual-determinant SV (RAP))*. Then the planner extracts the student answer, *RAP*, and passes it to the student modeler to diagnose the student answer.

The input understander must also understand student initiatives; whether the student is asking for an explanation, or referring to the previous remarks of the tutor, or wants to stop the session. For example, if the student initiative is *I don't understand about SV*, then the input understander returns the logical form *(question (explain SV))*. Then the planner suspends the current plan and carries out the student's request. This process needs to be studied in detail in order to understand what the student really wants. We are still investigating it by analyzing transcripts, and it may require a richer knowledge structure, like that used in the UNIX Consultant [Wilensky et al., 1988].

3.4 Student Modeler

The student modeler is responsible for representing the student's understanding of the subject by building a student model [VanLehn, 1988]. The student model is a data structure that represents the student's current state of knowledge; what the student knows, what the student does not know, and what misconceptions he or she may have. Based on this information, the tutor can give individualized instruction to the student. There are two major approaches for student modeling. One approach, the overlay model [Carr and Goldstein, 1977], is designed to represent the student's knowledge state as a subset of an expert's knowledge state. Another approach, the buggy model [Brown and Burton 1978], represents the student's misconceptions not as subsets of the expert's knowledge, but as variants of the expert's knowledge. In CIRCSIM-TUTOR, the student modeler integrates overlay and buggy strategies into one [Shim et al., 1991; Shim, 1991].

In CIRCSIM-TUTOR, the student modeler begins analyzing the student's entries in the Prediction Table. Based on this analysis, the planner generates a lesson plan and the natural language tutoring session begins to correct the student's misconceptions. During the tutoring session, the planner sends the student's answer to the modeler and the modeler analyzes it and returns the result. Based on this information, the planner can decide what to do next. Currently only the overlay information is used for choosing the next tutoring strategy.

3.5 Instructional Planner

The instructional planner is responsible for determining what to do next at each point during a tutoring session. The planner also performs the system controller function. It interacts with the input understander, the text generator, the student modeler, and the screen manager, in order to carry out tutorial activities. Although the design of the planner may vary depending on the purpose of the ITS, several researchers have recently proposed combining opportunistic control with a plan-based approach [Derry et al., 1988; Murray, 1990; Macmillan et al. 1987]. For instance, Murray [1990] suggests that the way to provide opportunistic control with global lesson plans is to implement a dynamic instructional planner. For CIRCSIM-TUTOR, the planner needs to generate the global lesson plan and take care of the discourse control as well [Woo et al., 1991a]. Since this is the main topic of my research, it will be discussed in detail in the later chapters.

3.6 Text Generator

The text generator is responsible for turning the tutor's output into a natural language sentence. It receives necessary information as a logical form from the planner and generates a natural language sentence or sequence of sentences [Zhang, 1990]. This information includes the current topic and text styles: question, hint, answer, etc. For example, the text generator is given a logic form from the planner, *(question (affected-by SV ?))*, then it produces the English sentence, "What are the determinants of SV?" The text generator can handle this kind of simple question, explanation, or acknowledgement. But giving a hint may require more deep knowledge information, either from the planner, student modeler, knowledge base, or from the input understander. The current version of the text generator only receives the necessary information from the planner, not from all the other modules, so that its behavior is somewhat passive.

3.7 Screen Manager

The screen manager takes care of the interaction between the student and the system. The interaction is closely controlled by the planner; the planner tells the screen manager what to display, and the screen manager sends back the student's input to the planner. Thus, every interaction passes through the planner. The screen manager may display its own messages, such as help messages or warning messages.

First, the screen manager displays system messages through the introductory windows. Then it displays the list of procedures that the student can select. When the student selects the problem, it displays the prediction table with instructions about how to use the mouse and how to make entries into the table. Then it receives qualitative answers, (+, −, 0), from the prediction table one by one from the clicking of the mouse and passes them to the planner. When the student clicks outside the boundary, for example, if the student clicks on the wrong column during Predictions Table entry, the screen manager displays a warning message with a beep. It also handles two other windows, the student window and the tutor window. From the student window, it receives the student's natural language input in English sentences. In the tutor window, it displays natural language sentences created and passed to it from the text generator.

3.8 Summary

This chapter began with a description of the basic ITS components; a domain expertise module, a student modeler, a tutoring module, and a communication module. It continued with a discussion of ways to apply Artificial Intelligence techniques to each component of the system to improve its performance. Then the chapter introduced the seven submodules of CIRCSIM-TUTOR; explained the functions, data structures, interactions, and the current development situation for each submodule. During future development of the system, the basic structure of the system may remain the same, but the functions of the each module can be modified or extended with different approaches. Figure 5 shows the overall structure of the system.

CHAPTER IV

SURVEY OF PLANNING IN TUTORING SYSTEMS

Planning is an approach to problem-solving that creates a sequence of actions (i.e., a plan) to achieve a goal. If the input to the planning system is a problem, specified with its initial state, goal state, and a set of actions, then the output to the system is a *plan* that satisfies the goal (Figure 7). Early research on planning focused on the physical actions of robots [Fikes and Nilsson, 1971; Sussman, 1975; Tate, 1975], in which planning and execution are separated. Recent planning systems try to extend the earlier classical planning systems, by integrating planning and execution, so that they can monitor the execution of a plan and revise the plan when it is necessary.

The application of planning techniques in the domain of instruction, instructional planning, becomes a major issue in an ITS [Woolf, 1984; Russell, 1988; Macmillan et al., 1988]. It functions as a control mechanism that decides what to do next by creating a sequence of instructions; determining what topics should be introduced, reviewed, explained, etc. This control mechanism is the central component of the ITS, an instructional planner, and alternative control strategies are the basis of different tutorial approaches. By deciding what to do next, the planner is controlling the system's interaction with the student.

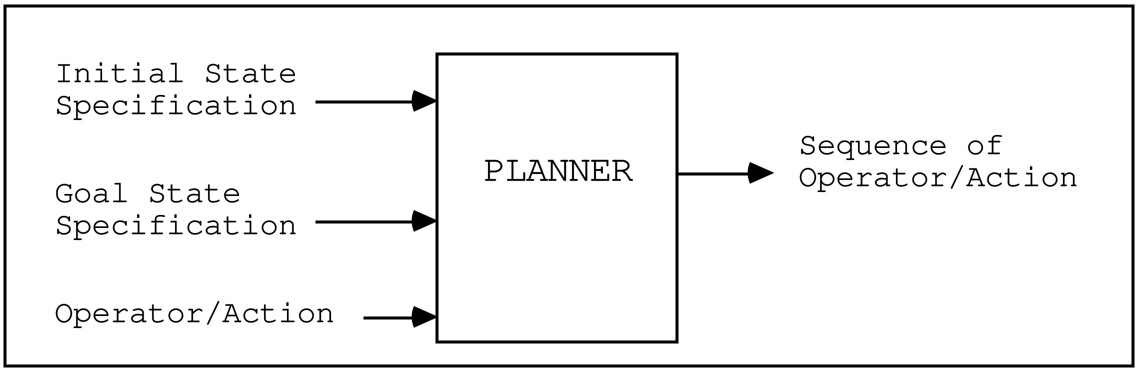

Figure 7. Planning System

4.1 Approaches to Planning

4.1.1 Classical Planning. Many real-world problems may be explored by the planning system: robot control, automatic programming, experiment design in molecular genetics, aircraft carrier mission planning, and natural language generation. The early research on planning focused on the physical actions of robots. In this planning system, an initial plan was generated, criticized, and then patched before any of the actions were carried out (Figure 8).

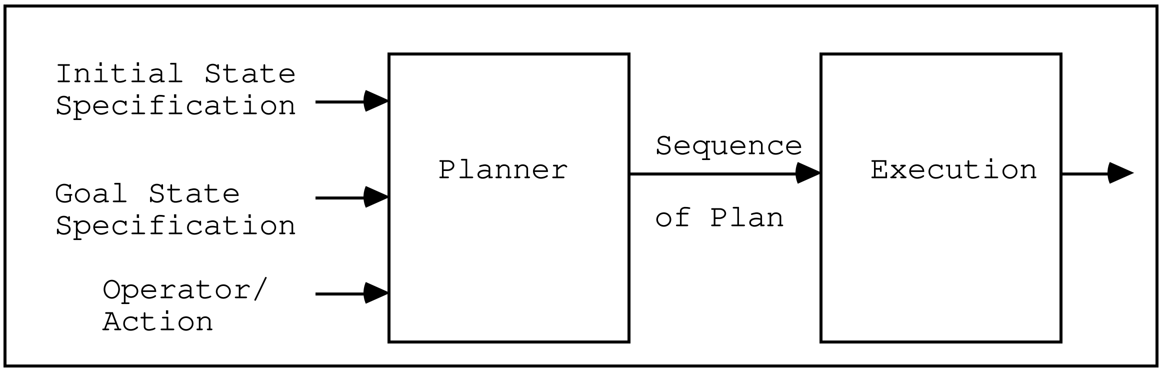

Figure 8. Organization of a Classical Planning System

This arbitrary ordering of steps in the plan may cause some problems during execution. In the example shown in Figure 9, the initial goal is divided into two actions (i.e., subgoals) arbitrarily, before execution. Eventually the initial ordering of actions will fail in this planning system, because of the protection violation rule for the first action (i.e., subgoal conflicts). Thus, the planner has to backtrack, reorder the subgoals, and execute them again to achieve a goal. The early planning systems, HACKER [Sussman, 1975], and INTERPLAN [Tate, 1975], applied a heuristic called *the linear assumption*, which states that one ordering of actions is as good as any other and fixes the interactions when they arise. However, this kind of *create and debug* strategy causes backtracking, which can be very expensive. Successful ordering can involve a combinatorial explosion if there is a huge number of possible orderings.

The early planning systems are classified as nonhierarchical planning systems or linear planning systems, since they assume linearity in solving problems. Hierarchical planning or nonlinear (partial-order) planning systems arose out of dissatisfaction with these linear systems. Hierarchical systems do not commit to an arbitrary order, instead they postpone the commitment until the order can be executed. The difference between hierarchical and nonhierarchical planning will be explained in detail in the next section.

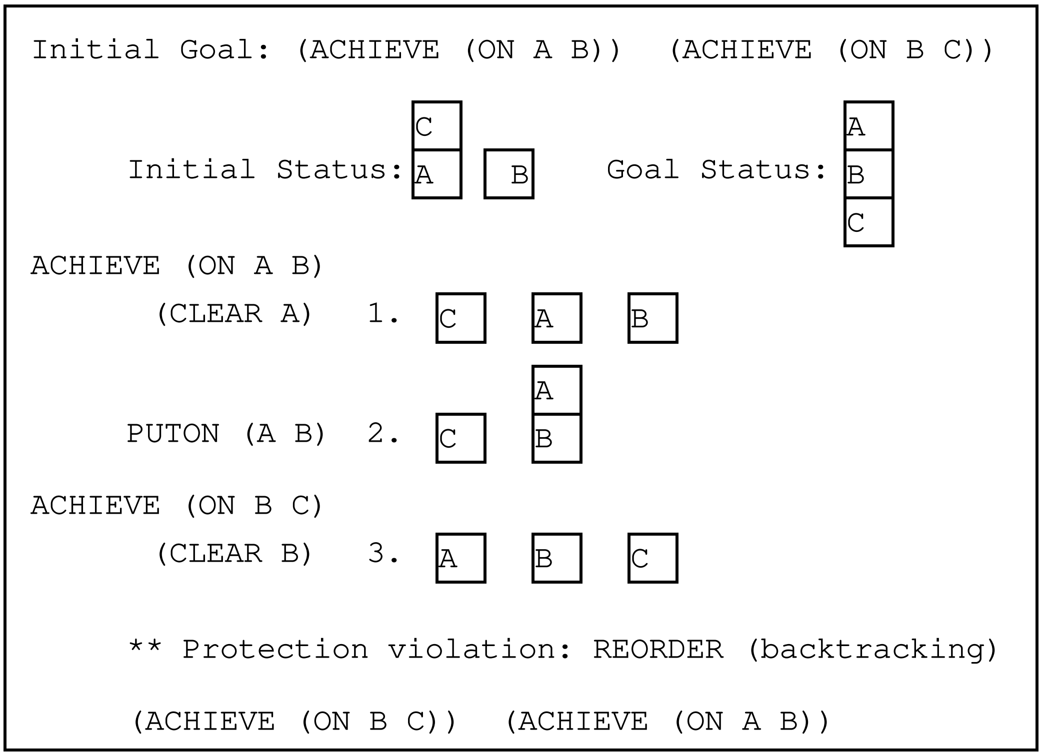

Figure 9. INTERPLAN [Tate, 1975] Backtracking Problem

4.1.2 Hierarchical vs. Nonhierarchical Planning.

Hierarchical planning came out of dissatisfaction with nonhierarchical planning such as that done by STRIPS [Fikes and Nilsson, 1971] and HACKER [Sussman, 1975]. Hierarchical planning is concerned with the relation between tasks and subtasks [Charniak and McDermott, 1986]. The planner can decide to perform a task A, and later decide to carry out A by performing its subtasks. It generates a hierarchical representation of a plan, in which the highest level is a simplification or abstraction of the plan and the lowest is a detailed plan, sufficient to solve the problem. Nonhierarchical planning has only one representation of a plan. A major disadvantage of nonhierarchical planning is that it does not distinguish between problem-solving actions that are critical to the success of a plan and those that are simply details. Thus, it needs to backtrack when an action fails. The advantage of a hierarchical planning system is that the plan is first developed at a higher level and the details are developed later; this prevents development of unnecessary plans in advance.

A well-known example is the comparison of STRIPS and ABSTRIPS [Sacerdoti, 1974] in the same *coffee* domain by Cohen and Feigenbaum [1982a]; the goal is to drink coffee with two subgoal actions of buying coffee and making coffee. In this example, ABSTRIPS, an extension of STRIPS, solves the problem with much less searching and backtracking than STRIPS. STRIPS generated many steps that were not necessary to solve the problem; on the other hand, ABSTRIPS uses a strategy that separates the subgoals into levels of priority, with the abstract and general subgoals being developed first and detailed levels developed later. ABSTRIPS finds a solution at the most abstract level and does less backtracking than STRIPS. This example shows the efficiency of hierarchical planning over nonhierarchical planning.

NOAH [Sacerdoti, 1977] is a hierarchical planning system, which uses a *least-commitment* strategy that involves partial-ordering of problem-solving operators by considering their preconditions. MOLGEN [Stefik, 1981] and NONLIN [Tate, 1977] are based on NOAH but use other methods for deciding what to postpone.

4.1.3 Recent Planning Systems. Most of the classical planning systems assume that the planner possesses complete information about the problem, and the generated plans will be carried out successfully (i.e., plan and execution are separated). But there is no guarantee that the execution will always be successful. Hence, this approach needs to be changed when separate execution cannot be guaranteed to succeed. Recently, a number of researchers have been working on this problem [Hendler et al., 1990].

More recent planning systems have extended classical planning approaches by integrating planning and execution (Figure 10): Opportunistic Planning [Hayes-Roth, 1985], Incremental Planning [Durfee and Lesser, 1986], Replanning [Wilkins, 1988], and Case-Based Planning [Hammond, 1989]. Most of these ideas were originally discussed on a theoretical level, but the concepts can be generalized and applied to instructional planning systems, since instructional planning involves a complex interleaving of a plan and its execution. An instructional planner has to be dynamic because during instruction, the student's cognitive status changes dynamically. It must be able to replan because the plan may need to be revised during instruction.

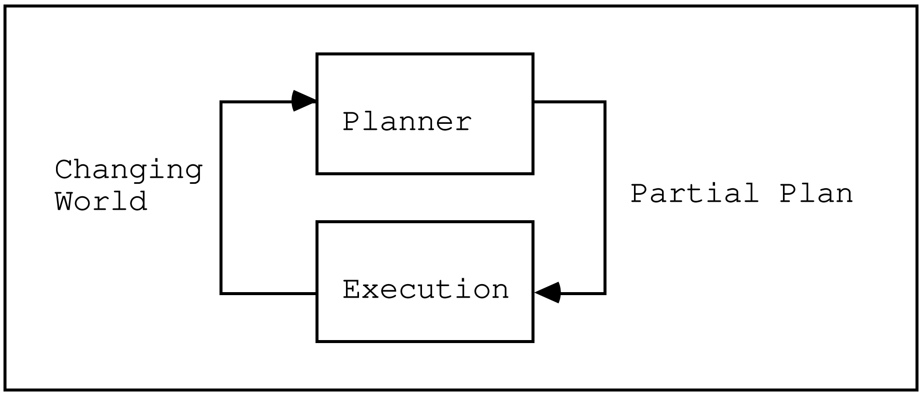

Figure 10. Organization of a Dynamic Planning System

4.1.4 Replanning. In real-world domains, like control of robot actions, things do not always proceed as planned. Thus, it is necessary to monitor the results of current plan execution to the expected results at each step. Plan revision is necessary when new information invalidates the old plan [Wilkins, 1988; Swartout, 1988]. However, finding an optimal plan, in order to revise the old plan, may take time and effort. Thus, there is always a trade-off between speed and optimality. If time is crucial, it may be better to replan than to revise the old plan.

Plan monitoring can be done by inserting monitoring steps in the plan, which must have a model of what the plan monitor can detect. This plan monitor generates the necessary monitoring information, and the planner checks this information after every step. This plan monitoring step acts like a student model in instructional planning.

4.2 Approaches to Instructional Planning

Any ITS must have a mechanism for determining what to do next in the tutoring session. This instructional planning component decides what subject material to focus on next, how to deliver the selected topic, and when to interrupt the student's problem-solving. These pedagogical decisions were hard-coded in early CAI systems, which made the systems hard to modify and hard to adapt to other domains. In this regard, the ideal system needs to represent this pedagogical knowledge explicitly as a form of rules or similar structures, so that it is expressed declaratively and interpreted into actual decisions automatically, whenever it is referenced.

Conceptually, these pedagogical decisions can be divided into two different levels [Wenger, 1987]: a global level and a local level. Global level decisions affect the sequences of subject matter being taught. Based on the information from the student model, the system is capable of providing different instructional content to different students (adaptive instruction). Knowledge about when to interrupt the student's problem solving, what to say and how to say it, belongs to the local level decision making process.

Making such decisions is very complex, since the order of instruction and the method of communicating with the student may produce different learning experiences [Cohen and Feigenbaum, 1982b; Wenger, 1987]. Successful learning may depend on many factors, but the first priority is that it should not destroy the interest of the student. Thus, it is important to determine the degrees of control over the interaction between the system and the student. For example, in a mixed-initiative strategy, the system and the student share the control and in a Coaching system, the student is in full control of the activity.

4.2.1 Teaching Strategies. Most existing ITSs base their teaching strategies on a *diagnostic method*, in which the tutor tries to estimate the student's knowledge by asking questions and evaluating his responses. From the tutor's feedback (by explanation or answer), the student is expected to learn about his mistakes. BUGGY [Burton and Brown, 1982] used this method.

Another method is the *Socratic method*. In the Socratic method, as in SCHOLAR [Carbonell, 1970] and WHY [Stevens et al., 1977], the tutor provides an environment and encourages the students to debug his own misconceptions (that is, the tutor does not teach a subject directly, but asks questions in a way that will encourage the student to reason about what he knows and what he does not know, to find the contradictions, and thereby to modify his misconceptions).

Another approach is the *Coaching method*, as in SOPHIE [Brown et al., 1982], WUMPUS [Carr and Goldstein, 1977], and WEST [Burton and Brown, 1982]. The Coach watches the student's behavior and does not interfere constantly. Instead, when the student asks for help, the Coach interrupts the process and gives an important lesson. The goal of the Coaching method is to encourage students in skill acquisition and problem-solving abilities like a computer game. The basic philosophy of the Coaching method is *learning by doing*.

The early ITSs explored the representation of these tutoring strategies explicitly, which led to the development of instructional planning. GUIDON [Clancey, 1982] uses Production Rules to represent its tutorial strategies. These early ITSs, WUSOR [Glodstein, 1977], WEST [Burton and Brown, 1982], and GUIDON, are the first to explicitly represent the discourse knowledge as rules, including rules for introducing a topic, asking a question, etc. However, the disadvantage is that if the domain is complex, a large number of rules are required. They also lack a global context.

More recent ITSs turned their research toward an explicit control mechanism, such as determining what to do next. Among a number of researchers, Woolf [1984] organized the discourse procedures of GUIDON into a discourse management network in her system (MENO-TUTOR). In this system the control mechanism and hierarchical representation of tutorial discourse strategies are represented explicitly in the network.

4.2.2 Plan-Based vs. Opportunistic Control. The ITS has goals for the student to achieve. Goals can be achieved by planning instructional activities (i.e., plan-based control), or by recognizing diagnostic opportunities from the interaction with the student (i.e., opportunistic control).

In recent systems the opportunistic control approach has been dominant over the plan-based approach. Opportunistic control uses diagnostic information to recognize opportunities for intervention, to introduce new material or remediate a misconception during the tutoring session. It is good for coaching in learning environments or guiding problem-solving activities (e.g., WEST or GUIDON). The disadvantage is that it provides very little control over the organization of tutoring sessions. On the other hand, the plan-based approach uses diagnostic information to monitor the progress and appropriateness of the current instructional plan. It provides a well-organized hierarchical structure, but the student's behavior tends to be less important. For example, student requests are ignored or not possible.

Wenger [1987, p. 400] argues that goals are best achieved by an appropriate combination of both control styles. Also studies of human tutors show a complex interleaving of these two styles [Leinhardt and Greeno, 1986]. Ideally, the design of the planner combines the plan-based and the opportunistic control approaches [Wenger, 1987; Derry et al., 1988; Murray, 1990].

4.3 Review of ITSs

Instructional planning began from the early 1960's as human-authored planning in a CAI system. As CAI evolved toward ITS, instructional planning has been approached in a number of different ways. The traditional CAI systems do not generate plans at all. Instead they follow a single conditional plan designed by the author. Although the decisions were made dynamically, plans were prespecified and fixed. These systems are well-organized and motivating, but they are inflexible and expensive to build, and the design is heavily dependent on the skill of the human author.

More recent approaches to instructional planning have explored ways to overcome the limitations, such as the high cost, the inability to generate plans, the lack of global context for planning of the earlier planners. For example, MENO-TUTOR is a sophisticated discourse planning system with explicit control mechanism and discourse strategies. COACH [Winkels et al., 1988] is an intelligent help system with the capability of handling local user needs opportunistically. TAPS, a plan-based opportunistic planning system [Derry et al., 1988], is the first system to raise the issue of the curriculum planning in an ITS and integrate curriculum planning with the discourse planning paradigm. A content planning system [Brecht et al., 1989], which is based on the SCENT system [McCalla et al., 1988], emphasizes the importance of content planning over discourse planning. SIIP [Macmillan and Sleeman, 1987] is a generic instructional planning architecture to support a dynamic instructional planning capability. A BB1-based dynamic instructional planning system [Murray, 1990] is a first step towards a dynamic instructional planner that can generate, monitor, and revise plans during the instructional sessions. SIIP and a BB1-based Dynamic Instructional Planner used a blackboard architecture for building their system. The system requires two blackboards: a domain blackboard and a control blackboard. A domain blackboard contains the lesson plan, and a control blackboard provides a control mechanism that decides how to construct and modify the lesson plan. It is capable of global context planning; it can plan, replan, and modify the plan.

In the next section, I will describe three well-known ITSs with different planning paradigms that I considered in building my planner. The contributions and the limitations are discussed.

4.3.1 MENO-TUTOR. MENO-TUTOR [Woolf, 1984] uses a discourse management network (DMN) to control a Socratic question and answer dialogue with a student by teaching new information or correcting misconceptions. Discourse planning in the DMN uses two mechanisms: a planning module and a language generator. The planning module makes decisions about what discourse actions to make and what information to convey or query. The language generator produces natural language output using templates. My concern is the planning module of the system.

The planning module contains a multi-level planner, tutoring states, and a knowledge base. The multi-level planner consists of three levels: the pedagogical, strategic, and tactical levels. The pedagogical level establishes an expository style of tutoring, for instance, *introduce a new topic* or *tutor a misconception*. It is further refined into a strategic level that determines the style of discourse, such as, *question the student* or *describe the concept*. The node at the strategic level is refined down to the tactical level to implement the strategy. The tactical states determine the form and content of the utterance. The control mechanism of

MENO-TUTOR consists of two phases: a default transition and meta-rule transitions. The default transitions are preplanned and are overridden by meta-rule transitions.

A. Contributions. First, MENO-TUTOR is an attempt at a generic tutor; it has been applied to reasoning about rainfall and about looping constructs in the PASCAL language. Second, it attempts to capture the discourse strategies observed in human tutors. Finally, this system supports Context-Dependent tutoring; output is different in different contexts.

B. Limitations. MENO-TUTOR lacks a lesson planning ability, so that it cannot generate customized, and globally coherent instruction. It is only concerned with planning the delivery of an already chosen topic. And there is no explicit mechanism to select the topic.

4.3.2 IDE-INTERPRETER. IDE-INTERPRETER [Russell, 1988] is a planner-based adaptive tutoring system that allows explicit representation of strategy, and shows the consequences of the strategy by synthesizing and delivering instruction accordingly. IDE-INTERPRETER delivers instruction for IDE [Russell et al., 1988]. The planner interprets a set of rules to expand instructional goals into subgoals. The planning approach is top-down plan expansion, in such a way that explicit plan representation is incrementally refined by applying rules. During each cycle of execution, it selects and executes an instructional unit (IU) as a primitive action. Student interaction with the IU is recorded and analyzed to update the student model. Then the planner uses the student model to modify the plan, constantly updating its plans to achieve the goals.

A. Tutoring Rules. There are three sets of rules, strategic, pedagogic, and tactical rules. Each of the rules represents tutoring knowledge. A strategic rule represents shifting instructional strategy, merging similar goals into one, controlling when and how to remediate. Pedagogic rules represent domain-specific tutorial information, for example, to teach definitions, first define the process, then the components. Tactical rules represent methods of presenting particular material (e.g., choose an IU that hasn't been used before).

B. Contributions. First, an explicit plan representation is refined by the rules, which enables customized instruction. The plan is represented as a tree that is refined top-down. Second, the use of rules, explicit plan representation, and an agenda control provides a dynamic instructional planning environment. Finally, it supports incremental planning since it does not select an IU before it begins execution.

C. Limitations. IDE-INTERPRETER emphasizes lesson planning over discourse planning since discourse actions are procedurally encoded in the IU and not reasoned by the planner, with the result that it cannot handle student requests between goals; it cannot make adjustments even if the student's performance is too low.

4.3.3 UNIX CONSULTANT (UC). UNIX CONSULTANT [Wilensky et al., 1988] is an intelligent, natural language interface that helps naive users to learn about the UNIX system. It consists of the following components: a language analyzer (ALANA), a goal analyzer (PAGAN), a domain planner (KIP), an expression mechanism (UCExpress), a language production mechanism (UCgen), a user's knowledge model (KNOME), a knowledge acquisition mechanism (UCTeacher), and a knowledge representation system (KODIAK). My interest is in the planning component (KIP) of the system.

KIP is a domain planner. It is given a task from a controller (UCEgo) and determines how to accomplish the user's goal based on knowledge of the user and knowledge of UNIX. It returns a plan, represented in KODIAK. Another planner, UCEgo, reacts to the user's request by forming goals and acting on them. It also acts as a controller for the whole system. The following are examples generated by UC.

A. Simple Examples. The first student is diagnosed as a novice user, so UC provides an answer with an example. The second student is a sophisticated user, so the UC answers without an example.

Student 1: How can I delete a file ?
UC: Use rm. For example, to delete the file named foo, type 'rm foo'
Student 2: How can I find out the inode of a file?
UC: Use ls -i.

B. More Complex Examples. Underlying goals are not addressed by a short answer. UC attempts to answer them using an extended answer.

Student: Is rn used to rename files?
UC: No, rn is used to read news. To rename a file to another file, use mv. For example, to rename file named foo to the file named fool, type 'mv foo fool'.

C. Contributions. UC addresses many AI research issues in planning, reasoning, natural language processing, and knowledge representation. It is an attempt to simulate a human consultant. And it has a very rich knowledge representation system (KODIAK).

D. Limitations. For efficiency, UC's control structure is overly simplistic; the planner does not do much work. The planner simply ends up retrieving the stored plan, making an instance of it, and passing it on. Since UC is a help facility, it is not capable of handling every task put to it to serve a useful function; it may discourage the student when UC cannot process a request correctly.

4.4 Summary

This chapter began with brief introduction to AI planning systems, including both classical and recent approaches. It continued with a discussion of the application of planning techniques in the domain of instruction, instructional planning, which is a control mechanism that decides what to do next by creating a sequence of instructions. Then the chapter discussed some design issues for building the instructional planner, and concluded with a review of well-known ITSs with different planning paradigms, contributions and limitations of the systems.

CHAPTER V

PLANNING INSTRUCTION

The instructional planner is the central component of the ITS; it is responsible for selecting or generating instructional goals, deciding how to teach the selected goals, monitoring and critiquing the student's behavior, and determining what to do next at each point during a tutoring session. That is, the planner makes two different types of important decisions during the tutoring session, decisions about the content of the lesson and decisions about its presentation strategy. Although the early ITSs largely focused on the delivery strategy of the planner, some recent planning research shows the integration of both aspects in building the planner [Macmillan et al., 1988; Derry et al., 1988; Murray, 1990].

The planning component of CIRCSIM-TUTOR must carry out both functions, since it needs to provide a global lesson plan, and it needs to carry on a natural language exchange with the student. For this reason, I developed the planner in two parts: a lesson planner and a discourse planner. This chapter discusses general design issues of the planner with the goal of providing most effective instruction to the student, a sample dialogue extracted from the transcript of an actual human tutor-student interaction and a scenario implementing that dialogue, and a description of the overall organization of the planner.

5.1 Design Issues

5.1.1 Capabilities of the Planner. Most machine planning systems, like STRIPS, HACKER, and NOAH, deal with the observable physical world, whereas instructional planning systems deal with unpredictable dynamic changes in the student's knowledge. The student's current learning status can never be observed directly. It can be only guessed; the results of this guesswork are stored in the form of the student model [VanLehn, 1988]. Thus, the instructional planner must possess unique capabilities for handling unpredictable situations as an expert human tutor does.

The planner must plan at different levels of the hierarchy; a *hierarchical planning* technique can reduce the complexity of the planning process. The plan is first developed at a higher level and the details are developed later; this technique prevents development of unnecessary plans in advance. The planner must plan at a *global* level; when the planner generates the next instruction, it must consider the past plan and the student's responses to provide continuity of instruction. The planner must *replan* when the current plan fails or a request is made by the student. The planner should be able to monitor the plan to identify the need for replanning. The planner of CIRCSIM-TUTOR provides all these capabilities.

5.1.2 Levels of Planning. Research by Leinhardt and Greeno [1986, cited in Derry et al., 1988] has shown that experienced teachers employ levels of planning in accomplishing their goals; planning instructional goals occurs at the most global level, planning actions and decision-making occur at a less global level. Inspired by this research, Derry et al. [1988] designed their TAPS system with three levels of instructional activity: curriculum planning (the agenda), lesson planning (instructional actions), and on-line tutorial intervention. Murray [1988] also distinguished three levels of instructional planning; curriculum planning (planning a sequence of lessons), lesson planning (determining the subject matter in a single lesson), and discourse planning (planning communicative actions between the tutor and the student). He argues that at least two levels of planning, lesson planning and discourse planning, must exist in an ITS to deliver more effective and flexible instruction, although these three levels cannot be cleanly separated and often the curriculum planning and the lesson planning interwine, as well as the lesson planning and the discourse planning.

CIRCSIM-TUTOR is capable of both lesson planning and discourse planning. It can be set up so that the student can select a problem from a list of seven experimental procedures or it can do complex curriculum planning. The number and types of procedures will be extended further in future versions of the system.

5.1.3 Mixed-initiative Strategy. Carbonell [1970] presented a mixed-initiative paradigm in SCHOLAR, where both the tutor and the student can initiate conversation by asking questions. This strategy best fits conceptual or procedural learning tasks [Kearsley, 1987]. SOPHIE [Brown et al., 1982] is an another system that used this strategy. The following dialogue shows a part of an interaction in SCHOLAR that illustrates a mixed-initiative form.

Tutor> What is the language of Argentina?
Student> Probably Spanish.
Tutor> Very good. Approximately what is the area of Argentina?
Student> Tell me something about Peru.

From the last line of the dialogue, we can see that the student is taking the initiative, rather than answering the question to the tutor. Human tutors allow the student to ask a question during the tutoring session, but responding to student initiatives increases system complexity a great deal, because it is hard to satisfy unexpected student questions and decide what to do in all circumstances. CIRCSIM-TUTOR is aiming to adapt this mixed-initiative strategy. At the present, we are analyzing student initiatives in transcripts of human tutoring sessions and implementing responses to a couple of simple and frequent types of student initiative.

5.2 Scenario

5.2.1 A Sample Tutoring Session.

We have recorded a number of tutoring sessions with our experts, Joel Michael and Allen Rovick, who are Professors of Physiology at Rush Medical College, and some of their first year medical students. After careful studies of the recorded transcripts, we extracted some possible tutorial strategies and tactics that provided us with the framework for building the instructional planner and the overall system. It is assumed that students have already learned much of the domain knowledge, hence the system will mainly assist the students to correct their misconceptions and to solve problems. Our current system can handle dialogues like the following.

Example 1:

Tutor> What are the determinants of SV?

Student> SV is determined by RAP and CO.

Tutor> RAP is correct, but CO is not a determinant of SV. Remember. SV is the amount of blood pumped per beat. What is the other determinant of SV?

One important point about the above tutor-student interaction is the content of the questions posed by the tutor. For example, on the first line of the excerpt, the tutor is asking the student about the *determinants of stroke volume*. Asking a question about *determinants* is the first part of the plan that the tutor is using to teach the student about the *causal relationships* between two variables, RAP and SV. Thus the content of the question has to be generated by the lesson planner before the tutoring begins. Another important aspect is how to present the selected topic. From the above short excerpt, we can see four different kinds of delivery modes: a direct question (line 1), positive and negative acknowledgements (line 3), hints (line 4), and follow up questions after hints (line 5). Thus, the planner (discourse planner) needs to plan how to present the selected content to the student effectively.

Example 2:

Tutor> By what mechanism is TPR controlled?

Student> Nervous System.

Tutor> Correct, TPR is controlled by the nervous system. Then what is the correct prediction of TPR?

Student> No change.

Example 2 is an another tutoring situation that focusses on one of the neurally controlled variables, TPR. The tutor first asks the student about its *control mechanism* in line 1. This control mechanism is the first strategy to teach the student about the neurally controlled variables. Since the student answered correctly, the tutor gives a positive acknowledgement and then uses its second strategy, asking for a *prediction*, in line 4. We have extracted this kind of tutoring strategy from the transcripts and designed explicit lesson planning rules.

From the above examples of tutor-student interaction, we can distinguish between the subject matter and its presentation formats. Ohlsson [1986, p. 217] argues that an effective ITS should be able to generate different presentations of each piece of subject matter in order to provide adaptive instruction to the student. The content of the questions posed by the tutor and its delivery modes lead to the development of two different kinds of instructional planning, lesson planning and discourse planning, because the subject to be taught has to be generated adaptively, and also its presentation form can vary according to the situation.

<u>5.2.2</u> <u>Implementation</u> <u>of</u> <u>the</u> <u>Scenario</u>. I am building an instructional planner based on observations of expert human tutors like those shown above. For example, assume that the current lesson goal is to tutor the causal relationships between two parameters, RAP and SV. This goal gets refined into a set of hierarchical subgoals by using strategic and

tactical rules. The subgoals generated at the tactical level, such as *determinants, actual determinant, relation, and value*, are kept in a stack, which is used by the discourse planner to pick the next topic.

The following scenario describes what each component of the system does, what kind of information it needs, and what is the result after each step. The steps are numbered to show the execution sequence. This tutorial interaction begins after the lesson planning is done. So that the discourse planner begins with the first topic in the stack, the determinants, and when that topic is completed, continues with the next topic, the actual determinant, and so on.

1. Planner: Picks the current topic from the stack, selects the discourse tactic, and passes it to the text generator as an internal logical form.

   current topic: (determinant SV),
   discourse tactic: question.
   call Text Generator: (question (determinant SV))

2. Text Generator: Generates a sentence,
   "What are the determinants of SV?"

3. Screen Manager: Displays the sentence in the window.

4. STUDENT: "SV is determined by RAP and CO."

5. Planner: Passes the student's input with the current lesson topic to the input understander.

```
(question (determinant SV)
            (SV is determined by RAP and CO))
```

6. Input Understander: Parses the student's answer, checks its coherence with the dialog history, and returns the answer to the planner in logic form.

```
call planner: (answer ((determinant SV)(RAP CO)))
```

7. Planner: Passes the current topic and student answer to the the student modeler in logic form.

```
current topic: (determinant SV),
student answer:  (RAP, CO),
call Student Modeler: ((determinant SV) (RAP, CO))
```

8. Student Modeler: Calls the problem solver, gets the correct answer: (RAP, CC), compares the correct answer with the student answer, and updates the student model.

In step 1, the discourse planner picks the topic, *determinant SV* from the subgoal stack, selects the discourse tactic, *question*, binds these two together into a logical form, *(question (determinant SV))*, which is passed to the text generator to generate a natural language sentence. After receiving the logical form from the planner, the text generator generates a sentence like the one in step 2. In step 3, the screen manager displays the sentence on the student window, and the student responds with the answer in step 4. So the current dialogue is:

Tutor> What are the determinants of SV?
Student> RAP and CO.

In step 5, the planner passes the student's input along with the current topic. The input understander has to recognize the student's answer; parse the answer, check its coherence with the dialogue history, and return the answer to the planner in its logical form. Then the planner sends the current topic with the student's answer to the student modeler in step 7. Finally, the student modeler analyzes the student's answer, and records the result in the student model. The next step will start with the planner checking the student model, and then deciding what to do. Since one of the student's answers is wrong, the planner consults its tutoring rules and decides to give some acknowledgement first:

Tutor> RAP is correct, but CO is not a determinant of SV.

At this point the tutor has two choices, either give a hint or just give an answer and continue with the next topic. Since this is the first trial, the tutor decides to "give a hint" and then ask a question to complete the previous answer. So a possible response would be:

Tutor> Remember. SV is the amount of blood pumped per beat. What is the other determinant of SV?

A different tutoring rule will be applied if the student again makes an error after receiving a hint; the student will be given a direct answer for the second question. Our current tutoring rules vary according to the topic and the student's responses (i.e., the tutor gives different responses in different situations). The question may be about neural variables or causal relationships; in each case the tutoring rules are different. Also we have different rules for each stage, DR, RR and SS.

5.3 Organization of the Instructional Planner

The instructional planner of CIRCSIM-TUTOR consists of two parts (Figure 11); the lesson planner and the discourse planner. The plan controller monitors the execution of the current plan. The planner can be thought of as a small expert system, which consists of two main parts: a knowledge base and an inference engine [Harmon, 1987]. Thus, I designed the lesson planner to consist of three sets of lesson planning rules, and an inference engine. Also the discourse planner consists of four sets of discourse planning rules and an inference mechanism (discourse network). This section introduces the organization and the main features of the instructional planner briefly.

5.3.1 Lesson Planning. Lesson planning determines the content and sequence of the subject matter to be taught in a single lesson [Murray, 1988; Brecht, 1989; Russell, 1988]. The lesson planning in CIRCSIM-TUTOR consists of two phases: goal generation and plan generation. The generation of the lesson goals is guided by a set of explicit domain-dependent heuristics (goal generation rules), and the lesson plans are determined by applying two set of rules, rules for selecting strategies and rules for selecting tactics. As a result the lesson planner does hierarchical lesson planning with its three sets of rules; at the topmost level it generates lesson goals, and then it expands one of the goals into a set of subgoals (a plan) at the next level.

The generated goals will be saved in the goal stack and the subgoals in the subgoal stack. The lesson planner must update the goals dynamically as the student model changes. The details of the lesson planning process will be explained in Chapter 6.

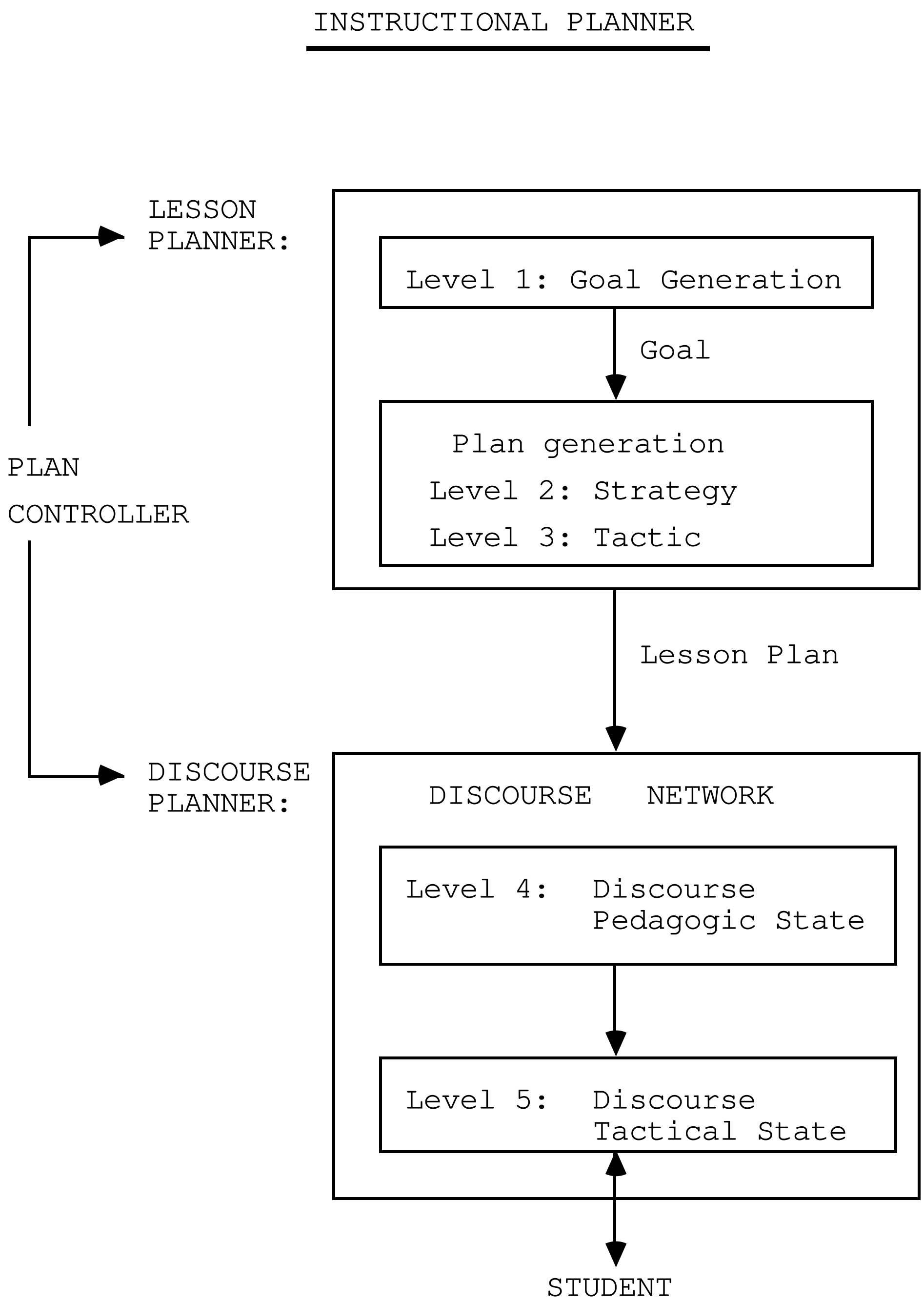

Figure 11. Instructional Planner

5.3.2 Discourse Planning. Discourse planning is a mechanism for planning communicative actions between the tutor and the student within a lesson [Woolf, 1984; Winkel et al., 1988]. CIRCSIM-TUTOR communicates with the student in natural language. Thus, the discourse planner must interact with the student modeler, the screen manager, the input understander, and the text generator using a flexible control mechanism. This control mechanism resides in its discourse network.

The network consists of two levels; the top level of the network specifies pedagogic decisions and the lower level consists of a set of discourse tactical states, the execution of which causes text generation, student model updates, and moves to the other states. It represents the discourse planning rules and the control mechanism in explicit form. The rules include all the necessary information to carry out the discourse with the student, and the control mechanism is also specified within the rules; two sets of default rules manage the fixed control flow, and two sets of meta rules handle dynamic control flow. The discourse planning will be explained in detail in Chapter 7.

5.3.3 Plan Monitoring. AI research on planning emphasizes that execution of a plan requires some monitoring [Charniak and McDermott, 1986]. In the recent robot planning systems [Wilkins, 1988; Swartout, 1988], the plan monitoring can be done by inserting monitoring steps in the plan, which behaves like a student model in instructional planning. In an ITS, since the student's learning status is unpredictable, the planner also needs to monitor the execution of the plan and revise the plan if necessary. As a result, plan monitoring should occur whenever there is a change in the student model. Plan revision may occur when the current plan is completed or when the student takes the initiative.

For the current version of CIRCSIM-TUTOR, the planner monitors the student problem solving in two different places. First, when the student enters predictions in the prediction table, the planner monitors the student's entries in the table and interrupts with some warning messages if the student violates the system constraints. The messages are designed by the experts, to help the students in their problem solving. The system gives different messages depending on the procedure, the variables, and the stages. Second, the planner monitors the student answer at each step during the tutoring session, by referring to the student model, and then decides what to say next; give a hint, give an answer, or continue with the next topic, etc. When the student takes control by asking a question during the tutoring session, the planner suspends the current plan, carries out the student's request and then simply resumes the suspended plan by asking the previous question again.

5.4 Summary

This chapter first discussed the design issues for building an instructional planner, the capabilities, the levels of planning, and the mixed-initiative strategy. Then it displayed a sample dialogue extracted from the transcript of actual tutoring session, and described a scenario, which is an important tool for building the planner and the other components of the system. The chapter concludes with a brief introduction to the planner. The overall architecture of the planner and its main mechanisms, lesson planning, discourse planning, and plan monitoring, are introduced. Figure 11 shows the five levels of the planning process; the three levels of the lesson planner and the two levels of the discourse planner.

CHAPTER VI

THE LESSON PLANNER

The lesson planner decides on the contents of a lesson, based on the student's current knowledge about the domain. The planner has to generate lesson goals, sequence the goals, and select the appropriate planning strategies to create a plan for the current lesson goal. Figure 12 shows the architecture of the lesson planner including the necessary planning steps, student model, and lesson planning rules. The result of the lesson planning is a set of subgoals (a plan), each of which will be the topic for a dialogue with the student. This chapter describes the lesson planner: implementation goals, an architecture, two main mechanisms (i.e., goal generation and plan generation), an example and an algorithm for lesson planning.

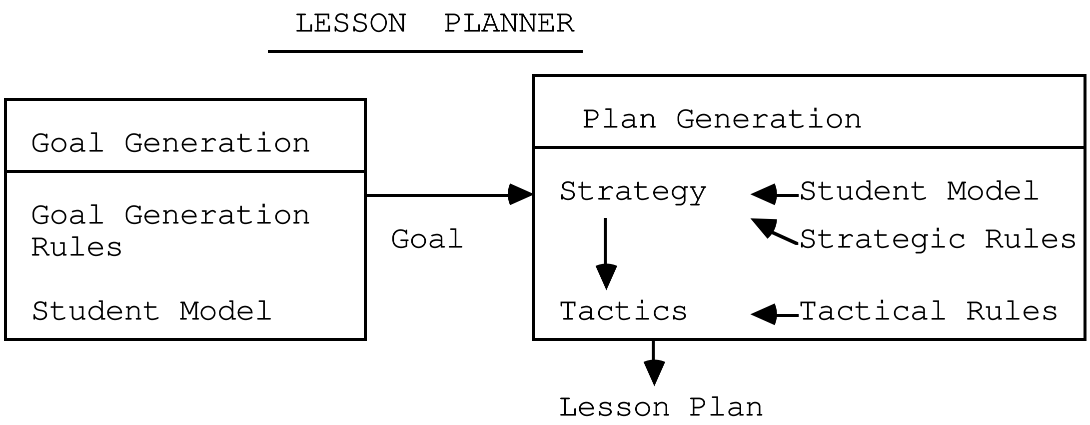

Figure 12. Structure of the Lesson Planner

6.1 Implementation Goals

CIRCSIM-TUTOR begins by asking the student to fill in the Prediction Table and uses the analysis of this information stored in the student model, to generate global lesson plans. In order to generate the plans, the lesson planner must have some capabilities: hierarchical planning, dynamic planning, and rule-based tutoring knowledge.

For example, the planner generates the instructional goals at the top level of abstraction, and then expands one of the goals into a set of smaller subgoals at the next level. So the planner expands the lesson plan hierarchically in a top-down manner, as in IDE-INTERPRETER [Russell, 1988]. Second, the planner can generate different lesson plans for different students, by referring to the student model. In this way, the planner generates plans dynamically and adaptively. Third, the planner must replan when it is necessary. Finally, the planning knowledge is expressed as a set of explicit rules, and a rule interpreter interprets the rules and selects an appropriate one. This gives a flexibility to the modification of the rules.

The lesson planning mechanism is an essential component of the instructional planner, since the system must generate globally coherent and consistent instruction for the student [Macmillan et al., 1988; Murray, 1990], in such a way that the topics are logically connected with each other, and sequenced and presented in a manner sensitive to the tutorial goals and the student's needs. These are the main implementation goals of the lesson planner.

6.2 Lesson Planning Rules

The contents of the tutoring strategies are extracted from the transcripts of the human tutor and student interaction, and we need to encode them explicitly in the program as rules. The prototype tutor was written in Prolog which has a built-in inference capability, but in our Lisp implementation I had to design an interpreter to understand rules. I designed this part as a *production system*, which consists of a rule interpreter and a set of rules. This is the most common approach used in expert systems [Hasemer and Dominque, 1989]. In this section, I will describe in detail how I designed the rule formal, and then implemented the rule interpreter to parse the rules.

6.2.2 The Rule Interpreter.

The rule interpreter consists of three main parts: its main loop, its working memory, and its pattern matcher. The working memory is crucial to the operation of the rule interpreter, because the working memory holds an initial representation of the problem that the system is trying to solve. Each time around the loop, the contents of the working memory will be compared to the antecedent of the rules, and then will fire only one rule if it matches. If an antecedent matches with the working memory, the consequent will be executed, and the content of the working memory will be changed for the next inference. The matching cycle will continue until no rules match. At this point the interpreter halts, and the content of the working memory is the desired result for the given problem.

The interpreter is built using LISP macro functions, which understand and interpret the rules for the system. As a result the rules can be written, not as Lisp code, but in any free format as long as the rule interpreter can understand them. I designed the rules with three parts: the name part of the rule, the antecedent part, and the consequent part. For example, *(Rule_name: (antecedent) => (consequent))*. My motivation was to make the system efficient in writing source code, and also make it possible for our expert tutors to read the rules and make modifications easily. If we want to change the format of the rules, then we need to change the interpreter to recognize that specific form. The next subsection will describe how to actually encode the rules.

6.2.2 How to Encode the Lesson Planning Rules. The lesson planner uses three sets of lesson planning rules (goal generation rules, strategy rules, and tactical rules). I designed the rules in an *if then* format, in a separate file to avoid hard coding them in the program. Therefore, it is easy to add, delete, and modify rules without restructuring the whole program. The general form in which the rules are written is *if X then Y*. Here X is the antecedent or left-hand side of the rule and Y the consequent part or right-hand side of the rule. Both the antecedent and the consequent may contain one or more terms.

For example, assume that the student made an error in predicting the variable TPR. One of the goal generation rules applies; if the student does not know TPR, then build the lesson goal, *tutor TPR about the neural control*. This rule can be expressed as *(G_Rule1: ((do-not-know TPR) => (neural-control TPR)))*. If the current lesson goal is *teach the causal relationship between RAP and SV, and the student does not know the direction*, then this rule can be written as *(S_Rule1: ((causal-relation)(do-not-know direction)) => (tutor-causality)))*. This is the strategy rule for dealing with non-neural variables. Assume that if the strategy rule is *tutor-causality*, then the corresponding tactical rule is to teach *determinants, actual-determinant, relation, and value.* This rule can be written as *(T_Rule1: ((tutor-causality) => (determinants) (actual-determinant) (relation) (value)))*.

Currently, there are about 50 goal generation rules, 20 strategy rules, and 20 tactical rules that handle DR, RR and SS phases, and for procedures 4, 6 and 7. The rules may need to be extended to handle the other procedures.

6.3 Lesson Planning

Instructional planning centers around instructional goals. There are two kinds of goals in the system: lesson goals and discourse goals. The lesson planning generates the lesson goals, the knowledge that the system intends the student to acquire through the tutoring session. This section describes how to generate the lesson goals, and how to develop a lesson plan for the each of the goals. The two main mechanisms of the lesson planning process, goal generation and plan generation, are explained below.

6.3.1 Goal Generation. CIRCSIM-TUTOR generates instructional goals based on the student's knowledge demonstrated as entries in the Prediction Table. The generation of the goals is guided by a set of explicit goal generation rules designed by our experts (Joel Michael and Allen Rovick), which ensures that the most serious misconception is selected and tutored first. For example, suppose the student made wrong predictions in the table for the variables, TPR and SV. The student modeler has determined, from its analysis, that the student is confused about the mechanism controlling TPR and the causal relationships between RAP and SV and SV and CO. So the lesson planner retrieves the information from the student model, applies the goal generation rules (see Figure 13), and generates the lesson goals dynamically. The result is a set of lesson goals in the goal stack (see Figure 14).

Goal Generation Rules

1. IF Current Primary Variable is CC and
   Student Answer is not NOCHANGE for TPR
   Then Build Lesson Goal (NEURAL-CONTROL (TPR))

2. IF Current Primary Variable is RAP and
   Student does not know the CAUSAL-RELATIONSHIP
   between RAP and SV
   Then Build Lesson Goal (CAUSAL-RELATION (RAP, SV))

3. IF Current Primary Variable is RAP and
   Student does not know the CAUSAL-RELATIONSHIP
   between SV and CO
   Then Build Lesson Goal (CAUSAL-RELATION (SV, CO))

Figure 13. Goal Generation Rules

Order	Lesson Goals
1.	NEURAL-CONTROL (TPR)
2.	CAUSAL-RELATION (RAP,SV)
3.	CAUSAL-RELATION (SV, CO)

Figure 14. Generated Lesson Goals in the Goalstack

The goal generation is significant in many ways; the goals are generated dynamically and adaptively; the goals are sequenced in the order that the experts tutors this material;

the goals provide a global context that remains coherent and consistent throughout the tutoring session, unless the goals are revised. New goals can also be generated, which tutor the student about a common misconception (a bug), if the student modeler detects such a misconception. The goals remain in force until they are changed by the planner dynamically (because of changes in the student model).

6.3.2 Plan Generation. The second stage of the lesson planning is the plan generation mechanism, which creates the instructional plan by applying two sets of rules, rules for selecting tutorial strategies to achieve the selected goal and rules for selecting pedagogic tactics to execute those strategies. Strategy rules (Figure 15) describe the tutorial approach from a domain-independent point of view. These include tutoring prerequisites before the material they underlie, reminding the student about relations between two parameters, explaining the definition before tutoring about it, and so on. Tactical rules (Figure 16) also represent a domain-independent tutorial approach; they involve asking about concepts and relations between the concepts.

For instance, if the goal is *teach the causal relationship between the two parameters,* then the fired strategy rule is *tutor the causality*, and this then fires the pedagogic tactical rule: *ask about: determinants, actual determinant, relationship, and correct value.* The result is a hierarchical goal tree (Figure 17). Thus the current goal is ultimately refined into four subgoals by two-step goal transformations. In order to solve the current goal, all the subgoals must be solved. This is the well-known AI problem-reduction technique, which transforms a goal into a set of immediate subproblems by a sequence of transformations [Barr and Feigenbaum, 1982]. The four subgoals generated at the tactical level are the current plan for the goal. These are kept in a subgoal stack (Figure 18), which is used by the discourse planner to pick the next topic.

```
            Strategic Rule

 1.   If the Goal = CAUSAL-RELATION  and
        Student does not know       and
        direction is incorrect
      Then Strategy = TUTOR-CAUSALITY

 2.   If the Goal = CAUSAL-RELATION  and
        Student does not know       and
        direction is correct
      Then Strategy = REMIND-RELATION

 3.   If the Goal = NEURAL-CONTROL  and
        this is the first procedure
      Then Strategy = DEFINE-TUTOR-NEURAL
```

Figure 15. The Strategy Rules

Tactical Rule

1. If Strategy = TUTOR-CAUSALITY
   Then Tactic = DETERMINANTS,
   ACTUAL-DETERMINANT,
   RELATIONSHIP,
   VALUE

2. If Strategy = TUTOR-NEURAL-CONTROL
   Then Tactic = MECHANISM,
   VALUE

3. If Strategy = TUTOR-SS-PHYSICAL-VARIABLE
   Then Tactic = VALUE-DR,
   VALUE-RR,
   VALUE-SS

Figure 16. The Tactical Rules

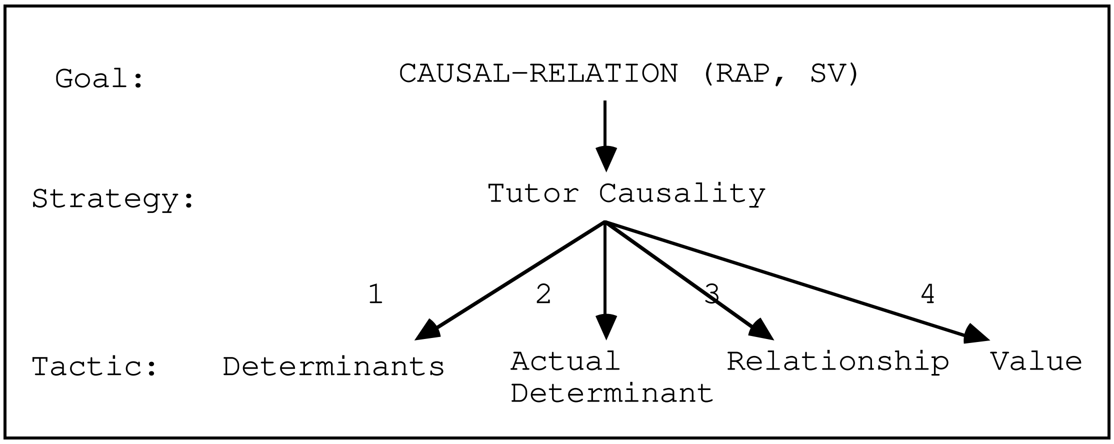

Figure 17. Generated Plan for "causal_relation(RAP,SV)"

Order	Subgoals
1.	Determinants
2.	Actual-determinant
3.	Relation
4.	Value

Figure 18. The Subgoal Stack

6.4 An Example

This section presents one lesson planning cycle as an example. It gives a top level algorithm along with its pseudo code, in order to clarify the functioning of the lesson planner. Figure 19 shows an example of the lesson planning process for the *causal-relationship between RAP and SV*. From the top of the Figure, the goal generation step is described with its other information: student model, rules used, goal stack, and current goal. Then the plan generation step is described in two steps, the strategic and the tactical steps. Since the lesson plan is carried by the discourse planner, the lesson planner suspends after generating a plan. It waits for the discourse planner to be activated and to interact with the student. When the plan controller sends a wake-up signal, then the lesson planner gets reactivated and continues with the next goal in the goal stack, if there is any. Plan monitoring begins when the discourse planner starts to plan the topic for the student.

Goal Generation

Rules Used:

Student Model: do-not-know (SV) — DR_G_Rule8

Goal Stack: Causal-relation (RAP, SV)
Causal-relation (SV, CO)

Current Goal: Causal-relation (RAP, SV)

Plan Generation

Rules Used:

Strategy: Tutor-causality — DR_S_Rule1

Tactics: (determinants) — DR_T_Rule6
(actual-determinant)
(relation)(value)

Subgoal Stack: (Plan) — (determinants)
(actual-determinant)
(relation)(value)

Discourse Planner

executes "determinants of SV"

Plan Monitoring: Waits for the student response

Figure 19. An Example of Lesson Planning

The lesson planning cycle can be summarized as the following top level algorithm (Figure 20 contains the Lisp pseudo code for the algorithm). In this algorithm, the function Main calls the screen manager, which displays some introductory messages first, and then draws the Prediction Table, the tutor window and the student window. Then, Main calls the controller which repeats the problem solving and the student evaluation cycle, until all the phases are done (DR/RR/SS). If the the student answers to a given phase are all correct, then it continues with the next phase. If not, then it calls the lesson planner to begin the tutoring session. The lesson planner generates the goals first, and then repeats the planning cycle until there are no more goals in the stack. Step 3.2.3 is carried out by the discourse planner, and it will be explained in detail in the next chapter.

<u>Top</u> <u>Level</u> <u>Algorithm</u>:

1. Main:
   - 1.1 Call the screen manager to set up environments.
   - 1.2 Call the controller.
2. Controller:
   - 2.1 Repeat until all phases are complete (DR/RR/SS).
     - 2.1.1 Ask the problem solver to solve the problem.
     - 2.1.2 Get the student inputs from the Prediction Table.
     - 2.1.3 Check the student model.

2.1.4 If there is any error, go to step 3.

End-Repeat.

3. Lesson planner:

3.1 Generate lesson goals.

3.2 Repeat until no more goals in the stack.

3.2.1 Take one goal and build lesson plan.

3.2.2 Call the discourse planner.

3.2.3 Remove the current goal from the stack.

End-Repeat.

```
(defun main ()

   (screen-manager set-up-environments)
   (controller))


(defun controller ()

   (repeat-until-all-phases-done       ; DR/RR/SS
      (problem-solver  procedure-number 'stage)
      (get-student-input-from-prediction-table)
      (check-student-model)
      (if (any-error) lesson-planner)))


(defun lesson-planner ()

    (generate-goals)
    (check-lesson-goals))


(defun check-lesson-goals ()

    (build-lesson-plan   (car *lesson-goals*))
    (call-discourse-planner  *subgoals*)
    (check-lesson-goals  (cdr *lesson-goals*)))
```

Figure 20. Pseudo Code for Lesson Planning

6.5 Summary

This chapter has discussed the implementation goals of the lesson planner that provides globally coherent and adaptive instruction to the student. The design of the rule interpreter and its usage is described with an example. The three levels of the lesson planning process are explained in detail with the rules and tutorial strategies. Finally, the top level lesson planning algorithm is explained along with its Lisp pseudo code to clarify the functioning of the lesson planner.

The system can run in two different modes: a tutor version and a student version. The tutor version displays necessary information about the current lesson planning situation on the screen: the current goal, subgoals, strategies, tactics, and rules used. The display of this internal process may be useful in order to understand the lesson planning mechanism better.

CHAPTER VII

THE DISCOURSE PLANNER

The discourse planner is responsible for controlling interactions between the tutor and the student. It needs to decide how the tutor should respond to a student with a given problem [Woolf, 1984; Winkels et al., 1988]. This discourse strategy must be planned explicitly by the discourse planner, so that the system can enter into flexible and coherent interactions. GUIDON [Clancey, 1982] and MENO-TUTOR [Woolf, 1984] provide sophisticated dialogue management by selecting pre-stored discourse plans, but they lack globally controlled plans and cannot allow student initiatives.

In CIRCSIM-TUTOR, the discourse planner is combined with the lesson planner, so that the discourse planner receives a global lesson plan from the lesson planner. The plan controller monitors the execution of the plan and forces the discourse planner to suspend or resume the current plan when the student takes control. The planner consists of sets of discourse planning rules and a two level discourse network, which is similar to MENO-TUTOR in its control structures, but with more functional differentiation (see section 7.2.2).

7.1 Implementation Goals

The main function of the discourse planner is to receive a lesson plan from the lesson planner and decide how to present it to the student using its discourse strategy rules. How to present the generated lesson plan requires sophisticated discourse management and flexible interactions with the other components of the system. The following are the main goals of building the discourse planner.

First, one of the main functions of the planner is to decide how to present the selected lesson to the student using natural language. Thus it needs to interact with other components of the system: the student modeler, the text generator, the input understander, and the screen manager. Second, the system must correct the student's misconceptions about the given problem. Thus, it must give immediate feedback on all the student's answers throughout the tutoring session, although it can be argued that sometimes alternative responses should be given instead of complete feedback [Galdes et al., 1991]. Third, if the student fails to give a correct answer to the tutor, the planner must provide an alternative learning path to the student, such as a hint. So the planner needs to decide what to do next at each point during the tutoring session. Fourth, the discourse strategies need to be explicitly expressed, rather than procedurally encoded in the program, such as in IDE-INTERPRETER [Russell, 1988], which procedurally encodes most discourse actions in its instructional units. The explicit representation provides an easy way of modifying the strategies. In CIRCSIM-TUTOR, these strategies are expressed as explicit discourse planning rules. Fifth, the planner should accept the student's questions and comments, and respond with an appropriate answer. CIRCSIM-TUTOR can respond to some kinds of student initiative at the moment.

7.2 Architecture of the Discourse Planner

The discourse planner interfaces with many other components of the system to control the question and answer dialogue. For instance, the input understander passes the student input to the planner, then the planner decides what to do next according to its tutoring strategy, and returns a response through the text generator in natural language. Therefore, the planner needs some knowledge of how to tutor the student at each point of tutoring sessions. In CIRCSIM-TUTOR this tutoring knowledge is expressed as the discourse planning rules, which has been extracted from the experts. The rules are organized as a two level discourse network. This section explains how I designed the rules and the overall structure of the network in detail.

7.2.1 Flow Chart Approach. Meta knowledge is knowledge about knowledge [Davis and Buchanan, 1987]; what you know and don't know (operational meta knowledge), and how you do things (control meta knowledge). The operational meta knowledge is needed to recognize a question outside the limits of the system. It can be ignored in the discourse planner, since the input understander receives such a question or answer and responds with *I don't understand, please rephrase*. The control meta knowledge controls how the system interacts with the student; it is based on our observations about how the human expert tutors the student. The integration of this knowledge into the system ensures that it appears to ask questions in a logical order.

The basic representation of the control meta knowledge in CIRCSIM-TUTOR is the *flow chart*. This is a model of *what* the expert does and *when* he does it. For our system, Allen Rovick designed several flow charts (see Figure 21), each of which is used for tutoring the student in a different situation. We need different tutoring strategies for handling neural variables, causal relationships, the regulated variable, logical relationships, and so on. The strategies dealing with the neural variables are different in all three phases (DR/RR/SS), as are those for the other variables. Figure 21 is the one that tutors the non-neural variables in DR. The content of the questions is determined by the lesson planner and passed on to the discourse planner, which must then decide how to express this content, determining whether to ask a question, give an answer, and so on. After the chart was created, I, the knowledge engineer, encoded this information as discourse planning rules. The next step was to create a sophisticated inference mechanism that can utilize these rules.

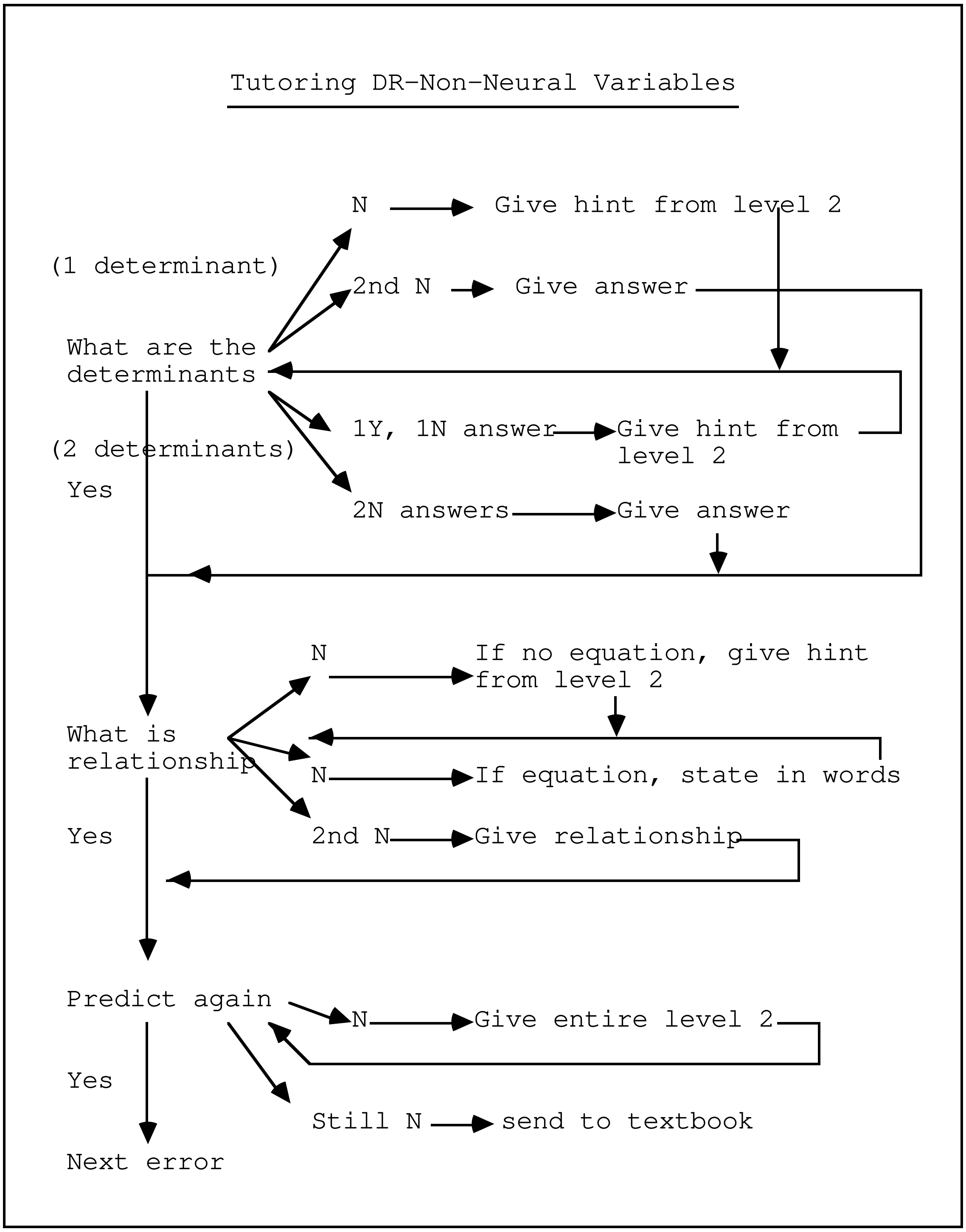

Figure 21. The Flow Chart for Tutoring Non-Neural Variables in DR

7.2.2 Discourse Network. The network is the main knowledge structure of the discourse planner. It consists of states, links, and arcs (see Figure 22). The states represent tutorial actions, the arcs imply state transitions, and the links indicate hierarchical dependencies; a state at the tactical level represents the refinements of the level above. Three important mechanisms need to be discussed: levels of planning, representation of the tutorial states, and control structures. The network cannot be considered without the control structure, since it integrates the other two mechanisms.

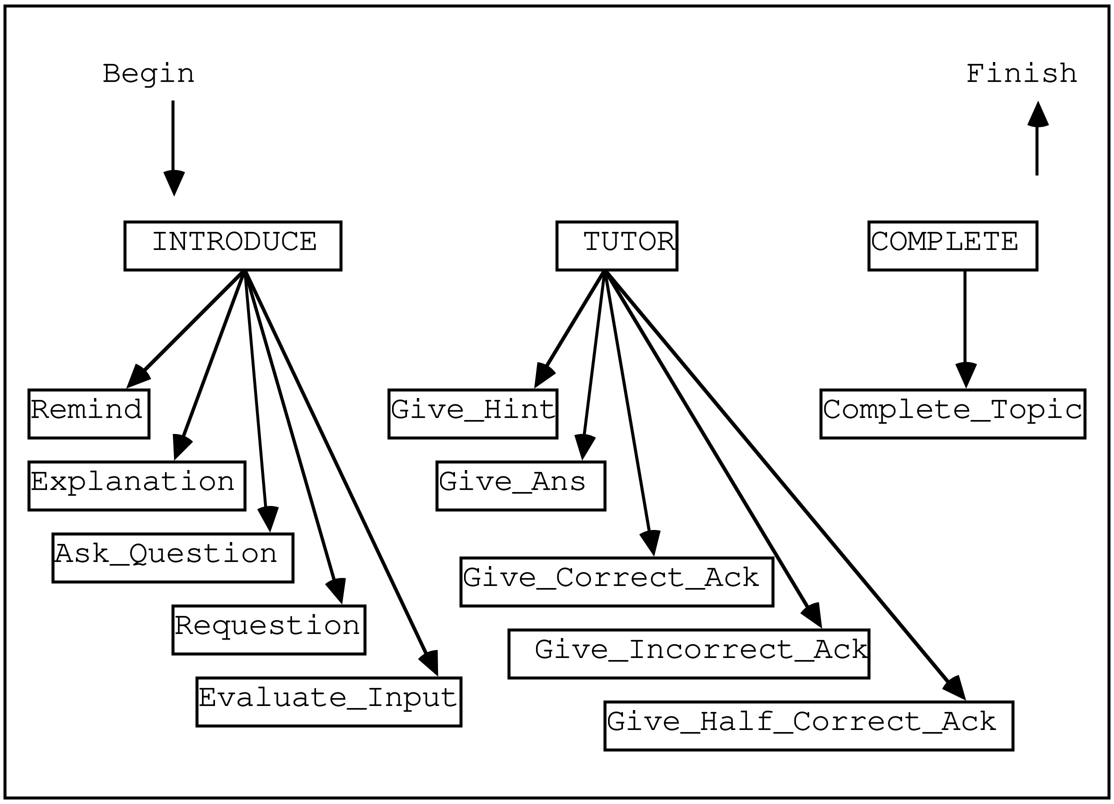

Figure 22. The Discourse Network

A. Levels of Planning. The discourse planner is divided into two planning levels: pedagogical and tactical. The pedagogical level makes decisions about the style of tutoring; introduces a topic, remediates the student's misconceptions, and completes a topic. The discourse action begins with the pedagogical level, *introduce* state, and then it traverses the network and finishes one topic as it reaches the *complete* state. The tactical level chooses an expository style to implement the pedagogy; *question the student, give acknowledgement, or give an answer*. The states at this level are refinements of the states at the pedagogic level.

B. Representation of Discourse Strategies. The second important mechanism is the representation of the tutorial strategies in the form of states. The discourse strategies were then extracted from the flow chart and expressed as discourse rules. The rules are written as a frame-like structure using Lisp macro functions, which represent the states in the network (see Figure 23, Figure 24, Figure 25, Figure 26). The states are divided into default states and meta states, and each state is further divided into pedagogic and tactical states. Each state consists of a state name and slots. The slots contain necessary information to perform text generation or explicit control mechanisms; the slots in the meta states contain preconditions and indicate the next state to move to, and the default tactical states contain text style and content to generate a natural language sentence. In Figure 25, the execution of *Ask_Question* state will cause the text generator to generate a question, and then move on to the next default state, *Eval_Input*. The slots also contain a register to keep track of the completion of the topic, and a flag to update the student model.

```
     Pedagogic Default Rule

 (Pedagogic_default   *introduce*
     (subgoal          current-task
      update           topic-completed
      next-state      *tutor*))

 (Pedagogic_default   *tutor*
     (subgoal          current-task
      update           topic-completed
      next-state      *complete*))
```

Figure 23. The Pedagogic Default Rule

```
       Pedagogic Meta Rule

(Pedagogic_meta      *m_tutor*
    (precondition     topic-completed
     prior-state     *tutor*
     next-state      *introduce*))

(Pedagogic_meta      *m_complete*
    (precondition     no-more-topics
     prior-state     (*introduce*  *complete*)
     next-state      *stop*))
```

Figure 24. Some Pedagogical Meta Rules

```
          Tactical Default Rule

(Tactical_default    *ask_question*
    (text-style       question
     content          current-task
     update           nil
     next-state      *eval-input*))


(Tactical_default    *give_answer*
    (text-style       give-answer
     content         (current-task correct-answer)
     update           student-model
     next-state      *complete-topic*))
```

Figure 25. Tactical Default Rules

```
          Tactical Meta Rule

(Tactical_meta       *m_correct*
    (precondition     correct-response
     prior-state     *eval-input*
     next-state      *correct-ack*))


(Tactical_meta       *m_incorrect*
    (precondition     incorrect-response
     prior-state     *eval-input*
     next-state      *incorrect-ack*))
```

Figure 26. Tactical Meta Rules

C. Control Structure. The discourse control in the network can be divided into a default control structure and a meta control structure. The default control is specified in the default states, so that the tutor moves from one state to another according to a pre-determined path. The meta control abandons the default path and moves to the state that is specified in the meta-rule. The system checks the meta-rules first and if none of the meta-rules fire, then the control flow will follow the default path. This control path is hidden in Figure 22, because the exceptional behavior by the meta-rules can not be predicted in advance. For example, the *Eval_Input* state will be selected right after the *Ask_Question* state as a default path, but the next state is unpredictable, since the student answer could be correct, wrong, or partially correct. This mechanism enables the dynamic behavior of the discourse planner.

The main disadvantage of earlier discourse management networks [Woolf, 1984; Clancey, 1982] is that they needed to be coupled with some other control mechanism, such as an agenda and an external memory to provide a topic. In CIRCSIM-TUTOR, since the lesson planner provides a globally coherent lesson plan, the network itself can function solely for delivery purposes while keeping all the advantages of the discourse management network, such as flexible discourse control and explicit representation of discourse strategies.

7.3 Discourse Planning

Discourse planning in CIRCSIM-TUTOR is managed by a simple algorithm. It iterates through the states until a topic becomes complete. Either the student responds with a correct answer or the tutor gives the answer. This section describes important features of the discourse planning: the discourse goals, a discourse planning algorithm, an example of text generation, and an example of a student initiative.

7.3.1 The Discourse Goal. The discourse planner needs a goal to tutor the student. This goal can be found in the subgoal stack, which the lesson planner has produced. In Figure 18, the subgoals are sequenced by number, so that the discourse planner can carry them out in that order. When the planner finishes carrying out one of the subgoals, it will be removed from the stack, and the planner picks the next one. This cycle continues until the stack is empty, or is suspended by the plan controller in favor of a student initiative.

7.3.2 The Discourse Planning Algorithm. The discourse planning algorithm is a simple iteration. It receives a subgoal stack from the lesson planner, picks one of the subgoals, and iterates through the states in the network until the goal is completed. The cycle is repeated with each subgoal in turn until the stack is empty. The most important feature of the algorithm is flexible transition between the two level planning process; if the current level is a default or meta strategy then process the upper level function, else process the lower level function; if the next state is not specified in a tactical state then pop up to the upper level and check meta rules, and so on. The following is the top level discourse planning algorithm and Figure 27 shows the pseudo code for the discourse planning algorithm.

The Top Level Discourse Planning Algorithm:

1. Repeat until the subgoal stack is empty.
   Pick one subgoal and call execute-one-subgoal.
   End-Repeat.
2. Execute-one-subgoal:
   Repeat until topic completed
      if strategy then process-upper-level
      if tactical then process-lower-level
   End-Repeat.

7.3.3 Generating Natural Language Sentence. The tactical default states have slots containing information for the text generator. When the planner processes the states, the text-style and content slots will be extracted from the current state. For example, assume that the planner is processing the *ask_question* state (Figure 25), while the text-style slot contains *question* and the content slot contains the current-task, such as *determinant (SV)*. Binding these two slot values provides us with a logic form, *(question (determinant (SV)))*, which will be passed to the text generator, which generates the sentence, *What are the* *determinants of SV?* Then the screen manager will display the sentence in the tutor window.

The logic form may need to be extended to generate richer sentences, since this kind of the logic form only contains information about a particular task or the solution of a problem. The text generator may need to collect more information from many other sources, the domain knowledge base, the student model, the dialog history, and so on.

```
(defun discourse-planner ()
   (repeat until no-more-subgoals
          (execute-one-subgoal)))

(defun execute-one-subgoal ()
   (repeat-until (STOP)
      (case (get-level (level))
          ((strategy)      (process-upper-level))
          ((meta-strategy) (process-upper-level))
          ((tactical)      (process-lower-level))
          ((meta-tactical) (process-lower-level)))))

(defun process-upper-level ()
   (check-start)
   (cond  ((current-state = meta-strategy)
           (get-next-state))
          ((topic-completed) (STOP)
           (get-next-state)

(defun process-lower-level ()
     (cond ((current-state = meta-tactical)
            (get-next-state)
           ((topic-completed) (get-next-state)
            (call-text-gen)   (get-next-state)))
     (if (next-state = nil) (pop-level-up)))
```

Figure 27. The Pseudo Code for Discourse Planning

7.3.4 How to Recognize a Student Initiative. CIRCSIM-TUTOR allows student initiatives during the tutoring session. So the planner must understand whether the student response is a question or an answer by checking the input logic form, which is being passed from the input understander. For example, if the input understander passes a logic form, *(answer (determinant SV)(RAP CO))*, the first item of the list, *answer*, indicates that this is an answer. The second item of the list, *(determinant SV)*, is the current topic, and the third item, *(RAP CO)*, is the student answer. Let's assume that the tutor asks the question, *What are the determinants of SV?* and the student responds with *I don't know about SV*. Then the input understander recognizes this as an implicit question and returns a logic form, *(question (do-not-know) (SV))*. The planner receives the logic form and recognizes that this is a student initiative, so it suspends the current plan and carries out the student request; asks the problem solver to get the definition of SV from the knowledge base, and then asks the screen manager to display it.

Student initiatives can involve a broad range of questions. It requires efforts from many components of the system to give helpful responses; the input understander must understand the question, the problem solver must get an answer, and the planner must keep track of the current plan and carry out the student request.

7.4 Trace of Discourse Transition

Figure 28 shows a short trace of a sequence of discourse transitions. The short arrows represent the pedagogic level transitions; the long arrows represent the tactical level transitions; and the double arrows represent the meta level transitions. The left side of the figure shows the processing of states, and the right side of the figure shows the discourse actions resulting from visiting the states.

The tutor begins by asking a question, then it moves to the evaluate state by the default control rule. At this time, the student responds with a half correct answer, which is recognized by the *meta tactical rule3*, which forces a move to the *half-correct* state. This state produces an acknowledgement and then another meta rule fires, which recognizes that this is the first try. So the meta rule forces a move to the *give-hint* state, which produces a hint. Since there is no default and a meta rule applies, the control pops up to the upper level and checks whether the topic has been completed. If not, then control goes back to the *introduce* state again, and moves down to the tactical level. This time the *requestion* state is selected, since this is the second try on the same topic.

One of the default paths is *ask-question* then *eval-input*. This can be considered as a partial discourse plan, because this plan will be overriden by the meta rules. Thus the overall discourse planning mixes fixed partial planning with dynamic meta planning.

Current Topic: Determinants of SV

->,=>: Pedagogic Level
-->, ==>: Tactical Level (-> Default , => Meta)

Discourse States	Discourse Action
-> INTRODUCE	
--> Ask-question	Tutor: What are the determinants of SV?
--> Eval-input	Student: RAP and CO
==> Meta-tactic3 (Incorrect-one)	
--> Half-Correct	Tutor: RAP is correct, but CO is not a determinant of SV.
==> Meta-tactic6 (First-try)	
--> Give-hint	Tutor: Remember. SV is the amount of blood pumped per beat.
-> TUTOR	
=> Meta-pedagogic (Not-completed)	
-> INTRODUCE	
--> Requestion	Tutor: What is the other determinant of SV?

Figure 28. Trace of the Discourse Transition Process

7.5 Summary

This chapter began by introducing the discourse rules and the discourse network. The discourse rules originated in the flow chart as a tutorial strategy, and I have transformed them into rules and expressed them as frame-like structures using Lisp macro functions. The rules include all the necessary information to generate a natural language sentence, and the control mechanism is also specified within the rules.

The network consists of two levels: the pedagogy level and the tactical level. The pedagogy level makes decisions about the style of tutoring and the tactical level decides on the expository style to implement the pedagogy. The execution of the tactical states causes text to be generated, updates the student model, and moves to the other states. The states represent explicit discourse planning rules and an explicit control mechanism.

Some of the important discourse planning features are introduced in the third section; discourse goals, the discourse planning algorithm, communication with the text generator, recognizing the student initiative. A short trace of example discourse state transitions is displayed in the fourth section.

The system provides two different running modes: a tutor version and a student version. The tutor version displays the subgoal stack, the current topic, and the discourse transition. As the dialogue proceeds, the subgoal stack gets updated, and displays all the states that have been visited including the current one.

CHAPTER VIII

CONCLUSION

8.1 Significance of this Research

This thesis describes the design and development of an instructional planner for a Physiology ITS, CIRCSIM-TUTOR. The planner has several significant features.

First, the planner combines two different instructional planning approaches: *lesson planning* and *discourse planning*. Lesson planning produces global lesson plans, which will be carried out during the discourse planning stage. This approach provides us with many advantages over other instructional planning systems, such as MENO-TUTOR [Woolf, 1984] and IDE-INTERPRETER [Russell, 1988].

Second, the planner plans *dynamically* based on the inferred student model; it *generates* plans, *monitors* the execution of the plans, and *replans* when the student interrupts with a question during the tutoring session. This approach provides *adaptive* instruction, so that it is better suited for tutoring individual students than CAI systems which produce fixed instruction.

Third, the pedagogical knowledge is extracted from the experts and represented *explicitly* as rules, lesson planning rules and discourse planning rules, in separate files. This way it is possible to add, delete, and modify the rules easily without restructuring the whole system. The rules are used to generate lesson plans and to control discourse strategies. The system interprets the rules and builds the lesson plans or returns an appropriate discourse action.

Fourth, the planner plans at different levels of the *hierarchy*; the higher level is a simplification or abstraction of the plan (lesson goals) and the lower is a detailed plan (subgoals), sufficient to solve the problem. This planning technique prevents development of unnecessary plans in advance and has been implemented successfully in several ITS systems [Murray, 1988; Russell, 1988].

Fifth, the planner allows *student initiatives* during the tutoring session. If the student asks a question the planner suspends the current plan, carries out the student request, and then resumes the suspended plan. This is another advantage of CIRCSIM-TUTOR over earlier dialogue management systems, such as MENO-TUTOR [Woolf, 1984], which does not allow student questions during the tutoring session.

Finally, the planner acts as a controller for the system, so that it controls all the other components of the system. Since one of the main goals of CIRCSIM-TUTOR is to provide a natural language interface, the discourse planner is designed not only to provide sophisticated discourse control, but also to create the internal logic forms for the text generator to generate the sentence. A short tutoring scenario is introduced, which came from a transcript of human tutor and student interaction, to explain the internal process of the system.

8.2 Future Research

Since the student modeler was not fully implemented by its designer (only DR in procedure 4), I had to implement a temporary student model for the planner. This model is limited to the overlay strategy, so the planner can support tutoring on the overlay errors only, not the bugs. The tutoring strategy for the bug library has not been developed yet, so the system cannot tutor the student about bugs at the moment.

Another very important tutoring strategy is giving a level 2 (more detailed knowledge) hint during the tutoring session. Giving a hint generally involves many different knowledge sources. In CIRCSIM-TUTOR, the domain knowledge base needs to but does not contain all the knowledge at the detailed level. The input understander and the text generator need to expand their lexicon and logic forms to contain all the variables at the detailed level. The problem solver needs to be able to access the knowledge base and extract a hint, and the planner needs to have a general strategy for deciding the content of the hint for every situation during the tutoring session.

We are currently analyzing student initiatives in transcripts of human tutoring sessions and starting with a couple of simple examples, such as *I don't know*, and *I don't understand about X*. The planner needs to develop its tutoring strategy to support more sophisticated student initiatives. Also the input understander needs to recognize student initiatives, and the problem solver must provide a correct answer.

CIRCSIM-TUTOR supports seven pre-determined problems as a curriculum, so that it does not really require curriculum planning. Our expert tutors are developing many more procedures for the system, which may require sophisticated curriculum planning in future versions of the system.

APPENDIX A

TUTORING RULES IN ENGLISH AND IN LISP CODE

Tutoring rules in CIRCSIM-TUTOR consists of lesson planning rules and discourse planning rules. The lesson planning rules are further divided into the three kinds of rules: goal generation rules, strategic rules, and tactical rules. The discourse planning rules are divided into pedagogic rules and tactical rules. In this section, the complete rules are displayed in both English and Lisp source code.

LESSON PLANNING RULES

1.1 Goal Generation Rules in English

DR Goal Generation Rules

(SYMBOL: X, Y --> Z; X and Y are determinants of Z

X ==> Y; X is the determinant of Y

X = Y * Z; equation)

1. If primary variable is neural variable
   and all other neural variables are not zero
   Then tutor other neural variables are zero in DR

2. If primary variable is not neural variable
   and all neural variables are not zero
   Then tutor neural variables are zero in DR

3. If RA is primary variable and MAP /= RA
   Then tutor TPR, CO --> MAP and MAP = TPR * CO
   and TPR ==> MAP

4. If RA is primary variable and CO is wrong
   Then tutor HR, SV --> CO and CO = HR * SV
   and HR is zero (neural variable), so CO can change
   only if SV changes

5. If RA is primary variable and SV is wrong
   Then tutor CC, RAP --> SV and CC is zero (neural
   variable), so CO can change only if SV changes

6. If RA is primary variable and RAP is wrong
   Then tutor CO --> RAP and 1/CO ==> RAP

and CO = 0, RAP = 0

7. If CC is primary variable and SV /= CC
Then tutor CC, RAP --> SV and CC ==> SV

8. If HR is primary variable and CO /= HR
Then tutor HR, SV --> CO and CO = HR * SV and
HR ==> CO

9. If HR is primary variable and RAP /= 1/CO
Then tutor 1/CO ==> RAP

10. If HR is primary variable and RAP = 1/CO and CO is wrong
Then remind_update (RAP = 1/CO, CO, RAP)

11. If (RAP is primary variable or if CC is not primary variable)
Then tutor RAP, CC --> SV and RAP ==> SV

12. If HR and RA is not primary variables and CO /= SV
Then tutor SV, HR --> CO and CO = SV * HR and
SV ==> CO

13. If HR and TPR is not primary variables and CO = SV and SV is wrong
Then remind_update (CO = SV, SV, CO)

14. If RA is not primary variable MAP /= CO
Then tutor CO, TPR --> MAP and MAP = CO * TPR
and CO ==> MAP

15. If RA is not primary variable and MAP = CO and CO is wrong
Then remind_update (MAP = CO, CO, MAP)

16. If CC is primary variable and RAP /= 1/CO
Then tutor CO --> RAP and CO ==> 1/RAP

17. If CC is primary variable and RAP = 1/CO and CO is wrong
Then remind_update (RAP = 1/CO, CO, RAP)

<u>RR Goal Generation Rules</u>

1. If baroreceptor already denervated
and all entries in RR are not zero
Then tutor all variables in RR is zero

2. If procedure /= denervate baroreceptors
   and neural variables /= 1/MAP in DR
   Then tutor neural variables
   and neural variables in RR = 1/MAP in DR

3. If procedure = denervate baroreceptors
   and neural variable(s) /= increase
   Then tutor baroreceptor denervation decreases
   afferent input to CNS
   and denervation is equivalent to decreased MAP
   and denervation causes RR which increases neural
   variables

4. If procedure /= denervate baroreceptors
   and CO /= 1/MAP in DR
   Then tutor CO = HR * SV and HR and CC in RR equals
   1/MAP in DR and HR ==> CO in RR

5. If procedure = denervate baroreceptors
   and CO /= increase
   Then tutor CO = HR * SV
   and HR and CC = increase and HR ==> CO in RR

6. If RAP /= 1/CO
   Then tutor CO = 1/RAP

7. If CO is wrong and RAP = 1/CO
   Then remind_update (RAP = 1/CO, CO, RAP)

8. If SV /=RAP
   Then tutor RAP, CC --> SV
   and RAP ==> CO

9. If RAP is wrong and SV = RAP
   Then remind_update (SV = RAP, RAP, SV)

10. If procedure /= denervate baroreceptor
    and MAP in RR /= MAP in DR
    Then tutor effects of reflexes in general
    and MAP = TPR * CO
    and TPR, HR and CC in RR = 1/MAP in DR

11. If procedure /= denervate baroreceptor
    and MAP in RR = 1/MAP in DR and CO is wrong
    Then remind_update (MAP = CO * TPR, CO, MAP)

12. If procedure = denervate baroreceptor and MAP /=
    increase then tutor
    denervation --> increase HR, CC --> increase CO
    and increase TPR and MAP = CO * TPR

SS Goal Generation Rules

1. If procedure = denervate baroreceptors
   and variable in SS /= variable in RR
   Then tutor all variables in DR = 0
   and variable in SS = variable in RR

2. If baroreceptor already denervated
   and variable in SS /= variable in DR
   Then tutor all variables in RR = 0
   and variable in SS = variable in DR

3. If MAP in SS /= MAP in DR
   Then tutor effects of reflexes on regulated variable

4. If variable in DR /= 0
   and variable in SS /= variable in DR
   Then tutor reflex only partially reverses direct
   effects of procedure

5. If variable in DR = 0
   and variable in SS /= variable in RR
   Then tutor variables that are unaffected in DR have
   same value in SS as in RR

1.2 Strategic Rules in English

1. If the goal = tutor causal-relationship
   and direction is incorrect
   Then strategy = tutor causality

2. If the goal = tutor causal-relationship
   and direction is correct
   Then strategy = remind relation

3. If the goal = tutor causal-relationship
   between CO and RAP
   Then strategy = tutor causality for one determinant

4. If the goal = tutor neural control
   Then strategy = tutor neural control

5. If the goal = tutor neural variable and this is
   the first procedure
   Then strategy = remind neural variable

6. If the goal = tutor MAP in RR
   Then strategy = define MAP in RR

7. If the goal = tutor neural variables in RR
   The strategy = define neural variables

8. If the goal = tutor reflex in RR
   Then strategy = tutor effect of reflex

9. If the goal = tutor logic relation in SS
   Then strategy = tutor logic relation

10. If the goal = tutor neural variable in SS
    Then strategy = define neural variable in SS

11. If the goal = tutor reflex in SS
    Then strategy = tutor reflex in SS

12. If the goal = tutor MAP in SS
    Then strategy = tutor compensate

<u>1.3 Tactical Rules in English</u>

1. If the strategy = tutor causality
   Then tactic = ask (determinants, actual-determinant, relation, value)

2. If the strategy = tutor causality for one determinant
   Then tactic = ask (determinants, relation, value)

3. If the strategy = remind-relation
   Then tactic = remind-relation

4. If the strategy = tutor neural control
   Then tactic = ask (mechanism, value)

5. If the strategy = remind neural variable
   Then tactic = redefine DR

6. If the strategy = tutor effect of reflex
   Then the tactic = ask (baroreceptor-reflex, value)

7. If the strategy = tutor neural variable in RR
   Then the tactic = ask (reflex, value)

8. If the strategy = tutor compensate in SS
   Then the tactic = ask (compensate, value)

9. If the strategy = tutor logic relation
   Then the tactic = ask (follow, value)

10. If the strategy = tutor neural variable in SS
    Then the tactic = ask (value-dr, value-rr, value-ss)

2.1 Goal Generation Rules in Lisp Code

DR Goal Generation Rules

```
 1. (G_ruleD1    (cc-sm) => ((neural-control *cc*)))

 2. (G_ruleD2    (hr-sm) => ((neural-control *hr*)))

 3. (G_ruleD3    (tpr-sm)=> ((neural-control *tpr*)))

 4. (G_ruleD4    (cc-sm tpr-sm) =>
                 ((redefine-dr)(neural-control *cc*)
                  (give-dr-neural)(neural-control *tpr*)))

 5. (G_ruleD5    (hr-sm tpr-sm) =>
                 ((redefine-dr)(neural-control *hr*)
                  (give-dr-neural)(neural-control *tpr*)))

 6. (G_ruleD6    (cc-sm hr-sm) =>
                 ((redefine-dr)(neural-control *cc*)
                  (give-dr-neural)(neural-control *hr*)))

 7. (G_ruleD7    (cc-sm hr-sm tpr-sm) =>
                 ((redefine-dr)(neural-control *cc*)
                  (give-dr-neural)(neural-control *hr*)
                  (neural-control *tpr)))

 8. (G_ruleD8    (sv-sm rap-sv sv-co) =>
                 ((causal-relation (*rap* *sv*))
                  (causal-relation (*sv* *co*))))

 9. (G_ruleD9    (sv-sm cc-sv sv-co) =>
                 ((causal-relation (*cc* *sv*))
                  (causal-relation (*sv* *co*))))

10. (G_ruleD10   (rap-sm co-rap rap-sv) =>
                 ((causal-relation (*co* *rap*))
                  (causal-relation (*sv* *co*))))

11. (G_ruleD11   (co-sm sv-co co-rap co-map) =>
                 ((causal-relation (*sv* *co*))
                  (causal-relation (*co* *rap*))
                  (causal-relation (*co* *map*))))

12. (G_ruleD12   (co-sm sv-co co-map) =>
                 ((causal-relation (*sv* *co*))
                  (causal-relation (*co* *map*))))

13. (G_ruleD13   (map-sm co-map) =>
                 ((causal-relation (*co* *map*))))
```

```
14. (G_ruleD14   (sv-sm co-sm cc-sv sv-co co-map co-rap)
          =>    ((causal-relation (*cc* *sv*))
                 (causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))
                 (causal-relation (*co* *rap*))))

15. (G_ruleD15   (sv-sm co-sm rap-sv sv-co co-map)
          =>    ((causal-relation (*rap* *sv*))
                 (remind (*sv* *co*))
                 (causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))))

16. (G_ruleD16   (sv-sm co-sm cc-sv co-map co-rap)
          =>    ((causal-relation (*cc* *sv*))
                 (remind (*sv* *co*))
                 (causal-relation (*co* *map*))
                 (causal-relation (*co* *rap*))))

17. (G_ruleD17   (sv-sm co-sm rap-sv co-map)
          =>    ((causal-relation (*rap* *sv*))
                 (causal-relation (*co* *map*))))

18. (G_ruleD18   (co-sm map-sm sv-co co-map co-rap)
          =>    ((causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))
                 (causal-relation (*co* *rap*))))

19. (G_ruleD19   (co-sm map-sm sv-co co-map)
          =>    ((causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))))

20. (G_ruleD20   (co-sm map-sm co-map)
          =>    ((causal-relation (*sv* *co*))))
                 (remind (*co* *map*))

21. (G_ruleD21   (co-sm map-sm sv-co co-rap)
          =>    ((causal-relation (*sv* *co*))
                 (causal-relation (*co* *rap*))
                 (remind (*co* *map*))))

22. (G_ruleD22   (sv-sm map-sm cc-sv sv-co co-map)
          =>    ((causal-relation (*cc* *sv*))
                 (causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))))

23. (G_ruleD23   (sv-sm map-sm rap-sv sv-co co-map)
          =>    ((causal-relation (*rap* *sv*))
                 (causal-relation (*sv* *co*))
                 (causal-relation (*co* *map*))))

24. (G_ruleD24   (sv-sm co-sm cc-sv co-map co-rap)
          =>    ((causal-relation (*cc* *sv*))
                 (causal-relation (*co* *map*))
                 (causal-relation (*co* *rap*))))
```

```
25. (G_ruleD25   (sv-sm co-sm map-sm cc-sv sv-co co-map
                  co-rap)
           =>    ((causal-relation (*cc* *sv*))
                  (causal-relation (*sv* *co*))
                  (causal-relation (*co* *map*))
                  (causal-relation (*co* *rap*))))

26. (G_ruleD26   (sv-sm co-sm map-sm rap-sv co-map)
           =>    ((causal-relation (*rap* *sv*))
                  (remind (*sv* *co*))
                  (causal-relation (*co* *map*))))

27. (G_ruleD27   (sv-sm co-sm map-sm cc-sv co-map co-rap)
           =>    ((causal-relation (*cc* *sv*))
                  (remind (*sv* *co*))
                  (causal-relation (*co* *map*))
                  (causal-relation (*co* *rap*))))

28. (G_ruleD28   (sv-sm co-sm map-sm rap-sv sv-co)
           =>    ((causal-relation (*rap* *sv*))
                  (causal-relation (*sv* *co*))
                  (remind (*co* *map*))))

29. (G_ruleD29   (sv-sm co-sm map-sm cc-sv sv-co co-rap)
           =>    ((causal-relation (*cc* *sv*))
                  (causal-relation (*sv* *co*))
                  (causal-relation (*co* *rap*))
                  (remind (*co* *map*))))

30. (G_ruleD30   (sv-sm co-sm map-sm rap-sv)
           =>    ((causal-relation (*rap* *sv*))
                  (remind (*sv* *co*))
                  (remind (*co* *map*))))

31. (G_ruleD31   (sv-sm co-sm map-sm cc-sv co-rap)
           =>    ((causal-relation (*cc* *sv*))
                  (remind (*sv* *co*))
                  (causal-relation (*co* *rap*))
                  (remind (*co* *map*))))
```

<u>RR</u> <u>Goal</u> <u>Generation</u> <u>Rules</u>

```
 1. (G_ruleR1    (map-sm) => (rr-reflex *map*))

 2. (G_ruleR2    (cc-sm)  => (neural-rr *cc*))

 3. (G_ruleR3    (hr-sm)  => (neural-rr *hr*))

 4. (G_ruleR4    (tpr-sm) => (neural-rr *tpr*))
```

```
5. (G_ruleR5    (co-sm rap-sm sv-sm) =>
                ((causal-relation (*hr* *co*))
                 (causal-relation-one (*co* *rap*))
                 (causal-relation (*rap* *sv*))))

6. (G_ruleR6    (rap-sm sv-sm) =>
                ((causal-relation-one (*co* *rap*))
                 (causal-relation (*rap* *sv*))))

7. (G_ruleR7    (co-sm rap-sm) =>
                ((causal-relation (*hr* *co*))
                 (causal-relation-one (*co* *rap*))))

8. (G_ruleR8    (rap-sm) =>
                ((causal-relation-one (*rap* *sv*))))

9. (G_ruleR9    (co-sm) =>
                ((causal-relation (*hr* *co*))
                 (causal-relation-one (*co* *rap*))))
```

<u>SS Goal Generation Rules</u>

```
1. (G_ruleS1    (map-sm) => ((give-ssmap)
                             (ss-reflex (*map*)))

2. (G_ruleS2    (cc-sm)  => ((give-ssneural)
                             (neural-ss (*cc*)))

3. (G_ruleS3    (hr-sm)  => (neural-ss (*hr*)))

4. (G_ruleS4    (tpr-sm) => (neural-ss (*tpr*)))

5. (G_ruleS5    (rap-sm) => (logic-relation (*rap*)))

6. (G_ruleS6    (sv-sm)  => (logic-relation (*sv*)))

7. (G_ruleS7    (co-sm)  => (logic-relation (*co*)))
```

<u>2.2 Strategy Rules in Lisp Code</u>

```
1. (S_ruleD1    (causal-relation direction-correct) =>
                (tutor-causality))

2. (S_ruleD2    (causal-relation direction-incorrect) =>
                (remind-relation))

3. (S_ruleD3    (neural-control) =>
                (tutor-neural control))
```

```
 4. (S_ruleD4   (redefine-dr first-procedure) =>
                (tutor-remind))

 5. (S_ruleR1   (neural-rr) => (tutor-neural-rr))

 6. (S_ruleR3   (causal-relation-one) =>
                (tutor-causality-one))

 7. (S_ruleR4   (rr-reflex) => (tutor-effect-reflex))

 8. (S_ruleS1   (logic-relation) =>
                (tutor-logic-relation))

 9. (S_ruleS2   (ss-reflex) => (tutor-ss-reflex))

10. (S_ruleS3   (neural-ss) => (tutor-neural-ss))

11. (S_ruleS4   (give-ss-map) => (define-ssmap))

12. (S_ruleS5   (give-ss-neural) => (define-ss-neural))
```

2.3 Tactical Rules in Lisp Code

```
 1. (T_ruleD1   (remind-relation) => (remind-relation))

 2. (T_ruleD2   (tutor-remind)  => (remind-dr))

 3. (T_ruleD3   (tutor-neural-control)=>
                (mechanism)(value))

 4. (T_ruleD6   (tutor-causality) =>  (determinants)
                (actual-determinant) (relation) (value))

 5. (T_ruleR1   (tutor-effect-reflex) =>
                (baroreceptor-reflex)(value))

 6. (T_ruleR2   (tutor-neural-rr) => (reflex)(value))

 7. (T_ruleR3   (tutor-causality-one) =>
                (determinants)(relation)(value))

 8. (T_ruleS1   (tutor-logic-relation) =>
                (remind-nv)(follow)(value))

 9. (T_ruleS2   (tutor-neural-ss) =>
                (value-dr)(value-rr)(value-ss))

10. (T_ruleS3   (tutor-ss-reflex) =>
                (reflex-change)(value))
```

DISCOURSE PLANNING RULES

1.1 Pedagogic Rules in English

Pedagogic Default Rules

1. If the current state is introduce
   Then select tactical state or move to tutor.

2. If the current state is tutor
   Then check the topic is completed and move to either
   introduce or complete state

3. If the current state is complete
   Then check the subgoal stack for the next topic
   and move to introduce state

Pedagogic Meta Rules

1. If the prior state is tutor and topic is
   not completed
   Then move to introduce state

2. If the prior state is either introduce or complete
   state and there is no more topic in the stack
   Then exit from the discourse planning

1.2 Tactical Rules in English

Tactical Default Rules

1. If the current state is remind-relation
   Then discourse strategy is remind,
   content is current task, update student model
   and move to compete-topic state

2. If the current state is explain
   Then discourse strategy is explanation,
   content is current task, update student model
   and move to complete-topic state

3. If the current state is ask-question
   Then discourse strategy is question
   content is current task,
   and move to eval-input state

4. If the current state is requestion
   Then discourse strategy is requestion
   content is current task,
   and move to eval-input state

5. If the current state is give-answer
   Then discourse strategy is give answer
   content is (current task and correct answer),
   and update student model, move to complete-topic

6. If the current state is correct-ack
   Then discourse strategy is positive-ack
   content is (current task and student answer),
   update student model, move to complete-topic

7. If the current state is incorrect-ack
   Then discourse strategy is negative-ack
   content is (current task and student answer),
   update student model.

8. If the current state is incorrect-ack-one
   Then discourse strategy is negative-ack-one
   content is (current task and correct student answer,
   incorrect student answer), update student model.

9. If the current state is incorrect-ack-one
   Then discourse strategy is negative-ack-one
   content is (current task and incorrect student
   answers), update student model, move to give-answer

10. If the current state is give-hint
    Then discourse strategy is hint
    content is current task, update student model.

11. If the current state is complete-topic
    Then discourse strategy is complete-topic
    content is current task, update student model,
    update topic-completed.

<u>Tactical</u> <u>Meta</u> <u>Rules</u>

1. If the prior state is eval-input, and
   student response is correct
   Then move to correct-ack state

2. If the prior state is eval-input, and
   student response is incorrect
   Then move to incorrect-ack state

3. If the prior state is eval-input, and
   student response is half correct and first try,
   Then move to incorrect-one-ack state

4. If the prior state is eval-input, and student response is both wrong, and first try Then move to incorrect-both-ack state

5. If the prior state is incorrect-ack and topic is neural control, Then move to give-answer state

6. If the prior state is (incorrect-ack, incorrect-one-ack), topic is causal-relation, and first try, Then move to give-hint state

7. If the prior state is incorrect-ack, topic is causal-relation, and second try, Then move to give-answer state

2.1 Pedagogic Rules in Lisp Code

Pedagogic Default Rules

```
1. (Pedagogic_default     *introduce*
      (subgoal            *current_task*
       update             *topic-completed*
       next_state         *tutor*))

2. (Pedagogic_default     *tutor*
      (subgoal            *current_task*
       update             *topic-completed*
       next_state         *complete*))


3. (Pedagogic_default     *complete*
      (subgoal            *current_task*
       update             *one_topic*
       next_state         *introduce*))
```

Pedagogic Meta Rules

```
1. (Pedagogic_meta        *m_tutor*
      (prior-state        *tutor*
       precondition       *topic-completed*
       next_state         *introduce*))

2. (Pedagogic_meta        *m_complete*
      (prior-state        (*introduce* *complete*)
       precondition       *no_more_topic*
       next_state         *stop*))
```

2.2 Tactical Rules in Lisp Code

Tactical Default Rules

```
1. (Tactical_default      *remind-relation*
     (text-style          remind
      content             *current_task*
      update              *sm*
      next_state          *complete_topic*))

2. (Tactical_default      *explain*
     (text-style          explanation
      content             *current_task*
      update              *sm*
      next_state          *complete_topic*))

3. (Tactical_default      *ask_question*
     (text-style          question
      content             *current_task*
      update              *sm*
      next_state          *eval-input*))

4. (Tactical_default      *requestion*
     (text-style          requestion
      content             *current_task*
      update              *sm*
      next_state          *eval-input*))

5. (Tactical_default      *eval-input*
     (text-style          nil
      content             nil
      update              *sm*
      next_state          nil))

6. (Tactical_default      *give-answer*
     (text-style          give-answer
      content             (*current_task* *correct_ans*)
      update              *sm*
      next_state          *complete_topic*))

7. (Tactical_default      *correct-ack*
     (text-style          positive-ack
      content             (*current_task* *student_ans*)
      update              (*sm*, *topic-completed*)
      next_state          *complete_topic*))

8. (Tactical_default      *incorrect-ack*
     (text-style          negative-ack
      content             (*current_task* *student-ans*)
      update              *sm*
      next_state          nil))
```

```
 9. (Tactical_default      *incorrect-ack-one*
      (text-style          negative-ack-one
       content             (*current_task*
                           *correct-one* *wrong-one*)
       update              *sm*
       next_state          nil))

10. (Tactical_default      *give-hint*
      (text-style          hint
       content             *current_task*
       update              *sm*
       next_state          nil))

11. (Tactical_default      *complete-topic*
      (text-style          complete-topic
       content             *current_task*
       update              (*sm* *topic-completed*)
       next_state          nil))
```

<u>Tactical</u> <u>Meta</u> <u>Rules</u>

```
 1. (Tactical_meta         *m_correct*
      (precondition        response-is-correct
       prior-state         *eval-input*
       next-state          *correct-ack*))

 2. (Tactical_meta         *m_incorrect*
      (precondition        response_is_incorrect
       prior-state         *eval-input*
       next-state          *incorrect-ack*))

 3. (Tactical_meta         *m_incorrect_one*
      (precondition        response_is_incorrect
       prior-state         *eval-input*
       next-state          *incorrect-ack-one*))

 4. (Tactical_meta         *m_incorrect_both*
      (precondition        response_is_incorrect
       prior-state         *eval-input*
       next-state          *incorrect-both-ack*))

 5. (Tactical_meta         *m_first*
      (precondition        (causal first-try)
       prior-state         (*incorrect-ack*
                            *incorrect-ack-one*)
       next-state          *give-hint*))

 6. (Tactical_meta         *m_second*
      (precondition        (causal second-try)
       prior-state         *incorrect-ack*
       next-state          *give-answer*))
```

APPENDIX B

TRACE OF A TUTORING SESSION

This section displays a short example of the system in operation that describes what each component of the system does, what kind of information it needs, and what is the result after each step during the tutoring session. This tutorial interaction begins after the lesson planning is done, so that there are already lesson goals in the goalstack. The discourse planner begins with the first topic in the stack and when that topic is completed, continues with the next topic. Let us assume that the current goalstack contains the lesson goal, "CAUSAL-RELATION (RAP,SV)."

The lesson planner picks the goal and expand it into a set of subgoals: "determinants, actual-determinant, relation, value." Then the discourse planner picks the first subgoal, "determinants," and the tutoring session begins as follows.

Planner: Picks first discourse plan, (ask: determinants)
text-style = question, topic = (determinants SV)
Calls *text-gen*: <u>(question (determinants SV))</u>

Text-Gen: Generates a natural language sentence,
"*What are the determinants of Stroke Volume?*"
and returns it to the Planner.

Planner : Calls Screen Manager to display the sentence.

Screen Manager: Displays the sentence in the TUTOR window.

Planner: Passes the current topic to the Input-Understander,
"<u>(question (determinants SV)</u>"

STUDENT: "HR, RAP"

Screen Manager: Passes the student's answer to Input Understander.

Input-Understander:
Evaluates the student's answer, (HR, RAP).
If the answer is inconsistent with the question, then it replies to the student "Please rephrase," otherwise return the answer to the planner in logic form, "<u>(answer (determinants SV (HR, RAP)))</u>"

Planner: Passes the current topic with student's answer to the Student Modeller,
"<u>((determinants SV) (HR, RAP))</u>"

Student Modeller:
Calls problem_solver, get correct_ans, (RAP,CC), compares (correct_ans with student_ans), updates student model.

Planner: Checks student model, picks the discourse plan,
"(give: Positive-ack, Negative-ack)"
text-style = Positive-ack,
topic = (determinants SV (RAP))
text-style = Negative-ack,
topic = (determinants SV (HR))
Calls Text_Gen:
"<u>((Positive-ack (determinants SV (RAP))</u>,
<u>Negative-ack (determinants SV (HR)))</u>"

Text-Gen: Generates a sentence,
"*Right Atrial Pressure is the correct answer. Heart Rate is not the correct answer.*"

Planner : Calls Screen Manager to display the sentence.
Screen Manager: Displays the sentence in the TUTOR window.

Planner: Picks the next discourse plan, "(give: hint)"
text-style = hint, topic = (definition SV)
Calls the Text_Gen: "<u>(hint (definition SV))</u>"

Text-Gen: Generates a sentence, "*Remember. Stroke volume is the amount of blood pumped per beat.*"
and return it to the planner.

Planner : Calls Screen Manager to display the sentence.

Screen Manager: Displays the sentence in the TUTOR window.

Planner: Picks the next discourse plan, "(ask: determinant)"
text_style = requestion, topic = (determinant SV)
Calls the Text-Gen:
"<u>(requestion (determinants SV))</u>"

Text-Gen: Generates a sentence,
"*What is the other determinant of Stroke Volume?*"

Screen Manager: Displays the sentence in the TUTOR window.

Planner: passes the current topic to the Input-Understander,
"<u>(requestion (determinant SV)</u>"

STUDENT: "Cardiac Output"

Screen Manager: Passes the student's answer to Input Understander.

Input-Understander:
Evaluates the student's answer, (CO).
return the answer to the planner in logic form,
"<u>(answer (determinant SV (CO))</u>"

Planner: Passes the current topic with student's answer to the Student Modeller: "((determinant SV) (CO))"

Student Modeller:
Calls problem_solver, get correct_ans:
compares (correct_ans with student_ans),
updates student model.

Planner: Checks student model,
picks the next discourse plan,
"(give: Negative-ack)"
text-style = Negative-ack,
topic = (determinant SV (CO)),
calls Text_Gen:
"(Negative-ack (determinant SV (CO)))"

Text-Gen: Generates a sentence,
"*Cardiac Output is the wrong answer.*"

Screen Manager: Displays the sentence in the TUTOR window.

Planner: Picks the next discourse plan, "(give: answer)"
text-style = answer, topic = (determinants SV)
Call Text_Gen: "(answer (determinants SV))"

Text-Gen: Generates a sentence,
"*Stroke Volume is determined by Right Atrial Pressure and Cardiac Contractility.*"

Screen Manager: Displays the sentence in the TUTOR window.

* The next step for the discourse planner is to pick the next subgoal, "actual-determinant", and process as above. A real dialogue produced by the system can be found in the Appendix C.

APPENDIX C

DIALOGUE PRODUCED BY THE SYSTEM

This section shows an extended sample of real dialogue produced by CIRCSIM-TUTOR. Two examples will be displayed for each stage (DR, RR, and SS).

Example 1 for DR: The lesson goal of the following dialogue is to tutor the student about the "*causal relationship between Right Atrial Pressure and Stroke Volume,*" which is the same example as in Appendix B.

TUTOR> What are the determinants of Stroke Volume?

STUDENT> RAP and HR

TUTOR> Right atrial pressure is the correct answer. Heart rate is not the correct answer.
Remember. Stroke Volume is the amount of blood pumped per beat.
What is the other determinant of Stroke Volume?

STUDENT> Cardiac Output

TUTOR> Cardiac Output is the wrong answer.
Stroke Volume is determined by the Right Atrial Pressure and the Cardiac Contractility.
What is the actual determinant in this case?

STUDENT> CC

TUTOR> Wrong, the actual determinant of Stroke Volume in this case is not the Cardiac Contractility.
The actual determinant of Stroke Volume in this case is Right Atrial Pressure.

What is the relationship between Right Atrial Pressure and Stroke Volume?

STUDENT> negative

TUTOR> Wrong, Stroke Volume is not inversely proportional to Right Atrial Pressure. Stroke Volume is directly proportional to Right Atrial Pressure.
What is the correct value of Stroke Volume?

STUDENT> increase

TUTOR> Wrong, the value of Stroke Volume is not increased. Consider the value of Right Atrial Pressure.
What is the correct value of Stroke Volume?

STUDENT> decreased

TUTOR> Correct, the value of Stroke Volume is decreased.

Example 2 for DR: The lesson goal of the following dialogue is to tutor the student about the "*neural control variable*" in the DR stage.

TUTOR> Remember the direct response occurs immediately and produces physical changes in the system before any reflex is activated.
By what mechanism is Heart Rate controlled?

STUDENT> neural

TUTOR> Correct, Heart Rate is controlled by the nervous system. What is the correct value of Heart Rate?

STUDENT> decreased

TUTOR> Wrong, the value of Heart Rate is not decreased.
Remember. Heart Rate is neurally controlled.
What is the correct value of Heart Rate?

STUDENT> nochange.

TUTOR> Correct, the value of Heart Rate is unchanged.

<u>Example 1 for RR</u>: The lesson goal of the following dialogue is to tutor the student about the "*regulated variable (Mean Arterial Pressure)*" in the RR stage.

TUTOR> Remember the reflex corrects the regulated variable. What variable is regulated by the baroreceptor reflex?

STUDENT> Heart Rate

TUTOR> Wrong, Heart Rate is not the regulated variable.
How is the baroreceptor measured?
Again, what variable is regulated by the baroreceptor reflex?

STUDENT> MAP

TUTOR> Correct, the baroreceptor reflex regulates Mean Arterial Pressure.
What is the correct value of Mean Arterial Pressure?

STUDENT> decreased

TUTOR> Wrong, the value of mean arterial pressure is not decreased.

	Consider the value of Mean Arterial Pressure in DR. What is the correct value of Mean Arterial Pressure?
STUDENT>	increased
TUTOR>	Correct, the value of Mean Arterial Pressure is increased.

<u>Example 2 for RR</u>: The lesson goal of the following dialogue is to tutor the student about the "*neural control variable*" in the RR stage.

TUTOR>	Remember that the reflex changes the value of all neurally controlled variables first, in order to correct the change in arterial pressure. Which variables are changed by the reflex?
STUDENT>	CC, HR, and TPR
TUTOR>	Correct, Cardiac Contractility, Heart Rate, and Total Peripheral Resistance are changed by the reflex. What is the correct value of Heart Rate?
STUDENT>	decreased
TUTOR>	Wrong, the value of Heart Rate is not decreased. The value of Heart Rate is increased.

<u>Example 1 for SS</u>: The lesson goal of the following dialogue is to tutor the student about the "*logic relation*" in the SS stage.

TUTOR> The value of a neural variable is SS is the same as its value in RR, because neural variables do not change in DR except when one is a primary variable. What is the correct value of Heart Rate in DR?

STUDENT> nochange

TUTOR> Correct, the value of Heart Rate is unchanged in DR. What is the correct value of Heart Rate in RR?

STUDENT> increased

TUTOR> Correct, the value of Heart Rate is increased in RR. What is the correct value of Heart Rate in SS?

STUDENT> increased

TUTOR> Correct, the value of Heart Rate is increased in SS.

<u>Example 2 for SS</u>: The lesson goal of the following dialogue is to tutor the student about the "*physical variable*" in the SS stage.

TUTOR> Remember cardiac output decreased in DR, it increased in RR.
What must the value of Cardiac Output follow in SS?

STUDENT> rr

TUTOR> Wrong, the value of Cardiac Output doesn't follow the value of Cardiac Output in RR. The value of Cardiac Output must follow the value of Cardiac Output in DR.

What is the correct value of Cardiac Output?

STUDENT> decreased

TUTOR> Correct, the value of Cardiac Output is decreased.

BIBLIOGRAPHY

Anderson, J.R. 1988. The Expert Module. In: Polson, M. C. and Richardson, J.J., Eds., *Foundations of Intelligent Tutoring Systems,* Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 21–53.

Anderson, J.R. and Reiser, B.J. 1985. The LISP Tutor. *BYTE*, 10(4):159–175.

Anderson, J.R., Boyle, F., Albert, T.C., and Lewis, M.W. 1990. Cognitive Modeling and Intelligent Tutoring. *Artificial Intelligence*, 42:7–49.

Barr, A. and Feigenbaum, E.A. 1982. *The Handbook of Artificial Intelligence*. William Kaufmann, Inc., Los Altos, California, 2:225–294.

Brecht, B., McCalla, G., Greer, J., and Jones, M. 1989. Planning the Content of Instruction. *Proceedings of the 4th International Conference on AI and Education,* Amsterdam, Netherlands, 32–41.

Brown, J.S. and Burton, R.R. 1978. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 2:155–191.

Brown, J.S., Burton, R.R., and deKleer, J. 1982. Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*, Academic Press, New York, 227–282.

Burns, B., Parlett, J.W. and Redfield, C.L. 1991. *Intelligent Tutoring Systems: Evolutions in Design*, Lawrence Erlbaum Publishers, Hillsdale, New Jersey.

Burton, R.R. and Brown, J.S. 1982. An Investigation of Computer Coaching for Information Learning Activities. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*, Academic Press, New York, 79–98.

Carbonell, J.R. 1970. AI in CAI: Artificial Intelligence Approach to Computer Assisted Instruction. *IEEE Transactions on Man-Machine Systems*, 11:190–202.

Carr, B. and Goldstein, I.P. 1977. *Overlays: a Theory of Modelling for Computer Aided Instruction*. AI Memo 406, AI Laboratory, Massachusetts Institute of Technology.

Charniak, E. and McDermott, D. 1986. *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts.

Clancey, W.J. 1982. Tutoring Rules for Guiding a Case Methods Dialogue. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*, Academic Press, New York, 201-225.

Clancey, W.J. 1987. *Intelligent Tutoring Systems: A Tutorial Survey*. Report No. STAN-CS-87-1174.

Clancey, W.J., Shortliffe, E. and Buchanan, B. 1984. Intelligent Computer-Aided Instruction for Medical Diagnosis. In: Clancey, W.J. and Shortliffe, E., Eds., *Readings in Medical Artificial Intelligence: The First Decade*. Addison-Wesley, Reading, Massachusetts, 256-274.

Cohen, P.R. and Feigenbaum, E.D. 1982a. Planning and Problem-solving. *The Handbook of Artificial Intelligence*. William Kaufmann, Inc., Los Altos, California, 3:515-562.

Cohen, P.R. and Feigenbaum, E.D. 1982b. Learning and Inductive Inference. *The Handbook of Artificial Intelligence*. William Kaufmann, Inc., Los Altos, California, 3:325-511.

Cohn, G.A. 1987. Qualitative Reasoning. In: Nossum, R.T., Ed., *Advanced Topics in Artificial Intelligence*, Springer-Verlag, Reading, Massachusetts, 345:60-95.

Davis, R. and Buchanan, B. G. 1985. Meta-Level Knowledge. In: Buchanan, B. G. and Shortliffe, E. H., Eds., *Rule-Based Expert Systems*. Addison-Wesley, Reading, Massachusetts, 507-530.

de Castro, M.I., Sanchez, A. and Verdejo Maillo, M.F. 1988. Building a Programming Tutor by Dynamic Planning: Case Studies and a Proposal. *Proceedings of Intelligent Tutoring Systems: ITS-88*, Montreal, Quebec, Canada, 230-237.

deKleer, J. and Brown, J.S. 1984. A Physics Based on Confluences. *Artificial Intelligence*, 24:7-83.

Dede, C.J. 1986. A Review and Synthesis of Recent Research in Intelligent Computer-Assisted Instruction. *International Journal of Man-Machine Studies*, 24:329-353.

Derry, S.J., Hawkes, L.W. and Ziegler,U. 1988. A Plan-Based Opportunistic Architecture for Intelligent Tutoring. *Proceedings of Intelligent Tutoring Systems. ITS-88,* Montreal, Quebec, Canada, 116-123.

Durfee, E.H. and Lesser, V.R. 1986. Incremental Planning to Control a Blackboard-Based Problem-Solver. *Proceedings* *of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania, 58–64.

Elsom-Cook, M. 1988. Using Multiple Teaching Strategies in an ITS. *Proceedings of Intelligent Tutoring Systems: ITS-88*, Montreal, Quebec, Canada, 286–290.

Farr, M.J. and Psotka, J. 1989. Introduction. *Machine-Mediated Learning*, 3:1–6.

Fikes, R. and Nilsson, N.J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208.

Forbus, K.D. 1984. Qualitative Process Theory. *Artificial Intelligence*, 24:85–168.

Galdes D.K., Smith, P.J. and Smith J. W. 1991. Factors Determining When to Interrupt and What to Say: An Empirical Study of the Case-Method Tutoring Approach. *Proceedings of the International Conference on the Learning Sciences*. Evanston, IL, 195–202.

Genesereth, M.R. 1982. The Role of Plans in Intelligent Teaching Systems. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*, Academic Press, New York, 137–155.

Goldstein, I.P. 1982. The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*, Academic Press, New York, 51–77.

Goodyear, P. 1988. Approaches to the Empirical Derivation of Teaching Knowledge for Intelligent Tutoring Systems. *Proceedings of Intelligent Tutoring Systems: ITS-88*, Montreal, Quebec, Canada, 291–298.

Halff, H.M. 1988. Curriculum and Instruction in Automated Tutors. In: Polson, M.C. and Richardson, J.J., Eds., *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 79–108.

Harmon, P. 1987. Intelligent Job Aids: How AI will Change Training in the Next Five Years. In: Kearsley, G.P., Ed., *Artificial Intelligence and Instruction*, Addison-Wesley, Reading, Massachusetts, 165–190.

Hammond, K.J. 1989. *Case-Based Planning*. Academic Press, New York.

Hasemer, T. and Dominque, J. 1989. *Common Lisp Programming for Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts.

Hayes-Roth, B., 1985. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:251-321.

Hendler, J., Tate, A., and Drummond, M. 1990. AI Planning: Systems and Techniques. *AI Magazine*, Summer, 61-77.

Heines, J.M. 1985. The Design of a Rule-based CAI Tutorial. *International Journal of Man-Machine Studies*, 23:1-25.

Kearsley, G. 1987. *Artificial Intelligence and Instruction: Application and Methods*, Addison-Wesley, Reading, Massachusetts.

Kim, N., Evens, M., Michael, J.A., and Rovick, A.A. 1989. CIRCSIM-TUTOR: An Intelligent Tutoring System for Circulatory Physiology. In: Mauer, H., Ed., Computer Assisted Learning, *Proceedings of the International Conference on Computer Assisted Learning*, Dallas, Texas, Springer-Verlag, Berlin:254-266.

Kim, N. 1989. *CIRCSIM-TUTOR: An Intelligent Tutoring System for Circulatory Physiology*. PhD. Dissertation, Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois.

Kuipers, B. 1984. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence*,_24:169-203.

Lajoie, S.P. and Lesgold, A. 1989. Apprenticeship Training in the Workplace: Computer-Coached Practice Environment as a New Form of Apprenticeship. *Machine-Mediated Learning*, 3:7-28.

Lee, Y., Evens, M., Michael, J.A., and Rovick, A.A. 1990. IFIHS: Ill-Formed Input Handling System. *Proceedings of the Second Midwest Artificial Intelligence and Cognitive Science Conference*. Carbondale, IL, 93-97.

Lee, Y. 1990. *Handling Ill-Formed Natural Language Input for an Intelligent Tutoring System*. Ph.D. Dissertation, Computer Science Department, Illinois Institute of Technology, Chicago, Illinois.

Leinhardt, G. and Greeno, J.G. 1986. The Cognitive Skill of Teaching. *Journal of Educational Psychology*, 78:75-95.

Lesgold, A.M. 1987. Toward a Theory of Curriculum for Use in Designing Intelligent Instructional Systems. In: Mandl, H. and Lesgold, A. M., Eds., *Learning Issues for Intelligent Tutoring Systems*, Springer-Verlag, New York, 114-137.

Macmillan, S.A. and Sleeman, D.H. 1987. An Architecture for a Self-Improving Instructional Planner for Intelligent Tutoring Systems. *Computational Intelligence*, 3:17-27.

Macmillan, S.A., Emme, D. and Berkowitz, M. 1988. Instructional Planners:Lessons Learned, In: Psotka, J., Massey, L.D., and Mutter, S.A., Eds., *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Publishers, Hillsdale, New Jersey, 229-256.

McCalla, G.I. and Greer, J.E. 1988. Intelligent Advising in Problem Solving Domains: The SCENT-3 Architecture. *Proceedings of Intelligent Tutoring Systems*: *ITS-88,* Montreal, Quebec, Canada, 124-131.

Michael, J. A., Rovick, A. A., Evens, M., and Kim, N. 1990. A Smart Tutor Based on a Qualitative Causal Model. *Proceedings of the AAAI Spring Symposium on Knowledge-Based Environments for Learning and Teaching*, Stanford, March 27-29, 112-117.

Michael, J.A., Rovick, A.A., Evens, M., Shim, L., Woo, C., and Kim, N. 1991. The Uses of Multiple Student Inputs in Modeling and Lesson Planning in CAI and ICAI Programs. *Submitted to the International Conference on Computer Assisted Learning*, Nova Scotia, CANADA.

Murray, W.R. 1988. *Control for Intelligent Tutoring Systems: A Comparison of Blackboard Architecture and Discourse Management Networks*. Research Report R-6267, FMC Corporation, Santa Clara, California.

Murray, W.R. 1990. *A Blackboard-based Dynamic Instructional Planner*. Research Report No. R-6376, FMC Corporation, Santa Clara, California.

Ohlsson, S. 1987. Some Principles of Intelligent tutoring. In: Lawler, R. and Yazdani, M., Eds., *Artificial Intelligence and Education: Learning Environment and Tutoring Systems*, Ablex Publishing, Norwood, New Jersey, 204-237.

Park, O.C., Perez, R.S. and Seidel,J. 1987. Intelligent CAI: Old Wine in New Bottles, or a New Vintage? In: Kearsley, G.P., Ed., *Artificial Intelligence and Instruction*, Addison-Wesley, Reading, Massachusetts, 11-45.

Patil, R.S., Szolovits, P., and Schwarts, W.B. 1984. Causal Understanding of Patient Illness in Medical Diagnosis. In: Clancey, W.J. and Shortliffe, E., Eds., *Readings in Medical Artificial Intelligence: The First Decade*. Addison-Wesley, Reading, Massachusetts, 256-274.

Pazzani, M. 1991. A Computational Theory of Learning Causal Relationships. *Cognitive Science*, 15:401-424.

Peachey, D.R. and McCalla, G.I. 1986. Using Planning Techniques in Intelligent Tutoring Systems. *International Journal of Man-Machine Studies*, 24:77-88.

Rovick, A.A. and Michael, J.A. 1986. CIRCSIM: An IBM PC Computer Teaching Exercise on Blood Pressure Regulation. *XXX IUPS Congress*, Vancouver, Canada.

Rovick, A. A. and Brenner, L. 1983. HEARTSIM: A Cardiovascular Simulation with Didactic Feedback. *The Physiologist*, 26:236-239.

Russell, D.M. 1988. IDE: The Interpreter. In: Psotka, J., Massey, L.D., and Mutter, S.A., Eds., *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Publishers, Hillsdale, New Jersey, 323-349.

Russell, D.M., Moran, T.P., and Jordan, D.S. 1988. The Instructional-Design Environment. In: Psotka, J., Massey, L. D., and Mutter, S.A., Eds., *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Publishers, Hillsdale, New Jersey, 203-228.

Sacerdoti, E.D., 1974. Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence*, 5:115-135.

Sacerdoti, E.D., 1977. *A Structure for Plans and Behavior*. Elsevier-North Holland, Amsterdam, the Netherlands.

Shim, L., Evens, M., Rovick, A.A., and Michael, J.A. 1991. Effective Cognitive Modeling in an Intelligent Tutoring System for Cardiovascular Physiology. *Proceedings of Fourth IEEE Symposium on Computer-Based Medical Systems.* Baltimore, Maryland, 338-345.

Shim, L. 1991. *Student Modeling for an Intelligent Tutoring System: Based on the Analysis of Human Tutoring Sessions*._PhD. Dissertation, Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois.

Sleeman, D.H. and Brown, J.S. 1982. *Intelligent Tutoring Systems*. Academic Press, New York.

Stefik, M.J., 1981. Planning and Meta-Planning. *Artificial Intelligence*, 16:141-169.

Stevens, A., Collins, A. and Goldin, S.E. 1982. Misconceptions in Student's Understanding. In: Sleeman, D.H. and Brown, J.S., Eds., *Intelligent Tutoring Systems*. Academic Press, New York, 13-24.

Sussman, G.A. 1975. *A Computational Model of Skill Acquisition*, Elsevier-North Holland, New York.

Swartout, W. 1988. DARPA Santa Cruz Workshop on Planning. *AI Magazine*, Summer, 115-131.

Tate, A. 1977. Interacting Goals and Their Use. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 215-218.

VanLehn, K. 1988. Student Modelling. In: Polson, M. C. and Richardson, J.J., Eds., *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey: 55-78.

Wenger, E. 1987. *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann, Los Altos, California.

Wielinga, B.J. and Breuker, J.A. 1990. Models of Expertise. *International Journal of Intelligent Systems*, 5(5):497-509.

Wilensky, R. 1983. *Planning and Understanding*. Addison-Wesley, Reading, Massachusetts.

Wilensky, R., Chin, D.N., Luria, M., Martin, J., Mayfield, J. and Wu, D. 1988. The Berkeley UNIX Consultant Project. *Computational Linguistics*, 14:35-84.

Winkels, R., Breuker, J. and Sandberg, J. 1988. Didactic Discourse in Intelligent Help Systems. *Proceedings of Intelligent Tutoring Systems: ITS-88*, Montreal, Quebec, Canada, 279-285.

Wilkins, D. E. 1988. *Practical Planning*. Morgan Kaufmann, Los Altos, California.

Woo, C., Evens, M., Michael, J.A. and Rovick, A.A. 1991a. Dynamic Instructional Planning for an Intelligent Physiology Tutoring System. *Proceedings of Fourth IEEE Symposium on Computer-Based Medical Systems*, Baltimore, MD:226-233.

Woo, C., Evens, M., Michael, J.A., and Rovick, A.A. 1991b. Instructional Planning for an Intelligent Medical Tutoring System. *Proceedings of the Third Midwest Artificial Intelligence and Cognitive Science Conference*. Carbondale, IL, 31–35.

Woo, C., Evens, M., Michael, J.A. and Rovick, A.A. 1991c. Planning in an Intelligent Tutoring System. *Accepted as Poster Session to the International Conference on the Learning Sciences*, Evanston, Illinois, U.S.A.

Woolf, B. and McDonald, D. 1984. Context-Dependent Transition in Tutoring Discourse. *Proceedings of AAAI*, 84:353–361.

Woolf, B. 1984. *Context Dependent Planning in a Machine Tutor*. Ph.D. Dissertation, University of Massachusetts, Amherst, Massachusetts.

Woolf, B. Theoretical Frontiers in Building a Machine Tutor. 1987. In: Kearsley, G.P., Ed., *Artificial Intelligence and Instruction*, Addison-Wesley, Reading, Massachusetts:229–267.

Zhang, Y. and Evens, M., 1990. Extending a Knowledge Base to Support Explanations. *Proceedings of Third Annual IEEE Symposium on Computer-Based Medical Systems*, 259–266.