A student performance evaluation method for an intelligent simulation-based tutoring system

1988

Camille Marie Dixon
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Industrial Engineering Commons

STARS Citation

https://stars.library.ucf.edu/rtd/4274

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
A STUDENT PERFORMANCE EVALUATION METHOD
FOR AN INTELLIGENT SIMULATION-BASED TUTORING SYSTEM

BY

CAMILLE M. DIXON
B.S.E., University of Central Florida, 1987

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1988
ABSTRACT

Intelligent simulation-based tutoring systems (ISTS) present intriguing and complex environments for the training of high performance skills. These skills involve the manipulation of objects within time and space constraints. As with any tutoring system, there must exist a performance measurement methodology. Within an ISTS there is a module which functions as a tutor and a module which functions as a student modeler. These fundamental modules are required to make effective individualized tutoring decisions. However, each of these modules rely on information from a performance measurement system or evaluation system.

The objective of this research was to establish a performance evaluation scheme for an ISTS. The evaluation method described in this document presents a performance based assessment of the student’s actions. The intent of the evaluation scheme is to provide information to the tutor and to the student model which permits inferences about the student’s capacity to learn, learning habits and level of expertise.
ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Dr. John E. Biegel for allowing me the opportunity to be a part of the ISTS project. The ISTS’s environment is one which stimulates emotional as well as academic growth. Dr. Biegel provided me with valuable experience in the innovative and exciting fields of artificial intelligence and expert systems.

I would like to thank all of the people in the Industrial Engineering Department of the University of Central Florida for the many years of companionship and inspiration. A special thanks to all of my ISTS team members for their encouragement, especially Gaju Nadoli whose many hours of tutoring and support are deeply appreciated.

Thanks to my parents, Robert and Frances Dixon, for their constant delight in my efforts and accomplishments. Finally, thanks to my friend, Stan Blazewicz, for providing me with the incentive to complete my degree earlier than expected and for adding a new excitement to my life.
# TABLE OF CONTENTS

**LIST OF TABLES** .......................................................... vi
**LIST OF FIGURES** .......................................................... vii
**LIST OF ABBREVIATIONS** ................................................ viii

**Chapter**

1. **INTRODUCTION** ......................................................... 1
   - Intelligent Tutoring Systems ........................................... 1
   - Objectives of the Research ............................................ 3
   - Thesis Evolution ...................................................... 4
   - Organization of the Thesis .......................................... 6

2. **BACKGROUND** ............................................................ 7
   - History and Requirements of ITS ..................................... 7
   - Student Models and Evaluation Methods ............................ 9
     - BUGGY ................................................................. 9
     - LMS ................................................................. 11
     - GUIDON ............................................................. 12
     - WUSOR ............................................................... 14
     - WEST ................................................................. 18
   - Simulation-Based Training Systems .................................. 20
     - SOPHIE ............................................................. 20
     - STEAMER ........................................................... 22

3. **PROBLEM STATEMENT** .................................................. 25
   - Human Performance in Simulation Systems .......................... 25
   - ISTS System Description ............................................. 27
   - Student Interaction with a Functioning ISTS ..................... 33
   - ISTS Evaluator ...................................................... 35

4. **METHOD** ................................................................. 39
   - Hardware and Software Systems Used ............................... 40
   - Solution Description ................................................ 40
   - Schema Creation ..................................................... 46
   - Schema Slot Values and Their Use .................................. 51
   - Schema Retraction at the End of the Lesson ...................... 59
   - Scoring Information Sent to the Student Model .................. 59

5. **RESULTS** ............................................................... 62

6. **CONCLUSIONS** .......................................................... 64

7. **FURTHER RESEARCH** ................................................. 67
LIST OF TABLES

1. Interpreter Snapshot Fact Tags . . . . . . . . . . . . . . . . 45
2. Initialization Facts for Scoring Students . . . . . . 58
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ISTS System Modules</td>
<td>28</td>
</tr>
<tr>
<td>2.</td>
<td>Evaluator - System Interaction</td>
<td>36</td>
</tr>
<tr>
<td>3.</td>
<td>Schema Types and Development</td>
<td>42</td>
</tr>
<tr>
<td>4.</td>
<td>Facts Sent to the Evaluator</td>
<td>44</td>
</tr>
<tr>
<td>5.</td>
<td>Part of Evaluator Decision Flow Chart</td>
<td>47</td>
</tr>
<tr>
<td>6.</td>
<td>Example of Schema Slot Inheritance</td>
<td>48</td>
</tr>
<tr>
<td>7.</td>
<td>Record-keeper Schema</td>
<td>50</td>
</tr>
</tbody>
</table>
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ART</td>
<td>Automated Reasoning Tool</td>
</tr>
<tr>
<td>ATC</td>
<td>Air Traffic Control</td>
</tr>
<tr>
<td>CAI</td>
<td>Computer Aided Instruction</td>
</tr>
<tr>
<td>CBI</td>
<td>Computer Based Instruction</td>
</tr>
<tr>
<td>ES</td>
<td>Expert Systems</td>
</tr>
<tr>
<td>EVA</td>
<td>Implemented Version of Evaluator Module</td>
</tr>
<tr>
<td>IPP</td>
<td>Intelligent Pre-processor</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Tutoring Systems</td>
</tr>
<tr>
<td>ISTS</td>
<td>Intelligent Simulation-Based Training System</td>
</tr>
<tr>
<td>LMS</td>
<td>Leeds Modelling System</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) are computer systems designed to perform the instructional functions of a human tutor. An ITS is substantiated with a domain knowledge base and a reasoning mechanism allowing it to access and make inferences concerning the domain knowledge. The system also has the capacity to make inferences about the student's current knowledge state. The desire to use computers as tutors that emulate the functional characteristics of a human tutor has challenged the fields of Artificial Intelligence (AI), Expert Systems (ES), Computer Science, Engineering, Education, and Psychology. Many AI and ES techniques have been researched and developed with the objective of addressing the concerns and the problems associated with creating intelligent tutoring environments.

Woolf (1984, 25-28) outlines the component requirements of a tutoring system. She suggests that there are four important elements. These are: (1) expert module, (2) teaching component, (3) student model, and (4) communication module. The expert module is used to store information about the domain. The teaching component
maintains the method and strategies of teaching. The student model contains information about the student's knowledge state. Finally, the communication module provides the capability for the student and the system to interact.

Wenger states that intelligent communication cannot occur if the system or instructor does not have knowledge about the recipient (Wenger 1987, 16). The student model attempts to reveal the student's problem solving ability and reasoning process. The tutor can then develop teaching strategies and remediation suitable for an individual student. Evaluation is usually considered to be a part of the student model. It is the method by which student errors are recognized. It is also the method for determining, to some extent, the student's strategic, problem solving skills.

Wenger points out that the student model is likely to reflect the student's knowledge as a function of deviation from some target expertise (Wenger 1987, 17). Many student modeling techniques have been developed. Some attempt to build a knowledge base of all the possible errors a student can make. These systems are generally referred to as bug models. The bug models perturb the expert correct method with "bugs" until there is a match.

Other systems attempt to have an expert knowledge base made up of procedures or rules allowing it to solve the problem presented to the student. These are often referred
to as overlay models. For each type of model, the student solution is matched against an expert solution. The overlay models compare the expert solution or solutions with the student solution and determine which student solution components are non-optimal or missing.

The decision concerning the development of a method for diagnosing student behavior is affected by many factors. These factors are as follows: (1) the observable student behavior is only representative of the student's reasoning processes; (2) misconceptions can create a variety of responses, even the desired one; and (3) real-time search requirements can be limiting if the diagnoses is to be an on-line function (Wenger 1987, 19).

**Objectives of the Research**

The objective of this research is to establish a performance evaluation scheme for an intelligent simulation-based training system. The evaluation scheme will perform an on-line, real-time evaluation of a trainee. Included is the development of a system that will generate objective measures of performance to be used to determine the student's capability in the domain. This assessment is based on inferences about the student's solution method and assigned credit values.
This thesis is based on research being conducted at the University of Central Florida at Orlando, Florida; General Electric's Simulation and Control Systems Department and Embry-Riddle Aeronautical University, both at Daytona Beach, Florida. These groups are collaborating on the design and development of an Intelligent Simulation-Based Training System (ISTS). The objective of the ISTS project is to develop a mostly generic intelligent simulation training system which will remove the human instructor from the training loop.

Although the research investigates domain independent relationships within the system, it is also recognized that domain dependent knowledge is required. Thus, domain dependent components are also being developed so that the system is made functional for specific domains. The ISTS is being developed on a modular basis and has the purpose of training students through individualized instruction.

Individualizing instruction is a function of the capability to model the student and on the capability to model the tutor so that appropriate teaching decisions are made. The ISTS has the intent of progressing the state of the art of training systems through the use of ES techniques integrated with an interactive, dynamic simulation. The simulation is specific to those training environments in which objects are moved through time and space. The test
domain of the system is Air Traffic Control (ATC) with the initial effort focusing on the training of radar air traffic controllers.

Other research has been accomplished on the idea of domain independence. This is a result of the fact that teaching has certain necessary tasks associated with it. These necessary tasks require the implementation of the fundamental components of a tutoring system. These fundamental components are the expert, the teaching module and the student model.

The ISTS design incorporates all of the modules recommended by Woolf. However, because the ISTS uses an interactive simulation as part of its training mechanism, it has incorporated many more modules. This thesis focuses on the development of an Evaluator module which is separated from the Student Model module. It makes use of the current system design of the ISTS to establish system parameters which may be encountered by or made available to the Evaluator.

Evaluation is usually considered to be a part of the student model. Again, it is the method by which the student errors are recognized and the determination of the student's understanding and reasoning. Within the ISTS, student errors are based on the data input from other system modules. The Evaluator has a close interaction with the Student Model module and is intended to provide sufficient
evidence of the student’s capability so that the Student Model can dynamically update the student’s learning status and knowledge state.

Organization of the Thesis

The remainder of the thesis details the proposed solution to the problem statement. Chapter 2 contains description of the research which has been conducted on student modeling and evaluation. It specifically concentrates on the application and development of student models because much of the research in ITS student evaluation incorporates the evaluation process in the student model. Also included is background on the use of simulation-based tutoring systems. The research discussed attempts to show a variety of methods that have been proposed.

Chapter 3 presents the problem statement. This chapter gives a detailed description of the ISTS and examines the role of the Evaluator. Chapter 4 reveals the methodology used to respond to the problem statement. It describes the software and hardware components of the implemented solution method and details the results of the implementation. Chapter 5 provides the results of the research. Chapter 6 gives the conclusions and summarizes the research effort. Chapter 7 suggests considerations for future research efforts.
CHAPTER 2
BACKGROUND

History and Requirements of ITS

Computer Aided Instruction (CAI) systems, otherwise referred to as Computer Based Instruction (CBI), began as programmed teaching machines (Park, Perez, and Seidel 1987). They enjoyed little acceptance due to criticisms aimed at their inability to emulate the characteristics of a teacher. Cochran (1985) points out several complaints of the early CAI systems. One is that the real power of the computer was not being used and that they were simply electronic workbooks. Another is that the computer had no knowledge about what it was teaching. Subject matter was hard coded into ad-hoc frame oriented systems. Hard coding meant that the subject matter could not be manipulated and used for a variety of related problems. Other complaints stated that the computer did not incorporate a model of the student so that it could not provide for individualized instruction. Also, the computer did not incorporate natural language processing or the capacity to allow the student to initiate dialogue with the computer. These criticisms caused researchers to focus their attention on AI's original idea. That is, the computer must embody some mechanism to make it appear to have human-like intelligence.
The expert-like capability of a computer system is a function of the knowledge it contains and of the efficiency of its inference mechanism. For instructional purposes, there are basically three types of required knowledge. As Park, Perez, and Seidel (1987) state, these are:

1. Expertise knowledge,
2. Student Model, and
3. Tutoring or teaching knowledge.

The expertise knowledge is made up of the facts, rules and heuristics associated with the subject domain. It represents the expert reasoning power and solution generation. The student model incorporates knowledge of the student; this knowledge is developed by diagnosing the student’s understanding of the domain. The tutoring knowledge is constructed of pedagogical methods and has the ability to make inferences about the best way to teach an individual student.

This paper is concerned with student performance evaluation. The procedures which focus on student performance evaluation are often found in literature detailing the student model. There are several significant techniques being used to model students. Essentially these techniques describe the evaluation method. Woolf (1984) states that there are three types of student modeling methods. These are the bug modeler, the overlay modeler, and the skill modeler. For the most part, the skill modeler
is a version of the overlay modeler. Each of these modeling techniques uses a knowledge base with which to compare the student’s response. This comparison procedure is the basis for evaluation of the student’s ability.

The knowledge base of the bug modeler consists of the possible bugs a student may incorporate in the solution process. The student’s method is compared with each of these bugs until there is a match. The knowledge base of the overlay modeler is generally considered to be a subset of the expert knowledge base. The overlay modeler compares the student’s solution to that of a system generated solution, usually an expert solution. The skill modeler groups the expert knowledge into a set of skills. This modeler then is able to use an overlay method to determine if the student uses a particular skill in the solution process (Woolf 1984, 40). Some examples of methods developed to determine the extent of the student’s ability via an evaluation procedure will be detailed in the next section. The examples given show a variety of research approaches.

Student Models and Evaluation Methods

BUGGY

BUGGY is a well known diagnostic tool which exposed the fact that a great majority of student errors do not occur randomly but that they are a product of a systematic misconception. Cochran (1985) details the origination and
the discoveries of the BUGGY program. The developers of BUGGY incorporated the idea of using a "procedural network" to describe the primitive tasks associated with simple arithmetic tasks. Within the network, the procedures required to perform arithmetic tasks such as addition are described as nodes. The nodes have two components. One, the conceptual part, which is the intent, and two, the operational part, which is the method for carrying out the intent. The procedure nodes are linked together and used by one another through a control mechanism. This allows one procedure node to call another if necessary. The breakdown of the procedures into primitives allows the BUGGY program to perform a diagnosis concerning the student's procedural misuse. The program systematically substitutes wrong procedures until it matches the student's method.

BUGGY was also developed to determine the "deep structure model" of student error by means of combining simple bugs and testing the combinations for a match with the student's error. Although this work was promising, there were still some complicating factors such as the fact that sometimes students did make random errors. Also, the procedural error could be combined. This led the researchers to discover the combinatorial explosiveness of a more complex domain.

Research continued on the BUGGY program and other program extensions developed. DEBUGGY and IDEBUGGY
represent two such extensions. While BUGGY and DEBUGGY were "off-line" diagnostic systems, IDEBUGGY's intent was to be an interactive system which would reveal its hypothesis to the student.

An important conclusion of the BUGGY system and its extensions is that in algorithmic, procedure specific domains, the student's skill can be represented as a subset of skills. These subset skills can then cause the deviations from the correct procedure (Cochran 1985, 276-292).

**LMS**

The Leeds Modelling System (LMS) was developed in 1979 with its primary purpose being the development of a student diagnostic model (Cochran 1985). Cochran (1985) overviews the functionality of the LMS. The LMS was established using algebra as its domain. However, the theory was to develop a domain independent diagnostic methodology.

The data base of the LMS is composed of the production rules required to work through the problem correctly, the "mal-rules," which are the possible deviant rules used in problem solving, and a set of tasks to perform. In complex domains, however, the discovery and combination of possible "mal-rules" could be quite large. Thus, another goal of this research was to develop some type of system to control
for the combinatorial problems associated with complex domains.

A number of searching techniques were attempted to deal with the problem. First, all combinations of the mal-rules and production rules were created and the system searched until it found a match. This search was termed the EXHAUSTIVE approach. Second, the EXHAUSTIVE-GROUPED method specified particular mal-rules for particular production rules. Third, the SELECTIVE method limited the search space by assuming certain student knowledge. Once a rule was considered to be mastered by the student, it was assumed that the student would continue to use the rule in a correct manner. Only one rule was added at a time.

Initial testing of the system suggested that this assumption caused other errors. Continued research showed that once this assumption was revised, an enhanced set of mal-rules was required.

GUIDON

GUIDON was developed by Clancey (1987) to be an interactive teaching program for a highly technical domain. Clancey asserts that GUIDON uses the knowledge base of the MYCIN program, an expert system designed to give advice on infectious diseases. The goal of GUIDON was to provide the teaching expertise that was lacking in the MYCIN program and
to make the student aware of inconsistencies or lack of knowledge (Clancey 1987, 3).

The student model used in the GUIDON program is an overlay model in which the student’s knowledge is considered to be a part of the expert’s knowledge. Clancey states that the rule in the expert rule base is a skill or problem solving technique; therefore, the rule can be distinguished as to whether the student knows it, whether the rule can be used for problem solving, and whether the student has actually applied the rule (Clancey 1987, 113).

The student model uses the domain rules, referred to as d-rules. The rules have three properties associated with them. Clancey (1987) refers to these properties as USE-HISTORY, SAPPLIED?, and USED?. The overlay model uses tutorial rules to determine if the student is deducing as the expert is. The assumptions of GUIDON are that the there are unique reasoning paths and that the student’s knowledge set is a subset of the expert’s. The student model is updated on the basis of evidence concerning particular rule usage. There are three types of evidence. These are background, implicit and explicit. The background evidence is used to determine if a student is at a certain sophistication or learning level. That is, the background evidence reveals if the student knows or should know a rule based on the perceived knowledge level. The implicit evidence is gathered from the student’s ability to solve
subgoals, or whether the student asks for help. Explicit evidence is obtained from asking the student specific questions so as to directly reveal his knowledge.

The evaluation of the student’s ability comes from the use of tutorial rules, referred to as t-rules. The t-rules first determine if d-rules used by the expert were used by the student. Second, the t-rules look for patterns in the expert’s behavior and in the student’s behavior to determine if the student is using a similar strategy. The t-rules contain factors for updating the evidence of use and the system’s belief about the student’s knowledge and strategy.

Clancey points out that the assumption of a unique reasoning path used by the system’s overlay procedure may cause the system to conclude student error and misconception even though the student may simply use a different approach (Clancey 1987, 113).

**WUSOR**

The WUSOR programs are products of research conducted on the WUMPUS exploration game. Wenger (1987) says that WUMPUS is a computer game in which the player must slay the Wumpus. The player travels through caves in search of the Wumpus monster. There are hazards which can be encountered such as bats or pits. The player receives warnings and other information to aid in the search. The player decides which caves to move to based on hints given by the system.
Skills required to play WUMPUS include logic and probabilistic reasoning. Time may also be a constraining factor. Therefore, decision making and planning skills are important strategic tools (Wenger 1987, 135).

Goldstein states that becoming skilled at the game is a non-trivial accomplishment; one that requires the systematic development of procedural skills (Goldstein 1979, 54). The WUMPUS game then provided an environment for investigating the use of the computer as a coach or tutor. Wenger (1987, 136-140) briefly describes the WUSOR-I and WUSOR-II programs. The first version of a computer based coach for the WUMPUS game was WUSOR-I developed by Ira Goldstein and Brian Carr. WUSOR-I, the expert-based coach, contained only an expert and an advisor. The expert consisted of heuristic based production rules, and the advisor simply explained moves without applying tutorial strategy. There was no diagnostic student model and the level of play was determined by the student at the beginning of each game.

WUSOR-II was developed so that the system could have some understanding of the student's knowledge state. The expertise required for the WUMPUS game was broken down into five phases. The expertise rules ranged from basic to probabilistic. WUSOR-II's student model was developed using the theory of overlay modeling. Therefore, some account was made for the student's knowledge level. However, the
program was unable to detect the gradual mastery of skills. WUSOR-III, a development of continued research by Goldstein, represents an attempt to further model the student's learning capacity.

Goldstein (1979) describes the WUSOR-III and the development of the theory of the "genetic graph." The genetic graph represents a formalization of gradual learning. It represents the evolution of rule use and the relationships between rules. Rules, or subskills, are represented as nodes. These nodes are linked together by some relationship link. Links can be classified as one belonging to one of the following groups: generalization or specialization, analogy, refinement or simplification, deviation or correction. The links provide the evolution track of the rules. For example, rules in phase two are refinements of rules in phase one.

Further development of genetic graphs grouped the rules and declarative facts, which explain the behavior of the rules, so that genetic relationships between groups of rules could be established. These groups are called islands and are made up of rules which have the same goal. The benefit of grouping rules allows the coach to focus on specific conceptual properties related to the rule group. The islands are linked together through facts and fact/rule links. The rules within the islands are linked together with prerequisite and post requisite fact links so that
problem solving knowledge or planning knowledge can be recognized.

The genetic graph method allows for the tutor's topic selection to be made based on the student's phase. It also provides for multiple explanation due to its evolutionary linking component. The student modeling capabilities of the genetic graph arise from the fact that knowledge is represented as phases. Therefore, a student knowledge overlay on the expert can be constructed to fit the phase level of the student.

The system determines the student's state by hypothesizing that the student does not know the rule if the student's answer is worse than that of the expert at that particular phase. The evaluation of the student's skill occurs as the student is compared to the five "phase experts" in the system. The hypothesis of the phase experts is attached to a node. The belief that a student knows the rule is the summation of the hypotheses.

Thus, the genetic graph offers a modeling technique that allows the coach to discern the level of the student's ability, to justify solutions at a level the student can understand and learn from, and to discover the student's planning knowledge.
WEST

WEST (Burton and Brown 1979) is an example of a computer based coaching or tutoring environment developed for the computer game "How the West Was Won." Burton and Brown (1979) offer some new insights into modeling a student's understanding in an open ended gaming environment.

The game, intended to give practice in arithmetic, is essentially a board game with seventy spaces and three spinners. The spinners give the players three numbers with which to perform arithmetic functions. The player uses the value of the function to move a certain number of spaces along the board. The game, however, allows special moves such as shortcuts or the landing on towns to make it a game of skill.

The underlying theories of WEST are that student's build onto their knowledge base as they play the game and that they learn from their mistakes. The student diagnostic modeling used in WEST is considered to be a hidden type or inferred modeling system. The approach used here is based on the belief that the modeling should not interfere with the student's actions and should not be built from diagnostic questions. Therefore, the primary method for inferring what a student knows or does not know is that of a differential approach.

The differential approach or model means that the student is compared to an expert. Burton and Brown (1979)
suggest that the differential model requires the accomplishment of two tasks. One is the determination of the quality of the student's input as referenced to a number of possible solutions generated by the expert. The other is the determination of the underlying skills that are used in the solutions of both the student and the expert.

WEST's coach has limited information concerning the student. Therefore, the coach must determine why a student's solution was not better. Burton and Brown (1979) suggest the "Issues and Examples" paradigm as a method for evaluating behavior. Issues contain the skills and concepts the student is expected to master; the Issue Recognizer looks for evidence that the student has used particular skills. The Issue Evaluator looks for weak use of the skill. That is, if the student uses the skill, but does not present the best solution, the Issue Evaluator compares skills used in the student's solution to better solutions.

WEST contains three levels of Issues. The first level consists of basic mathematical skills. The second level consists of the skills necessary to play WEST, and the third level is related to general game playing capability.

This modeling technique offers the tutor or coach an approach for helping the student. For example, the coach will know which issue the student is weak in and will be able to provide an example of a better method. While the aforementioned modeling techniques provide many examples and
methodologies for use in cognitive domains, simulation based tutoring systems present a new arena for student modeling and evaluation.

**Simulation-Based Training Systems**

**SOPHIE**

SOPHIE is one of the earliest intelligent computer aided instruction systems (Kearsly 1987, 5). From the research of Brown and Burton (Brown, Burton, and de Kleer 1982), three SOPHIE systems have evolved. SOPHIE I began in early 1973 and was completed in 1975; SOPHIE II was built in 1976, and SOPHIE III was built in the two years following the completion of SOPHIE II. In their paper, "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III," Brown, Burton, and de Kleer (1982) tell of the concepts and constraints of the SOPHIE systems. SOPHIE began as a project funded by the Air Force. The desire was to utilize computers in an electronics troubleshooting course. Therefore, the domain of the SOPHIE systems is electronics troubleshooting. The general purpose circuit simulator, SPICE, was used to simulate specific circuits.

This brief explanation of the SOPHIE projects will focus on SOPHIE III since it represents the latest advancement. According to Brown, Burton, and de Kleer (1982), the purpose of the SOPHIE project was to develop a
laboratory environment which allows the student unrestricted implementation of solution ideas and/or fault hypothesis and which provides a coach to critique the student’s actions and guide the student to better understanding of the domain. Three factors deemed necessary to provide such an environment are reported by Brown Burton and de Kleer (1982). These are:

1. Allow student initiative,
2. Have powerful inference techniques, and
3. Provide good explanations.

SOPHIE III provided for a knowledge engineering testbed and is reported to allow student initiative, have a powerful inference technique and provide good explanation. One of the underlying concerns during the design and development of SOPHIE III is that it must be able to hypothesize and reason about a student’s limited understanding. Therefore, many redundant problem solving strategies had to be incorporated. SOPHIE III also investigated the separation of knowledge from general to specific. In SOPHIE’s case, the development was to have circuit specific knowledge separated from general electronic knowledge.

SOPHIE III is made up of three major modules. These are the electronic expert, the troubleshooting expert and the coach. Electronic troubleshooting expertise represents the goal of removing as many circuit components from
suspicion with each measurement as possible. The purpose of the electronic expert is to make deductions about the measurements to be made. That is, it describes the most effective measurement as a function of the number of components it releases from suspicion as being the faulted component. The expected value of the student's measurement is achieved by a mathematical function related to the number of components that are released from suspicion. The student is critiqued or complimented if his measurement is sub-optimal or near optimum. Student modeling in the SOPHIE systems was a concern to the researchers; however, they focused their efforts on the coaching ability of the system and allowed student modeling to take a less important role in this research.

STEAMER

STEAMER is a simulation-based training system based on a steam propulsion system found on Navy ships. Hollan, Hutchins, and Weitzman (1987) described STEAMER's functions in a their article entitled "STEAMER: An interactive inspectable, simulation-based training system." The article, written in 1984 was reproduced in a 1987 book. At the time the article was written, the student model was considered to be very limited. The system was developed to show students the interrelation of the components of the steam propulsion unit. The program uses a dynamic
simulation with a graphical interface which allows the system to be viewed at a variety of levels. STEAMER uses object oriented programming to develop its graphical interface; the interface is designed to display the status of the components and to allow for their control.

It is considered that there are many possible casualties which could arise in such a complex system. One way STEAMER shows the expert model of the system state is to show the actual meaning of the gauge values in terms of qualitative information. For example, the system shows the derivative of the actual signal so that the student can gain an understanding of the meaning of the signal. This "continuous explanation" is considered more easily understandable than verbal explanation to show the dynamic nature and effective influences on the system. Work continues on the representation method for the expert procedures and on a student model. It is stated that the student model is a simple differential model but no details are given. Hollan et al. (1987), state that the research on STEAMER is to aid in the development of methods that will support students' ability to understand and reason about complex dynamic systems.

Although these well-known simulation based systems are important contributions to the development of intelligent tutoring systems, they focus on the cognitive ability of the student to discern the state of the system and implement a
solution method. The focus of this thesis is on an intelligent simulation based tutoring system which allows the student to manipulate independent objects within a time and space domain. The difference between this ISTS and STEAMER is the independence of the objects.

The student is required to provide effective control of all objects within time and space constraints; that is, the student must be aware of the passing of time and must be aware of the space requirements of the objects. The time and space constraints of dynamic interactive simulation environments presents a challenging task for student performance evaluation. This task description will be presented in the following chapter.
CHAPTER 3
PROBLEM STATEMENT

An inherent part of a tutoring system is student performance evaluation. As suggested by the previously described research, this is usually achieved by comparing the student to an expert. The intent of this research is to establish a performance evaluation system to be used in a dynamic, on-line intelligent tutoring system. The purpose of this evaluation mechanism will be to reveal information which can be intelligently used by the Student Model and the Tutor.

Human Performance in Simulation Systems

There has been much research conducted on human performance measurement systems. However the training of high performance skills such as air traffic control continues to be an area of debate and concern. Training high performance skills brings up many issues and questions. Schneider (1985) states that training programs are often based on assumptions which may not be correct in a high performance domain.

The training of complex skills is often conducted using simulation systems or simulators. Vreuls and Obermayer (1985) point out that there are fundamental
problems associated with performance measurement in simulation systems. Although these are stated in general terms, it is necessary to note them and understand that they influence any performance measurement system.

The fundamental problems apparent to Vreuls and Obermayer (1985) are:

1. the hidden and embedded nature of performance,
2. the lack of a general theory of human performance,
3. the determination of the validity of performance measures, and
4. the establishment of the criteria for performance.

Although each of these problems is significant, performance measurement serves necessary purposes in training situations. One purpose is so that an assessment of the training method may be achieved. Another function is so that an assessment of the student may be conducted. Still another purpose is so that timely performance feedback can be provided. These three functions are required in any system whether it be simple or complex. Intelligent tutoring relies on the ability of the system to perform in a rapid and accurate manner.

Within the field of intelligent tutoring systems, there has been limited research directed at intelligent simulation-based training systems. Well-known research projects such as SOPHIE and STEAMER use simulation as a
means of portraying the system as well as the expert knowledge base.

The research described in this thesis is based on concepts being developed for an ISTS. The ISTS described earlier (for the tutoring of Air Traffic Control) will be used as an example system description. It provides an example of possible inputs and outputs to an Evaluator module.

The assumptions used for this thesis include the use of a complete or perfect expert knowledge base containing expert production rules, the use of a student model to describe and maintain individual student status information, and the use of a dynamic, interactive simulation. It is assumed that the student is required to identify discrete events, analyze their potential effects, and respond in a corrective and timely manner. The objective of this assumed ISTS is to train the skills necessary for efficient and strategic manipulation of objects within time and space constraints. In the next section, the ISTS' system description is explained in general so that the problem statement of this thesis can be expanded.

**ISTS System Description**

The ISTS is a rule based intelligent expert tutoring system which utilizes a blackboard architecture for storing all of the modules' asserted facts. The inference and
control mechanisms allow for the firing of the rules. Each module performs specific fact assertions and retractions according to the system mode and whether the module is activated by the Control. The modules of the ISTS are grouped by the function they serve. The groups are Simulation, Input Analysis, Control, Interface, Instruction, and Expert Knowledge. Figure 1 shows the structure of the ISTS. The groups represent the general system functions, and the names of the individual modules within each group represent individual system functions.

Figure 1. ISTS System Modules.
The Simulation contains the simulator and the simulation software for the particular domain of interest. The function of the simulation is to display a visual environment for the student to observe. That environment responds dynamically to the student's input.

The Input Analysis Group contains the Translator, the Input Filter, the Intelligent Pre-Processor (IPP), and the Interpreter. The function of this group is to determine the reasonableness of the student input for the domain and to pass the input to the instruction group.

The Translator module parses the student's input. Its purpose is to check for syntax errors and pass the student's acceptable parsed input to the Evaluator.

The Input Filter determines the logic errors associated with the domain. These include object specific restrictions and environment specific restrictions. That is, the Input Filter determines if the object is capable of performing a requested state change or if the environment is restricted from the object. For example, in the ATC domain, certain plane types can fly only up to certain altitudes. Also within the ATC domain, airspace is partitioned into sectors, and sectors may be restricted from use by particular plane types such as non-military or by the non-sector air traffic controller.

The role of the IPP is to determine specific domain events known as IPP-situation-facts and IPP-snapshot-facts.
IPP-snapshot-facts are used by the expert in conjunction with the IPP-situation-fact to specify which expert action fact will be asserted (Draman 1988). The IPP-snapshot-facts are used by the Interpreter and the Evaluator.

The Interpreter maintains a record of IPP-snapshot-facts. This record or list is referred to as the delta list and is made up of recorded-snapshot-facts. It is the difference in the IPP-snapshot-facts before the student input occurs and after the input is acknowledged. Therefore, the delta list is updated when an event occurs. The Interpreter tags the facts so that they can be used by the Evaluator. These tags give the Evaluator information about whether the event was generated by the student, whether the event is critical, or whether the event is eliminated.

The Interface Group is designed to allow communication to occur between the system's analyst, who would originally fill the system with domain knowledge, and the system. The Interface Group also allows the student to communicate with the system. The Interface Group is made up of the Knowledge Acquisition and Discourse modules. Because the system is being designed with generic components, the domain dependent data is required to reside in or be input by specific modules.

The Knowledge Acquisition has the purpose of allowing the system analyst a user friendly environment to input
knowledge about the domain (Biegel et al. 1988). The intent is for the Knowledge Acquisition to accept domain data and transform it so that it can be used by the system. The specific knowledge required by the various modules is delivered by the Knowledge Acquisition. For example, domain instructional strategies would be sent to the Expert Instructor Module. The Discourse provides a user friendly environment for communications between the student and the system. The Discourse utilizes menus, windows and messages (Biegel et al. 1988). It allows the student to ask a question, input a comment for a human instructor to review, or request a tutoring mode.

The Control Group contains the Control and the Inference Mechanism. The purpose of the Control Group is to govern the interaction between the system’s modules and to make use of the data structures of the system.

The Control module is responsible for coordinating the system actions. It determines when different operating modes should be made active and when modules should be made active.

The Inference Mechanism provides the reasoning method used by the system. It makes use of the knowledge structure to direct the system through goal states.

The Instruction Group makes use of student information and system data to conclude which method and degree of tutoring is required. This group includes the Tutor, the
Evaluator and the Student Model. The intelligent tutoring process requires the use of instructional strategies which are student specific. To accomplish this, the Tutor uses information from the Student Model which has previously received information from the Evaluator.

The Evaluator is responsible for categorizing error types and for scoring the performance. This is achieved by matching the student’s response to the expert’s response and by considering other system based data. The Evaluator uses the Interpreter’s delta list as well as other system information to provide an evaluation process based on the significance and efficiency of the student’s solution.

The Student Model uses data from the Evaluator and past performance data to determine the student’s current state of knowledge. The Student Model updates the student’s status dynamically so that the Tutor will know when the student’s performance has deteriorated to a point where Tutor intervention would be helpful.

The Tutor is responsible for implementing the best method for tutoring the student based on Student Model information. Domain specific levels of mastery about a topic area reside in the Expert Instructor knowledge base. The Tutor accesses this information to develop lessons in agreement with the student’s level. The Tutor can operate in several modes such as coach or test giver. The Tutor provides the student with remediation on certain skills if
the student is unable to perform a particular task. The level of complexity of lesson design is a function of the student's knowledge state and ability.

The Expert Knowledge group contains the domain knowledge. The modules of the Expert Knowledge Group include the Expert and the Domain Expert Instructor. The Expert provides the knowledge to develop and support the expert solutions. The Domain Expert Instructor is composed of the knowledge pertaining to teaching strategies, and performance characteristics of the domain.

**Student Interaction with a Functioning ISTS**

The aforementioned functions of the different modules are utilized throughout the training program. The student gains access to the system through a log-in procedure which retrieves the student's past performance record. This record, maintained by the Student Model, is the basis on which the Tutor starts a tutoring session.

Tutoring sessions are made up of lessons. The lessons combine to form a lesson sequence. The lesson sequence provides the student with scenarios and situations with which to practice certain skills. The Tutor makes the tutorial decisions such as to what level of difficulty the student is prepared to attempt, what topics to tutor and what scenario to generate for the student. As the tutoring session begins, the selected scenario is displayed on the
Simulation's object environment screen and the simulation is started. The IPP takes a snapshot of the simulation and determines what events are taking place. It expresses these events in the terms of snapshot-facts. The Expert generates solutions to the snapshot-fact events and the Interpreter keeps a list of the IPP's events.

The student interprets the situation and inputs a command necessary to manipulate the objects in the time and space domain. The command is screened by the Translator for acceptance by the system. That is, the system must be capable of understanding the student's input. After the input passes the Translator, the Input Filter reviews the input for environmental or object logic errors.

Once the input has passed the Input Filter, the Interpreter discerns which events have been altered. For example, a simulation event which was in the scenario before the student's input may now be eliminated because of the student's input. Thus, the Interpreter records this change of the events' status in a delta list. The Evaluator is also activated at the time of the student's acceptable input. The Evaluator accesses the Interpreter's delta list to determine if any simulation events have been eliminated, have gone critical or have been introduced by the student's input.

The Evaluator uses the object to which the student communicated to find the Expert solution set associated with
that object. Therefore, the Evaluator follows a decision flow chart and scores the student accordingly. The Tutor can use the information contained within the Evaluator to make other tutorial decisions. At the end of the lesson, the Evaluator sends the student’s score to the Student Model.

The Student Model updates its performance records and determines the student’s ability level. This information is also used by the Tutor to make pedagogical decisions. Thus, the dynamic nature of the ISTS makes for constant module interaction and information flow. The tutoring system thereby creates a complex and dynamic environment for performance evaluation.

**ISTS EVALUATOR**

The ISTS Evaluator requests data from many system modules. These include the Translator, Input Filter, Interpreter, Simulation, Tutor, and Expert. Figure 2 displays the input and output modules interacting with the Evaluator. The inputs are used to aid in the classification of errors made by the student. The Translator passes the student’s parsed input to the Evaluator. The Input Filter sends logic errors committed by the student. These errors give insight to the student’s understanding of the domain environment. The domain environment includes the objects within the domain.
Figure 2. Evaluator - System Interaction.
The Interpreter accepts inputs from the Intelligent Pre-Processor (IPP). The facts, as described earlier, are tagged by the Interpreter in terms of new, eliminated or still there. There are other fact tags such as critical or non-critical, side-effect or scenario attached by the Interpreter. The facts from the Interpreter allow the Evaluator to discover problems in the student's awareness and in the student's strategies.

The Tutor gives the Evaluator the end of lesson message. The end of lesson message allows the Evaluator to know when to conduct the final tabulation and scoring for the errors committed as well as when to clear the evaluation record maintenance systems.

The Expert reveals possible solutions to a particular event. The alternative solutions are ranked so that the student's solution may be matched and so that the student's closeness to expert-like responses may be assessed. For example, a novice attempts to solve a problem in a much different way than an expert. Although a strategy might produce the desired result, there may be a more strategic method.

The Evaluator sends messages to two modules. One, the Tutor, receives information during the course of the lesson. Two, the Student Model, receives information at the end of the lesson.
The Evaluator sends messages to the Tutor so that the Tutor can make effective tutoring decisions while the lesson is being conducted. The Tutor may want to intervene based on the effect of the student's input. This could be in the form of hints or reinforcements.

The messages sent to the Student Model consist of scores for a particular event type. Also included is the total number of events which occurred during the lesson. The Student Model can use this information to arrive at the score for the completed lesson. During the course of a lesson, the Evaluator sends the Student Model facts which correspond to the student's action and the system's reaction.

The dynamic nature of the ISTS makes it necessary for the Evaluator to constantly monitor any possible facts relating the student's input. The proposed solution to this significant task is described in the next chapter.
CHAPTER 4
METHOD

The performance evaluation system described in this chapter is based on the development of a comparison procedure between the student’s input and the expert’s solution method. It is constructed so that some inferences concerning the student can be made. The comparison focuses on discrete events. Events are considered to be the interaction of objects with other objects or with their environment. That is, the environment is assumed to be object oriented. The event types and objects associated with the student’s input are the primary matching parameters. As described earlier, the Evaluator receives input from a variety of sources. An analysis of this input is conducted to facilitate the comparison procedure. The classification and point value of any errors are established based on the input.

Quantitative and qualitative assessment of the student’s understanding will be an output of the evaluation process. These assessments do not depend on the student’s history or other particular student information. They result from the closeness of the student’s procedure with the expert’s procedure or from the student’s inability to achieve a desired solution method. For example, if the
expert does not include a solution close to that proposed by the student, the student’s solution may be inappropriate because it causes the occurrence of other events.

**Hardware and Software Systems Used in This Research**

The hardware system used for this research is a Symbolics 3630 LISP based machine developed by Symbolics, Incorporated of Cambridge, Massachusetts. The software is the Automated Reasoning Tool (ART) version 3.0. ART is an expert system shell developed by the Inference Corporation of Los Angeles, California. This research utilizes ART’s forward chaining capability, schema knowledge representation, fact knowledge representation and rule structure. Symbolics’ Common LISP programming language is used to provide some functions which are not accommodated by ART.

**Solution Description**

The dynamic nature of the simulation system and student interaction makes it necessary to constantly monitor the changes which occur in the simulation and to define these changes in terms of student manipulation of the simulation environment. The solution method investigated in this research considers that the possible events which can occur within the simulation and the trainee environment can be uniquely identified. Causal investigation of missed solutions is achieved through a continuing review of the
delta list of events which is supplied to the Evaluator from the Interpreter.

The proposed solution utilizes schemas, a frame based type of knowledge representation, to establish the distinguishing error features and point value features of the event types. Events are separated into two types. One is simple, and the other is complex. Each event type is classified initially by the system analyst or initializer. The system analyst would state what type of events the system is to be grading and whether these events constitute a simple event or complex event. For example, within the ATC domain, separation would be entered as being an instance of a complex event. Therefore, whenever a separation schema is created, it would contain all of the slots associated with a complex event type. Schema creation is discussed later. Figure 3 presents the schema types and their slots.

Schemas are created and filled by facts which cause the Evaluator rules to fire. Facts utilized by the Evaluator are sent to the blackboard. For example, the Interpreter will send a fact to the blackboard which looks like the following:

Figure 3. Schema Types and Development.
As an example fact relating to ATC, the following fact would be sent to the blackboard:

(Int-eval-recorded-snapshot-fact A01 separation 1247 1245 4 (pl1 pl2) regular critical side-effect original)

Where:

- Int-eval-snapshot-fact is the name of the fact
- A01 is the unique-id
- separation is the event type
- 1245 is the time the event was noticed as impending
- 1247 is the critical time for the event or the time which the event will occur
- 4 is the priority of the event (i.e. of all the events to be addressed this ranks as the fourth)
- (pl1 pl2) are the objects involved in the event
- regular is the Interpreter’s tag1
- critical is the Interpreter’s tag2
- side-effect is the Interpreter’s tag3
- original is the Interpreter’s tag4

Figure 4 shows a complete list of the fact templates for facts utilized by the Evaluator.

After the student’s input, the Evaluator checks the delta list and the event tags to determine the status of the events. These tags reveal whether the event is in a critical state, whether the event was caused by the student or by the scenario, and whether the event is still-there, eliminated or original. This information allows the Evaluator to make inferences concerning the student’s input. The Interpreter tags are displayed in the Table 1.
**Figure 4. Facts Sent to the Evaluator.**

<table>
<thead>
<tr>
<th>From: INTERPRETER</th>
<th>Module</th>
<th>Fact Template</th>
<th>Fact Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(INT-EVAL-SECOND-ALERT ?ALERT)</td>
<td></td>
<td>(INT-EVAL-SECOND-ALERT MARGINAL)</td>
<td></td>
</tr>
</tbody>
</table>

| From: EXPERT | | |
|----------------|--------|---------------|--------------|

| From: TRANSLATOR | | |
|-------------------|--------|---------------|--------------|
| (TRANS-EVAL-PHRASE ?PHRASE-ERROR ?UNIQUE-TRANS-ID) |        | (TRANS-EVAL-PHRASE PHRASEOLOGY TP01) |

| From: INPUT FILTER | | |
|-------------------|--------|---------------|--------------|
| (INFIL-EVAL-STUDENT-LOGIC LOGIC-ERROR ?LTYPE) |        | (INFIL-EVAL-STUDENT-LOGIC LOGIC-ERROR ENVIRONMENT) |

| From: TUTOR | | |
|--------------|--------|---------------|--------------|
| (TUT-EVAL-END-LESSON END-LESSON-MESSAGE) |        |               |              |
The Interpreter tags are defined below. Tag 1 is intended to reveal whether the event could have been initiated by the system at start time or whether the event pops up during the course of the lesson. Tag 1 is not significant at this time (within the ATC domain). Tag 2 displays the current state of the event in terms of time. Non-critical indicates that the event is impending, but the student still has enough time to implement a solution. Critical means that the student must implement a solution if he is to have some positive effect on the event; otherwise, the event will occur. Late means that the event has already occurred, and the student has no chance to recover. If the late tag is there, the total negative points will be assessed for the event type.

Tag 3 represents whether the event is the result of the student’s input or a result of the system. The side-effect tag is representative of the student’s input.
effect whereas scenario is representative of the system effect. The final tag, tag 4, indicates the status of the event as it relates to the scenario. An event can be new to the scenario in which case it is tagged as original. It can still be within the scenario in which case it is tagged as still-there; or it can have been eliminated from the scenario in which it is tagged as eliminated.

All of these tags serve a purpose throughout the evaluation process. Essentially, they provide the Evaluator with its ability to infer the student's response to the situation. The Evaluator follows a decision flow chart. One portion of this decision flow chart is displayed in Figure 5. The entire flow chart is detailed in the Appendix. The decisions in the flow chart are used to fill the schema slots associated with the event.

**Schema Creation**

When a new event is recognized by the Evaluator via the delta list or via the Translator, the Evaluator creates a unique schema for the event based on the event type (i.e., simple or complex). The point value slots have been previously filled by the system analyst. An example of slot inheritance is shown in Figure 6. This figure shows that initially a user would specify the positive and negative point values associated with a particular event type. When a new event occurs and a schema is created for it, it
Figure 5. Part of Evaluator Decision Flow Chart.
NOTE:
--- schemas part of the system

--- schema created as a result of system-analyst input

** slots inherited from event schema

*** slots inherited from complex schema

Figure 6. Example of Schema Slot Inheritance.
inherits the point value slots from the schema created for that particular event. It also inherits other slots from any schemas of which it is an instance.

The Evaluator recognizes new events as those which have the Interpreter's tag 4 set as original or those which the Translator sends as a phraseology error with a fact which describes a unique-trans-id. Refer to Figure 4 to review the Translator fact.

When an event schema is created, its unique identification is added to a major schema for the Evaluation system called the record-keeper. The record-keeper's slots are named for the possible event types which can occur. The record-keeper is used to keep a list of the unique tags of each type of event, as shown in Figure 7.

The record-keeper's list does not maintain any of the slots for the specific schemas. The specific schemas maintain all of their information. In the previous example of the Interpreter's input fact (page 44), A01 is the unique-id which is sent to the record-keeper's separation slot. The record-keeper allows for the categorization, and mixing of the event types in any order so that the evaluation can be based on not only the number of errors of a particular event type but also on the individual assessment of each event type. That is, the combination of the individual scores can present a more detailed and perhaps more accurate assessment of the student's ability
Figure 7. Record-keeper Schema.
rather than a general number given for a particular type of error without reference to causal scoring.

In the case of the hierarchy of skills where one skill group incorporates skills from other groups, the record-keeper can be employed to select the event types which are referenced by the higher level skill group. Again, the scoring is based on the student’s ability to individually attain a mastery of skills.

**Schema Slot Values and Their Use**

The slots for the schemas were determined so that they can be filled and utilized by the tutoring system. The filled slots can provide evidence to the Tutor as to how the student is performing during a particular instance within the lesson. The schema slots are different for each event subgroup, that is, whether the event is complex or simple. Provided below is an explanation of the schema slot values for the complex event schema. The simple event schema slots, where they are the same as the complex schema type, represent the same meaning and information. The slots for the complex event schema are:

- Event name
- Negative point value
- Positive point value
- Priority
- Goodness
- Response Time
- Slack Time
- Method
- Environment Logic
- Object Logic
Inefficient
Side-effect critical
Side-effect non-critical
Eliminated by-product
Missed critical event

The student's input causes the Evaluator to fill the slots according to the decision flow chart. The inferences are made based on the student's ability to match the Expert's solution. If the student does not match the Expert, a review of the delta list allows the Evaluator to make inferences concerning the causes.

The event name is for the event for which the schema is being created. This slot is filled in at the creation of the schema during the check of the delta list. If the list contains an original event, then a schema is built based on that event type. The negative point value and positive point value slots are required for the quantitative assessment of the error. The idea here is that the positive points can be used when the student performs correctly or within a reasonable or acceptable solution method. The positive points are used in conjunction with the values in the priority and goodness slots to give the student partial credit for the strategy used. For this research, the priority and goodness are multiplied together. The result is then divided into the positive point value to give the student score.
The negative points are used when the student performs incorrectly due to some error in judgment, method, time or logic. That is, the Evaluator follows the decision flow chart and makes inferences concerning the reason for the error. This decision procedure fills in one of the following slots method, environment logic, object logic, inefficient, side-effect critical, side-effect noncritical, or eliminated by-product. The definitions of each of these slots, as well as the definitions of the priority and goodness slots are provided below. Negative scoring is detailed after these definitions.

**PRIORITY**: This number is available from the Interpreter's recorded-snapshot-fact. The priority represents the need for the student to respond to an event. For example, if there are two events and one has a priority of 1 while the other has a priority of 2, the student should be working on the event with the priority of 1 first. The Student Model can use this information to track the student’s ability to address the events with the highest priority. The priority is used to discover the student’s ability to assess the whole domain environment at once and to determine which events require response first.

**GOODNESS**: This number is available from the Expert's action-fact. The goodness represents the ability of the student to perform in an expert-like fashion. Novice students most likely perform differently from experts.
Novice student strategy may focus on one move to correct one event while an expert may use one move to correct two events. Tracking the student’s goodness over time can help the Student Model to place the student in a particular level such as novice, intermediate, etc., and expert.

RESPONSE TIME: Response time is the difference between the time the event was created and time the student’s input is acknowledged concerning the event. The Interpreter fact holds the time of creation for the event. The Student Model can use the information to track the student’s response time over a period of lessons or over a period of time on the system. This will allow the Student Model to discover the student’s ability to address situations rapidly.

SLACK TIME: The slack time is the difference between the events’ critical time and the time the student’s input was acknowledged for that particular event. It represents how close the event comes to being critical. This may be important because if the student does not respond with enough slack time, he may not be allowing time for unexpected situations such as emergencies.

METHOD: In the early stages of development of the system, it is assumed that the phraseology in the ATC domain is structured enough so that there can be a complete list of phrase possibilities. Also, it is assumed that the Translator will allow only those phrases which are a part of
the acceptable list to pass through the system. Finally, it is assumed that there is a perfect Expert (one which has all possible solutions for a particular problem). If a student attacks a problem by addressing an object yet not matching any of the Expert's keywords, it is assumed that the student's method is not appropriate for the situation. The information from this slot can be used by the Student Model to determine how many times the student makes a method error. Over a period of time, the Student Model can determine the student's real understanding of the domain environment and real ability to manipulate the objects in an efficient manner.

ENVIRONMENT LOGIC: The environment logic slot is filled when the student's input does not provide any changes to the Interpreter's delta list. The Evaluator then looks for facts (from the Input Filter) concerning logic errors. Logic errors are separated into two groups, object logic or environment logic. This is because one type of logic error may be more important than the other. For example, within the ATC domain, the student's environment logic errors are considered more important than his object logic errors. Environment logic represents the student's ability to direct objects into acceptable space within the environment. For instance, a student may try to direct a plane into restricted airspace or into the middle of a hurricane.
OBJECT LOGIC: The object logic represents the student's ability to understand the physical capabilities of the object. For example, the student may try to send a small plane to an altitude it is physically incapable of attaining.

INEFFICIENT: The inefficient slot provides information about the student's ability to effectively eliminate an event. An inefficient input means that student is able to match the Expert's keywords; however, the student does not match the Expert's argument range. If there has been no side effect nor logic error created, then the student did not direct the object to change its position enough to effect a change in the Interpreter's delta list. This information can be used by the Student Model over a period of time to track how often the student makes an inefficient input and during what lessons. For example, in high traffic situations the student may have more inefficient inputs because he is afraid to take risks.

SIDE-EFFECT CRITICAL: This slot is filled if the student's input causes a side effect to be created and that side effect is in a critical state. This information can be used by the Student Model to assess the student's concern for other objects and to assess the student's ability to understand the consequences of his actions. For example, over a period of time, the student may cause less and less critical side effects to occur. This may mean that the
student is beginning to think of the whole environment rather than just one event.

**SIDE-EFFECT NON-CRITICAL**: This slot is filled if the student's input causes a side effect to be created but the side effect is in a non-critical state.

**ELIMINATED BY-PRODUCT**: This slot is filled during the flow of the Evaluator through its decision chart. The eliminated by-product (may not occur often) results from an event being eliminated from the delta list. However, the eliminated event cannot be traced back to be a result of the student's input.

**MISSED CRITICAL EVENT**: This slot is filled when the student works on an event that is not in the critical state. That is, the slot for the critical event is modified to reflect this and the student is evaluated for missing this critical event.

The slots just mentioned, beginning with method, provide an indication of error cause; they also represent causes which may be more or less severe. Therefore, scoring is based on the slot that is has been filled and is based on the percentage of the negative points the student is to be assessed.

The system analyst initializes the value or percentage of the negative points the causal slots represent. For example, if the student commits an error which results in a critical side-effect error, the student may be deducted (1.0
* (neg point value)). Whereas if the student committed an object logic error, the student may be deducted (0.2 * (neg point value)). Table 2 shows the initialization facts which must be set by the system analyst. The fact names should remain the same but the values can be changed depending on the domain and the importance of the committed error to the domain.

**TABLE 2**

INITIALIZATION FACTS FOR SCORING STUDENT

<table>
<thead>
<tr>
<th>FACT NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval-scorolog</td>
<td>0.2</td>
</tr>
<tr>
<td>eval-scorelog</td>
<td>0.5</td>
</tr>
<tr>
<td>eval-scorsec</td>
<td>1.0</td>
</tr>
<tr>
<td>eval-scorsenc</td>
<td>0.3</td>
</tr>
<tr>
<td>eval-scorineff</td>
<td>0.1</td>
</tr>
<tr>
<td>eval-scormeth</td>
<td>0.4</td>
</tr>
<tr>
<td>eval-scorbyp</td>
<td>0.5</td>
</tr>
</tbody>
</table>

During the course of the lesson, the Tutor can use the information concerning the student’s use of time, the introduction of side effects by the student, or the student’s ability to reach expert-like solutions (accounted for in the goodness slot) as well as the other slot values to perform a dynamic investigation of the student’s ability. This information will allow the Tutor the capability to give...
the student constructive criticism or positive reinforcement while the lesson is in progress.

**Schema Retraction at the End of the Lesson**

The schemas, created by the Evaluator, are used to maintain a record of the student's actions during a lesson. Again, this record is not a part of the student model and is only used for the evaluation purpose. The record-keeper will maintain the lists of the schemas for the particular event types. At the end of the lesson, the Evaluator will receive a fact from the Tutor which states that the lesson has ended. This fact will cause the Evaluator to retract all schemas and to clear out the record-keeper schema. Therefore, for each new lesson, the associated event occurrences will be maintained for use only during that lesson. For the student's individual lesson sequence performance and overall past performance, records are maintained by the Student Model.

**Scoring Information Sent to the Student Model**

At the end of the lesson, the Evaluator sends the student's score facts to the Student Model. The score for each event, whether positively or negatively scored, is maintained in the schema slot called student score. At the end of the lesson, the Evaluator sums all of the student score schema values for a particular event type. For example, within the ATC domain, if the student committed
three separation errors during the lesson, the record-keeper's separation slot value has a list of those three schemas. The student's overall score for separation is based on the summation of the three individual student score slot values. The Evaluator to Student Model facts are sent at the end of the lesson. These facts are of the form (eval-sm-sum-score ?type ?score).

During the course of the lesson, as stated previously, the Evaluator fills in the slot values of the particular schemas. When these slots are filled, the Evaluator will send facts to both the Student Model and the Tutor which state that the slot was filled. This will allow the Student Model to dynamically track the student's performance and to keep a record of those parameters that affect the student's status, learning ability, learning habits and skill level. The purpose of this is to allow a greater depth to the Student Model's capacity to model the student. For example, the student may be committing a large number of critical side effects and may be performing at a level of a novice (as would be indicated by the goodness of the solution method).

This kind of information may be more meaningful in establishing the student's learning level than a straight score. The Student Model then can deduce the student's overall learning status and capability as a function of many parameters over the course of many lessons. The Tutor will
also be able to function with a greater understanding of the current student response. The results and conclusions of this research are presented in the chapters following.
CHAPTER 5
RESULTS

The issues of this research: (1) creating a dynamic on-line evaluation scheme, (2) providing objective measures of performance, (3) investigating a generic methodology, and (4) providing a real time evaluation method were tested through an implementation of a portion of the Evaluator module. The inputs of the interacting modules were simulated to represent common facts that would be asserted by these various modules.

The implemented version of the Evaluator (EVA) tested the creation of the schemas through simulated input of the Interpreter and of the Translator. The slot values of the created schema were modified through the use of simulated data. The simulated input of the Interpreter, Input Filter, Translator, and the Expert caused EVA to follow the portion of the decision flow chart given in Chapter 4, Figure 5. The implementation also tested the creation and utilization of the record-keeper schema. EVA also tested the assertion of facts to the Student Model which revealed the total score for an event type. Along with this, EVA tested the retraction of the schemas and the resetting of the record-keeper slots by means of a simulated Tutor input stating that it was the end of the lesson.
EVA is not incorporated in any functioning ITS; however, this is the intent of future research on the ISTS project. EVA was able to demonstrate the functions in the aforementioned paragraphs. The results indicate that a dynamic, semi-generic evaluation procedure can be established.
CHAPTER 6
CONCLUSIONS

The objective of this research was to establish a performance evaluation system to be used in a dynamic, on-line tutoring system. It also intended to produce performance parameters that could be used to help the Student Model and the Tutor react intelligently to the student's input. The implemented version of the Evaluator, EVA, demonstrated that there can be a dynamic performance evaluation mechanism.

The student evaluation mechanism developed in this study is a performance-based system. There is much research as well as debate concerning the usefulness and acceptability of performance-based evaluation systems. The controversy emerges from the subjectivity associated with grading performance. In an intelligent tutoring system, however, the subjectivity should be reduced because the computer system provides a more stable decision making environment. It does not respond in a "sour grapes, sweet lemons" way as might a human instructor. Although the ISTS is based on the knowledge provided by a human expert, the student's individual characteristics are not a factor in the evaluation process. Individualized teaching and not
individualized grading is the goal of the intelligent tutoring system.

The evaluation tools, the modification of the slots in the created schemas, provide information that can be used dynamically throughout the lesson by the Tutor and can be used dynamically throughout the lesson sequence by the Student Model. The established parameters allow the Tutor to make effective tutoring decisions and they allow the Student Model to make a more complete analysis of the student.

Because the other modules of the ISTS are not available for integration at this time, the Evaluator was unable to be tested completely in an on-line tutoring system. Also, the implemented portion of the Evaluator was essentially a testbed for ideas and is not the complete Evaluator module. However, the complete flowchart is attached in the Appendix.

This research developed a generic approach to an evaluation mechanism. While a system analyst has to fill in specific values for topics or events which can be related to performance, this data can be accepted into the system during knowledge acquisition for system initialization. This required data represents only a limited number of values as compared to a complete revision of the Evaluator module for each specific domain.
As stated in Kelly (1988), the performance measurement system must carefully define the purpose it is to serve. The use of the ISTS to demonstrate the inputs and outputs of a dynamic, simulation-based, expert tutoring system reveals two basic parameters which are common to any domain being trained in a similar environment. These are time and space. The manipulation of objects through time and space incorporates fundamental movements of the objects either to avoid or make contact with one another during a particular time frame. These fundamental parameters allowed for the development of a mostly generic performance measurement system.

The validation of this system is unable to be performed without the integration of a functional tutoring system. The research presented in this thesis provides a basis for continued research in the area of dynamic ITS evaluation procedures. Areas of further research are presented in the next chapter.
CHAPTER 7
FURTHER RESEARCH

Research indicates that there are as many human performance measurement systems as there are systems involving human training. This appears to be especially true during the training of complex tasks in a simulation environment. Thus, future research in the area of human performance seems inevitable.

For further research relating to this thesis, effort must be applied to human performance measurement systems in an intelligent simulation-based tutoring system. This thesis was based on assumptions which may not be valid in a completely functional intelligent tutoring system. Therefore, further research needs to address these assumptions. The assumptions include: (1) the use of a domain which has a structured and limited phraseology, (2) the use of a perfect expert; that is, the expert is able to generate every solution to any problem that occurs, and (3) the use of single inputs.

For the first assumption, many domains may not require that the phraseology be structured; therefore, a strict comparison procedure with the Expert module may cause the system to incorrectly evaluate the student.
The second assumption, the requirement of a perfect expert, may not be realistic in a complex, dynamic, and infinitely manipulable environment. A student's solution may be correct; however, the assumption of a perfect expert may cause the system to grade the student improperly.

Finally, the third assumption, that of a single input, can be challenged in many domains. However, the complexity of the evaluation system may allow for this assumption to be valid. For instance, if the expert provides solutions for each event in a stepwise manner, the student could be graded on for each step in the process. Again, the assumption of the perfect expert is made.

Not only is it necessary to review the aforementioned assumptions, but it is also necessary to review the use of a performance-based measurement system. Because the tutoring system is a computer based system, the idea of using criterion referenced measurement could be investigated. The use of criterion referenced performance measurement in a simulation training system will provide some definitive numbers to be used for performance measurement. Criterion such as start of turn, turn rate, number of deviations from desired path, number of inputs used to direct the object from one point to another, use of object's fuel, and time delay such as landing or taking off in the ATC domain could be made available from the system. This may be the desired approach when the domain has the possibility of an infinite
number of solutions. Therefore, the student solution set may not match any of the expert's solution methods. Instead of grading the student wrong, the parameters mentioned above could provide some insight as to the student's ability.

Finally, the validation of the evaluation mechanism is not only a possible area of further research but it should be an area of required research. The evaluation system is the means by which the student's knowledge state is inferred and the means for making some pedagogical decisions. The usefulness of the tutoring system is a direct function of its ability to use meaningful measurements and criteria to effectively monitor and evaluate student performance. This is an on-going and debatable as well as testable area. It is hoped that the research described in this thesis has provided an intriguing approach that will be further investigated.
APPENDIX

EVALUATOR DECISION FLOW DIAGRAM
INTERPRETER - EVAL -SECOND - ALERT
DETERMINE WHERE TO START ON
EVALUATOR FLOW DECISION FLOW CHART

SECOND ALERT
MARGINAL -INEFFICIENT-INPUT
OR DIRECT SIDE EFFECT

SECOND ALERT
ADDITIONAL-SIDE-EFFECT
OR SATISFACTORY INPUT

LATE EVENT?

MODIFY SCHEMA VALUE
STUDENT SCORE
WITH NEG POINTS
1. NEW SIDE EFFECT EVENT?
   - Y: NEW SIDE EFFECT CRITICAL?
     - N: MODIFY SCHEMA VALUE: NON-CRITICAL SIDE-EFFECT
       - MODIFY SCHEMA VALUES:
         - STUDENT SCORE
         - REACTION TIME
         - SLACK TIME
     - Y: MODIFY SCHEMA VALUE: CRITICAL SIDE-EFFECT
       - RETRACT SCHEMA WHEN MESSAGE FROM TUTOR SAYS END OF LESSON
       - SEND STUDENT SCORE TO STUDENT MODEL
   - N: EXIT EVAL LOOP
MODIFY SCHEMA
ELIMINATED BY-
PRODUCT (NOT
STUDENT OBJECT)

FOR ALL ELIMINATED
CRITICAL EVENTS FOR
STUDENT OBJECT
ASK EXPERT FOR
SOLUTION

ARGUMENT RANGE
MATCH ?

MODIFY SCHEMA
VALUE: GOODNESS
PRIORITY

NEW SIDE EFFECT
EVENTS ?

FOR ALL
NEW SIDE EFFECT
EVENTS

CRITICAL
SIDE EFFECT
EVENT ?

MODIFY SCHEMA
VALUE: NON-CRITICAL
SIDE-EFFECT

MODIFY SCHEMA
VALUE: CRITICAL
SIDE-EFFECT

MODIFY SCHEMA
VALUES:
STUDENT SCORE
REACTION TIME
SLACK TIME

RETRACT SCHEMA WHEN MESSAGE FROM
FROM TUTOR SAYS END OF LESSON

SEND STUDENT SCORE
TO STUDENT MODEL

MODIFY SCHEMA
ELIMINATED
CRITICAL EVENT ?

Y

Y

Y

N

Y

N

N

Y

N

Y

N

Y

N

N
ELIMINATED NONCRITICAL EVENT?

EXIT EVAL LOOP

ELIMINATED NONCRITICAL EVENTS STUDENT OBJECT?

FOR ALL ELIMINATED NONCRITICAL EVENTS FOR STUDENT OBJECT ASK EXPERT FOR SOLUTION

MODIFY SCHEMA ELIMINATED BY-PRODUCT NOT STUDENT OBJECT

KEYWORD MATCH?

ARGUMENT RANGE MATCH?

NEW SIDE EFFECT EVENTS

FOR ALL NEW SIDE EFFECT EVENTS

CRITICAL SIDE EFFECT EVENT

MODIFY SCHEMA VALUE: NON-CRITICAL SIDE-EFFECT

MODIFY SCHEMA VALUE: CRITICAL SIDE-EFFECT

LOGIC ERRORS?

MODIFY SCHEMA VALUE: LOGIC RETRACT LOGIC FACT

WARN EXPERT OF SOLUTION ERROR

MODIFY SCHEMA VALUES:
STUDENT SCORE
REACTION TIME
SLACK TIME

RETRACT SCHEMA WHEN MESSAGE FROM FROM TUTOR SAYS END OF LESSON

SEND STUDENT SCORE TO STUDENT MODEL
REFERENCE LIST


