RC 16652 (#73857) 3/19/91
Computer Science 26 pages

Research Report

A Framework for Proactive Interactive Adaptive Computer Help

Ted Selker

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

NOTICE

This report will be distributed outside of IBM up to one year after the IBM publication date.

IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

A Framework for Proactive Interactive Adaptive Computer Help Thesis Proposal

Ted Selker
City University of New York Graduate Center
33 West 42th Street
New York New York 10036

IBM, Thomas J. Watson Research Center
P.O.Box 218 Yorktown Heights, N.Y. 10598
Selker@IBM.COM

December 1990

1 Overview

User interfaces can be difficult to learn to use. When problems occur, computers respond with generic, typically difficult to interpret feedback (if any). This thesis presents a framework enabling a new style of computer help to alleviate these problems. An interactive computer user scenario allows feedback from a system as a user works with it. An adaptive help system would learn about and change in response to a user. A proactive interface would attempt to anticipate the needs of a user. This thesis's framework combines these enabling concepts into a user interface scenario.

The approach utilizes dynamic models of both the user and the domain to facilitate and guide the user's goals. The feasibility of a system that can adapt to a user's working level of ability in order to give help during a computer session is demonstrated. This system, called COgnitive Adaptive Computer Help (COACH) is a shell for enabling adaptive help systems be studied. The thesis also presents a study showing that an adaptive strategy can improve user confidence and productivity.

This proposal includes the following sections:

1. **Objectives of thesis** describes the set of scientific questions which this thesis addresses.
2. **The Scenario** introduces the framework for an automatic adaptive help facility by walking the reader through hypothetical users' work sessions.
3. **Literature Search** discusses previous work related to the research issues presented in this thesis.
4. **Related Research in User Interface and AI in Education** sets the research in the context of an AI framework and discusses its relationship to developments in user interface research.
5. **Architecture** describes the central functional concept of an adaptive user model and lays out modules which can enable such a scenario.
6. **An Adaptive User Model (AUM) shell** shows that the system which instantiates the scenario can be used as a tool for creating adaptive help systems.
7. **Status** describes the state of implementation.
8. **Evaluation** describes experiments and evaluation of the implementation.
9. **Future Directions** research issues which when pursued could extend this thesis work.

2 Objectives of this Thesis

Since the invention of computers, designers have been striving to make them easier to use. We try to make computers more "user-friendly", thus better matching the computer's actions to expectations. Still, users find themselves needing classes, tutoring, help, or reference materials. Attempts to computerize these teaching roles has created an active field of research. The impact and acceptance of computers in teaching roles continues to be elusive.

Creating computer interactions so natural that they require no learning would allow all user effort to be focused on the primary task. This being (so far) unattainable, we are instead concerned with giving the most effective assistance to users while they try to focus on their work.

Most uses of computers in education focus on a single objective specified by the desinger. This thesis focuses instead on tracking the user's objectives in a problem-solving session. Is it possible for a computer to work with a user to produce an educational scenario concerning their goals? Can a computer decide when to interact productively to advise a student? Can a computer demonstrate adaptation to these educational goals *while* a user is working during a productive session? These questions drive this thesis.

The following is a list of major objectives of this thesis:

- A goal of this thesis is to develop a scheme for tracking and adapting to a user which runs concurrently with the interpreted computer program interface that the user is trying to use. This will demonstrate the feasibility of a computer interface which uses AI reasoning and learning techniques to guide its reactions without introducing delays in the user-system conversation.

- A goal of the research is to demonstrate the use of machine-learning mechanisms to shift computer education paradigms away from a classroom format and towards an apprenticeship, learn-while-doing approach.

- A goal of this research is to demonstrate advantages of an automatic adaptive help system relative to a standard passive help system.

- A goal is to create a tool for enabling research in interactive Adaptive User Model (AUM) help systems. Until now it has been believed that an adaptive interactive teaching system was infeasible. The COACH system reported on in this thesis demonstrates this feasibility. It is also meant to be a system with which a researcher can test ideas about adaptation in education, develop adaptive pedagogical approaches and build working adaptive help systems for text-based interfaces.

  A system which implements the scenario in text based interfaces is included and evaluated as proof of the claims of the scenario. It can be connected to a computer language or a computer interface interpreter.

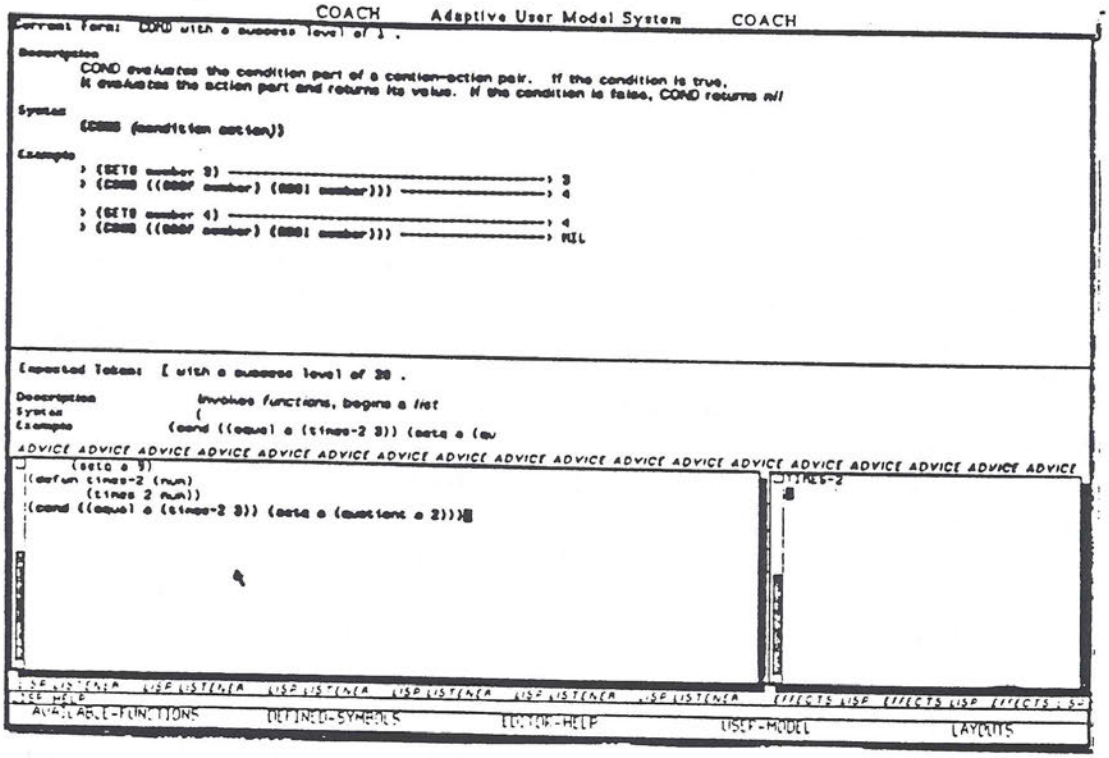

Figure 1: The System Interface

3 The Scenario

What is proactive interactive adaptive computer help? The scenario presented in this thesis expects a user to need help while working on solutions to problems in a curriculum or attempting to do productive work. The system creates a record of a user's experience and expertise in an *Adaptive User Model.* Machine-learning and reasoning adapt to the user presenting proactive appropriate help. Both the user and the computer initiate help in a *mixed initiative* interaction.

Hypothetical users working at three levels of proficiency will introduce the paradigm for learning the interpreted Lisp language:

– Freshman Bill is taking his first programming class. He has attended two classroom sessions and is sitting down for his first time in front of the system. The interface he is working with is segmented into panes: an input pane to type and edit work, panes to present help, an output pane and a fixed menu (figure 1).

To help Bill get started, the system encourages him to type an open parenthesis beginning an S-expression or a defined word. Examples show him this. He types (. The Help changes to tell him that he must type a function name and gives an example. Function, S-expression and defined word are concepts in the systems's domain knowledge lattice. Bill remembers these words but doesn't quite remember what they mean. An **example** of the use of a function is displayed to get him started. Bill types **AD.** The help window tells him that no known function starts with AD, and suggests that he type **rubout.** He could press the mouse button to browse available functions, but PLUS, not ADD was the function he now remembers from his class and types it. As Bill types a space after **PLUS,** an example of using PLUS, together with a simple description and a simplified syntax is presented on the help pane. The context dependent help allows Bill to avoid the usual startup stalemate, in which a user does not know quite what to type to get started. Novice programming problems which involve getting syntax and ideas together at the same time have also been averted.

– Sophomore Harry is trying to write a program. He types **DEFUN,** gets syntax help which reminds him that he must name the function being defined, and then gives it an argument list. The system gives him an abbreviated syntax not showing the difficult argument types (optional arguments, keyword arguments,

etc.). He types **TIMES-2 (I) (PLUS** . The system turns its focus to helping with the PLUS function. An example of a use of PLUS he has already made is displayed. He realizes he really didn't mean to add numbers. He back spaces and types **TIMES I 2)**. The system changes its focus of help to TIMES as he is typing it and back to DEFUN when he is done with TIMES.

Intermediate programmers like Harry often have problems keeping track of context and appropriateness of program pieces. COACH works to keep this type of programmer oriented through context sensitive help and user examples.

– Expert programmer Connie is working on an internal part of the Update-Rule computer program. A model of her expertise allows the system to know to present very little help. When she types **(SETF**, the system shows the very complex argument list syntax for the SETF function. If she uses a function for which no help has been written, the system reaches into that function's definition to present an argument list for it. When she makes an error (ie. wrong argument type) the system changes its view of her slowly. If she keeps making errors, it will change its opinion of her more quickly; it shows her examples and reminds her of things which are related to the constructs she is using and the language concepts involved.

Expert programmers must be aware of anomalous as well as simple relationships between parts of a computer language. Experts are likely to use (even if they don't memorize) sophisticated syntactic features. If Connie were using function or variable names which she had not yet defined, the system would put these names on a list of undefined functions. A menu would allow Connie to select from these names to aid her in remembering to define them later COACH shows experts the delicate anomalous things about parts of Lisp they are using without bothering them with introductory information.

The above sample scenarios are examples of an adaptive user model tracking a user that illustrates the integration of the educational scenario with the user's work scenario.

The completed thesis will include a Video demonstrating the above scenarios in a temporal way in a working COACH system.

I will now motivate this approach and describe its innovations, presenting a brief overview of past relevant work.

4 Review of the Literature

The framework presented in this thesis is a vehicle for research in human computer interaction and AI as applied to education. The following is a literature review of related work in these fields.

The use of computer systems for teaching has been called Computer Aided Instruction (CAI), Intelligent Computer Aided Instruction (ICAI), Artificial Intelligent Computer Aided Instruction (AICAI), or Intelligent Tutoring Systems (ITS). These names have been created by their proponents to reflect the technological and research progress through the years. A common component of all such work is a prescribed educational goal to which students are guided with subgoals and tasks. The scenario can take the form of text with comprehension tests or problem sets or educational games. Collectively these computer teachers can be referred to as Intelligent Tutoring Systems (ITS).

Research in ITS has included experiments using Artificial Intelligence (AI) representation, reasoning and machine-learning techniques to direct a tutoring session [43, 30, 41] Progress inand the role that AI has played in ITS is expanded on below. The following sections describe alternatives to ITS, concentrating on student motivated teaching scenarios. These include:

- help systems that answer [[help" questions a user asks.
- coaching systems that remark on user problems and successes as they occur.
- critic systems to which a user brings a "finished" assignment for evaluation.

.

4.1 Tutoring Research

Tutoring Systems depend on syllabi to guide a student. Computer tutors have often guided student lessons by responses to questions. Other ITS work uses more sophisticated techniques to help guide a user through a lesson plan.

ITS work dates back to the early sixties. Suppes may have been the first [43] to describe the classic CAI method of mechanizing the programmed text book. In place of a programmed textbook, a user reads text and questions from a computer screen. The user works through the text by typing word or number responses to problem questions. Even early systems varied their responses relative to a user's knowledge, something that current commercial "help" systems fail to do. Most commercial tutoring systems use this mechanized programmed text book method of presenting information to a student.

ITS research has taken the syllabus approach to learning much farther than the initial programmed textbook efforts. John Anderson's Lisp tutor has a sophisticated way of presenting the lesson questions as programs for a student to write [30]. Routines and programs are designed to teach about a particular concept or tool. The student's solution can vary from the teacher's prototype in the naming of variables, but cannot vary the functions used; for example, a student cannot use the "IF". function where the system expects the :"COND" function. The student answers questions and writes programs; the system guides the student through the syllabus. The system certifies a student as a learned programmer relative to the problems in the syllabus that have been completed correctly.

Anderson's system improves the learning abilities of Lisp students. His system expanded the CAI programmed textbook "if, then" syllabus. Students' progress is guided by a production system; the word or number answers of early CAI systems are replaced with programs the user must write. Efforts to allow students using Anderson's system to "explore the system", while working a problem set [1] has only hindered students. In this otherwise controlled learning environment, the flexibility seems to distract students.

Seminal work in applying AI in the field of education is typified by John Seely Brown and Richard Burton's productive collaboration. Brown and Burton's DeBuggy [4] introduced knowledge representation and reasoning into ITS. In grade school studies, they showed Debuggy could teach a student about long subtraction with carrying, understanding the student's mistakes better than a teacher. Their approach to teaching subtraction was to catalogue the 120 or so possible types of mistake a student can make while doing a subtraction problem. The system uses a static sub-skill lattice to characterize what skills might be missing to generate errors in an answer to a problem. For each possible mistake, the system has knowledge describing underlying missing concepts which could be responsible for it. Debuggy used a sophisticated representation of the problem domain enabling a reasoning approach to catalog all possible mistakes. Pre-analyzing the entire solution and error domains gives the system the ability to explain all incorrect subtraction algorithms.

The reasoning approach which Brown and Burton used in Debuggy required them to *completely* describe and analyze all possible subtraction errors. Many domains of interest are much larger than subtraction; identifying all possible mistakes in them is usually impractical. In fact, Brown and Burton found teachers seldom understood subtraction in the detail that the Debuggy approach required.

4.2 Teaching with Simulations

Another approach to ITS includes games with simulations. Game scenarios often include a consistent simulated environment referred to as a microworld. Microworlds and other game teaching approaches have the advantage of addressing student motivation as an explicit goal.

Burton and Brown's electrical circuit trouble-shooting learning environment (SOPHIE 1, 2 and 3) [27] is an example of ITS work with simulation, environments and microworlds. SOPHIE 3 included an evaluation strategy which compares students' performance with that of an expert circuit designer. The system reasoned about user problems and differences between novices and experts, and attributed these differences to bugs in the user's otherwise expert approach. SOPHIE research promoted user exploration as a way of improving the task relevance of a syllabus. Since either the system or the user could control the session, these systems can be said to have incorporated mixed initiative interaction.

While tutoring systems present a curriculum through which a student travels, help systems at the other extreme allow motivated users to ask a system questions. Many students are not motivated to follow a didactic lesson plan. Many computer users come to a new computer system with related experience. Their reason for using the new system is a problem they want to solve. It is preferable to center one's instruction on their problem.

4.3 Help Systems

Rather than motivating users to learn by a game or gradepoint, help, coach and critic systems work with them in productive situations. Systems which make computer assistance available in an actual work situation are termed help systems. Unlike ITS research, most research on help systems has concentrated on the quality and efficiency of delivery of information and has not yet extensively explored the use of Artificial Intelligence (AI)[3].

Help systems which support student goals can allow the student flexibility. They can also provide user support more easily than systems that give students simplified so-called training-wheel tutoring systems. Training-wheel tutoring systems protect students from a realistic work situation but must be left behind when a student is ready to begin a real or self-motivated project.

Many modern uses of computers involve mixed initiative, interactive solutions. Borenstein [3] performed behavioral experiments showing that help systems are more effective when they are available from within (integrated in) the computer program. His studies also show that help systems are improved when they

can give a user context dependent responses, basing information a user receives on the part of the computer program they are interacting with. Borenstein also observed that the quality of help text and its relevance to a situation are more important than other usability issues such as graphic design or simplicity of asking for the help. Content matters more than the form.

4.4 Coaching Systems

Teaching styles impact the students' role in their education tasks. Mastigalio [20] described teaching interaction scenarios as a continuum: from a tutor who prescribes what a user should do — to a coach who kibitzes with a student while trying to do something — to a critic who reviews work after it is completed. In these three teaching styles the point at which the system intervenes is varied relative to a student's design, construction and evaluation phases of work.

A computer aid for using or learning a body of knowledge may be called a coach when, like a human coach, the computer trains, reprimands, gives small syllabi of homework for a personal weakness, or tries to provide a needed idea or fact when appropriate. To respond to actual work, a system needs to learn from what a user does in an adaptive user model.

Zissos and Witten [47] built a prototype adaptive coaching system which could analyze transcripts of EMACS text editor usage as a critic (after a user session). ANACHIES, as it was called, could decide how to improve a person's use of EMACS editor commands. Their paper offers a pessimistic view of the feasibility of reacting as the user needs a coach. Their research convinced them that the computational requirements for using adaptive AI techniques in interactive applications are not feasible with the computers available in the foreseeable future, a pessimism which this research demonstrates to have been unwarranted.

5 Research in *User Interface & AI in Education*

Educational technology research can be thought of as attempts to balance trade–offs in interaction style, educational scenario, and educational goals. This section places this thesis's framework relative to educational technology approaches.

All frameworks for user interfaces in education have a way that the system is trying to teach (its educational approach), a way that the material is presented (the educational task), and a way that the student interacts with the system (its interaction style).

5.1 Educational Approach

Model of user: Teaching tools like programmed text have a static model of a user. **Our framework demonstrates the utility of a dynamic model of a user.**

Model of teaching: Current systems have a static model of teaching. **This thesis introduces a scenerio that changes the kind of information it gives a user based on a history of responses.**

Model of knowledge: Tasks in a classroom range from learning mechanistic (procedural) techniques to learning generally applicable (conceptual) approaches. **This thesis introduces a model of knowledge which concentrates on the syntactic and static semantic properties of a program.** This naturally focuses users on techniques. As a user gains more experience, a knowledge lattice allows the framework to bring conceptual background to bear as well. Explanations, sequencing and choice of text are built to present generally applicable approaches as well.

Model of development: The developmental versus behavioral distinction is brought up in [7]. as an overly ignored issue. Novice user models are not simply subsets of expert user models as Burton and Browns early systems assumed [42]. Our framework records idiosyncrasies in users knowledge.

Curriculum style: Exploratory environments, often called microworld simulations, stand in contrast to systems with tests or problem sets. The user interface scenario presented in this thesis gives the hands–on experience of a microworld in the "real world" supporting education with a rule system which augments teacher prescribed educational principles and facts.

Model of feedback motivation: Tone of discourse is crucial in motivating students [18]. Effects of tone ranging from strict discipline to positive cheering have been studied extensively as well. Courseware in our framework puts feedback in as positive a tone as possible, while still calling attention to any user input which cannot be part of a legal program.

5.2 Educational task

Model of domain: most teaching systems and formal teaching situations have a fixed or static set of things they are teaching. **This thesis introduces the model of a teaching domain which is designed expand as a students goals expand.**

Teaching setting: An educational scenario could be designed to be part of doing something productive or the education itself could be considered to be the productive work. This thesis's scenario presumes that a user is trying to to do useful work. The program could be a problem from a course or motivated more personally by the user.

Motivation of problem: Tutoring systems present problems to a user. The exploratory environment approach, on the other hand, responds to user's curiosity. Like help systems, our scenario responds to user exploration.

5.3 Interaction Style

Model of intervention: Different systems intervene at different points in the design process; This thesis's framework approach lies between the extremes of predefining a set of educational tasks of a tutor and waiting until a student is ready to have finished work scrutinized, as a critic system does [20]. Working with students *during* the design process is an attempt of mine to reduce user uncertainty without reducing user control.

Interactivity: Some automated teaching interactions have a batch interaction style, a question or analysis is submitted, time passes, and the computer responds. Don Gentner system [13], Anachies [47], is an example of a system which interacts with a batch scenario; analyzing a user state off–line before responding to a situation. Other scenarios such as the one in this thesis's are interactive and respond as a user does work.

Initiative: Some systems (like books) present information to a user but do not take input from users. Other systems (like a typewriter or text editor) only allow user input, . Still others allow mixed initiative, allowing either the user or computer to initiate communication. The approach taken in this thesis is mixed initiative.

.

These issues create a structure in which educational technology and research can be pursued. What roles should a computer take in an educational situation? How should techniques feasible in these situations be expanded? These questions motivate the development of the adaptive help framework.

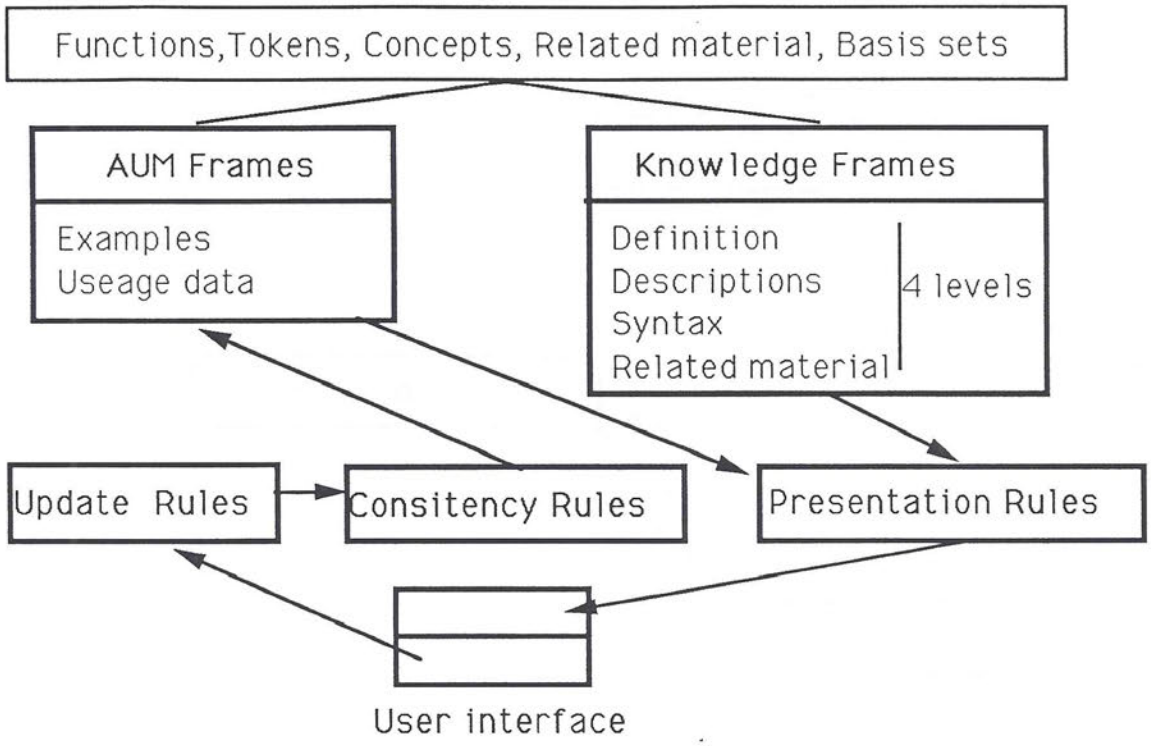

Figure 2: Knowledge Frames

6 Architecture

This section focuses on the structure and adaptive strategies which address the above issues.

I conceived the architecture to describe a framework for proactive interactive adaptive help. I built a system based on it to demonstrate that the scenario can be realized. The system is built as a set of interacting objects which work together using knowledge to guide it.

In this architecture, guiding knowledge is embodied in facts and rules. Parts of the knowledge are shown in (figure 2). They include:

- the adaptive user model,
- knowledge about how to build the adaptive user model,
- knowledge of teaching,
- and knowledge about the skill domain a user is trying to learn (eg. Lisp).

6.0.1 An Architecture Enabling Interactive Adaptive Help

Five interacting objects are sufficient to model a system which embodies this scenario (figure 3).

- The **FRAME** manages the editor, the menu, and screen real estate. It also dispatches input key and mouse events.
- The **READER** handles token level interpretation of what the user is typing.
- The **PARSER** handles the lexicon and builds the syntactic unit the user is typing. Both the READER and the parser send the AUM information about the user.
- **The AUM** is a model of the user which relies on the production system to make decisions based on the user model it has built and to decide how to advise the user.
- **The PRODUCTION-SYSTEM** uses rules to control help presentation and user model generation.

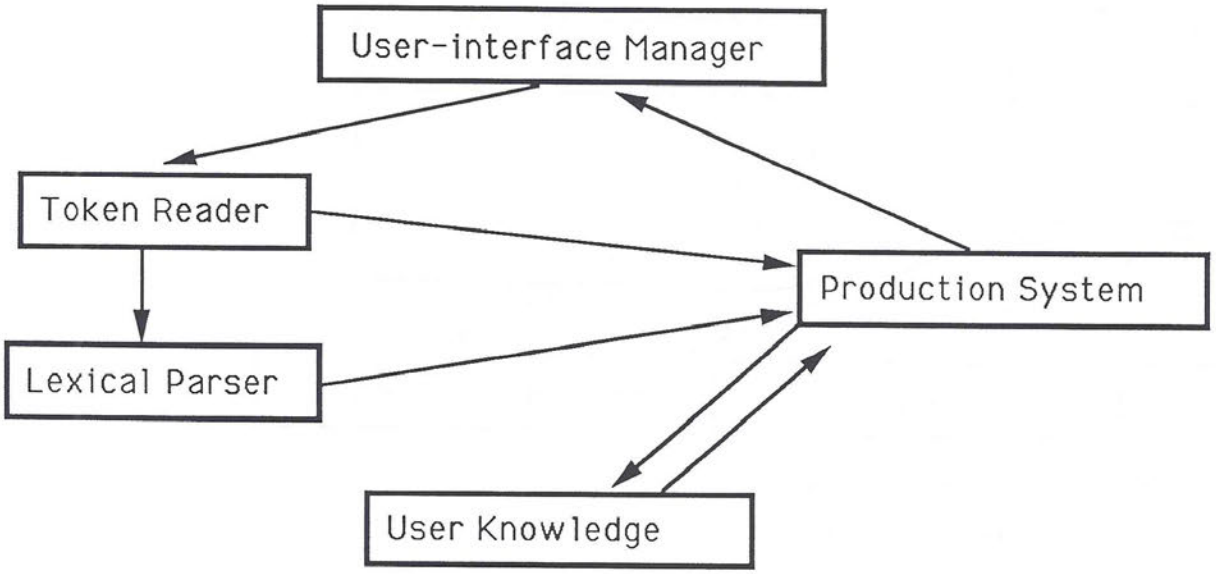

Figure 3: Architecture

6.1 Skill Domain and Teaching Knowledge

A user is trying to learn a *skill domain*, e.g. Lisp. The current scenario and system are designed for skill domains which require interpreted text input. Many operating system command languages, computer application interfaces, text markup languages and programming languages are interpreted text interfaces. Lisp was chosen as the initial domain to demonstrate for several reasons. Lisp is a domain for which other people have built intelligent tutoring tools, allowing our framework to be compared to their work. Lisp is extensible requiring a system which will work for a so–called open systems [14]. Lisp is a complex, open system that allows demonstration coaches to work in a realistic domain. Lisp has redundant ways of doing things which requires the system to act in ambiguous situations. For a second demonstration of my framework (UNIX) many of these issues are different, which shows that the framework can work for quite different domains.

6.2 Skill Domain Knowledge

A skill domain a user is trying to learn (e.g. writing a Lisp program) has a syntax; a set of things a user can type which are correct and interpretable.

As stated earlier, knowledge frames describe each domain language token, form, concept, and basis set. The system, like the standard UNIX facility: Yet Another Compiler Compiler (YACC), uses a state table to classify the character types that drive lexical analysis, and a formal language to drive actual parsing strategy. Unlike other parsers, the system parser adds knowledge to a user–model after each keystroke.

The domain language token parse can be modeled by a finite state automaton. Below is such a state transition table which can be used to tokenize Lisp. The table shows details of the actual transitions which drive READER. Characters are added to TOKEN? (an instance variable of READER), which holds the partially constructed token.

The token parse table (table 1) takes the current parser "state" (the column) and the current input character (the row) to look up the next state for the reader. For example, if the reader were in symbol state (column 3) and got an unused character (row 2) the reader would change to hlp state.

TOKEN? and the state of reader, are used to drive READER. A system built on this architecture

1 Form+ pls	2 delim- min	3 Symbol sym	4 Quote qte	5 string st	6 comment cmt	7 number num	8 macro Shp	9 help hlp	10 Character read
hlp	hlp	hlp	hlp	hlp	hlp	hlp	hlp	hlp	er unused char
hlp	hlp	sym	sym	st	cmt	hlp	hlp	hlp	sp special macro char
num	num	sym	Sym	st	cmt	num	hlp	min	num number
sym	sym	sym	Sym	st	cmt	Sym	hlp	min	chr character a- Z
Pls	pls	pls	Pls	st	cmt	pls	hlp	hlp	opn open parenthesis
min	min	min	hlp	st	cmt	min	hlp	min	cl close parenthesis
Hlp	qte	qte	qte	st	cmt	qte	Shp	hlp	qte '
st	st	st	st	min	cmt	st	hlp	hlp	st "
cmt	cmt	cmt	cmt	st	cmt	cmt	hlp	hlp	cmt ;
pls	min	min	hlp	st	min	min	hlp	hlp	ecm "end comment"
pls	min	min	MIN	st	cmt	min	hlp	hlp	blnk blank
Shp	Shp	sym	sym	st	cmt	hlp	hlp	hlp)	Shp #

Table 1: Model of a Token Parsing table for Lisp

builds a parse stack for pending parses. For the purpose of READER parses, a domain language is defined in a formal, context–sensitive syntax notation.

This notation is shown with the Lisp system key symbols in (figure 4).

A legal Sentence: S in this language consists of a string of symbols from the alphabet described in (figure 4).

S == *{A} : key symbols, control symbols and delimiters surrounded by a set of parentheses.

The language can be described as being made up of the alphabet:
A == {A,S,N,L,F,X,Q,FS,{,},[,],V,?,*,@XXXX}, where XXXX is any string of characters.

In practice, a new S is signaled by an open parenthesis, (the delimiters always come in pairs, except that all S-expressions end with].

The architecture uses an attribute grammar parser: each token type and each completed parse sends an **:action** message when recognized. Each key symbol in the notation has a method (function) associated with it which can cause an action during the parse.

Functions which the system parses are described in this notation. The simple example '(ABS N]) defines the absolute value function as requiring a number. A slightly more complicated syntax such as '(SETQ * A X]) requires 0 or one atom–"anything" pair.

The completed thesis will include a section which describes complex uses of this language, and an appendix which shows the COACH Lisp definition and the COACH UNIX definition.

6.3 Teaching Knowledge

Teaching Knowledge is embodied in rule sets. Present-Token, Present-Function and Present-Concept all suggest information to place on the help windows. These rule sets depend on the user model to make decisions about what to present. They are ordered to put the most important help on the blackboard first. The function rule sets, for example, have 8 such rules with the following mnemonically chosen names.

- **Losing-Ground,**
- **Out-of-Practice,**
- **Encourage-Exploration,**
- **Veto-Arglist,**

Machine Syntax

Key Symbols:

A Lisp Atom,

S defined Lisp Symbol,

N Number,

L List,

F Lisp Form; an S-expression

X any of the above types

Q check only parenthesis level
FS - Function Spec

Syntax Delimiters:

{ open a syntactic parse unit,

} close a syntactic parse unit,

[(open a clause,

]) close a clause,

Control Symbols

* next syntax part can occur 0 or more times,

? next syntax part can occur 0 or 1 time,

V At least one component of the next clause must occur at least once,

@ consider the following characters a symbol.

Logical Conjunction in a template is indicated by juxtaposition, disjunction is indicated by V {} syntax.

Figure 4: A Token Parsing table for Lisp

- **Veto-User-Example.**

The rule sets post their desired help on a blackboard. The present-help function presents as much of this help as it determines is appropriate for the user.

To get an idea of how these rules work, we examine Losing-Ground. The actual Lisp code the rule system interprets the following condition-action pair included as formal description:

```
(Define-Rule (Losing-Ground Present-Form) (form-used form-proposals)
  IF (NOT (SEND form-used Good Slope))

  THEN (PUSH User-Example form-proposals)
       (PUSH Example form-proposals)
)
```

The code implements the following concept: If the **slope** at which the user's performance is changing, and **good**, a cumulative index of performance are not positive then the person is doing poorly. In such a situation, the rule proposes a user example as well as system example for the confused student be placed on the **form-proposals** blackboard.

Besides pushing things onto the proposals, the system might use rules such as Veto-Arglist to take inappropriate information off the present blackboard:

```
(Define-Rule (Veto-Arglist Present-Form) (form-vetos)
  IF (NOT (>= (Get-Numeric-Property 'Lambda-List 'Best)
      (SEND *User* Good-Best)))

  THEN (PUSH ArgList form-vetos)
 )
```

This rule's code implements the following concept: If a user doesn't know how to read a lambda list, the arglist will be confusing and is not to be presented.

Together with state of the user model such rules model the reasoning in the framework.

6.3.1 Adaptive User Model

An AUM is a formal description of a user relative to a domain which tracks changes in the user's knowledge in that domain. The framework uses an explicit user model. Frames, facts and rules represent a model of the user and the skill domain the user is learning. The AUM is set a of user model frames [22] for syntactic and conceptual parts of the domain being coached. The user model frames record aspects of success histories for a user.

A simple rule set for creating and maintaining the user model consists of Update and Maintain-Consistency rules sets.

Update consists of rules with the following mnemonic names:

- Note-Success
- Note-Failure
- Was-Bad-but-Getting-Better
- Was-Good-but-Getting-Worse

- Best-and-Getting-Better
- Worst-and-Getting-Worse

Maintain-Consistency consists of rules with the following mnemonic names:

- Note-Used
- Maintain-Best
- Maintain-Worst
- Bound-Goodness-and-Best
- Bound-Goodness-and-Worst

User model frames are recorded for user-defined functions as they are created, to give help for these as well. A skill domain, like Lisp, for which the system helps a user, is represented in the system by syntactic and conceptual parts; tokens, forms, concepts and basis sets.

The completed Thesis will include a section which defines the rule syntax and describes exactly how to build new ones along with an appendix which shows all Lisp rules.

Skill domain parts

1. Language **Tokens**, keywords, and acceptable variable types for a skill domain are defined in a table with associated token methods. e.g. "CONS", "(".
2. Language **Forms** are defined in a syntax facts table. Plus, for example, is defined as **(PLUS * N)** . This table is extended by user defined functions.

1. Key **concepts** are important notions which are not codified by syntactic parts; e.g. EVAL.
2. **Required Knowledge** is the set of skill domain parts a user must be familiar with to use something (e.g. knowledge concerning EVAL, S-Expressions and Atoms is necessary to learn about CONS).
3. **Basis-sets** are sets of skill domain parts, all of which must be known to do a kind of task; e.g. simple-LISTS have a basis-set including: Atoms, S-expression, CONS, CAR, CDR.

AUM frames for skill domain parts have the following user model characteristics, or **slots:**

experience; How much it has been used?

latency; How long since the user has used this?

slope; How fast is a user learning or unlearning something?

examples; Examples of errors and corrections to those errors. When a user makes a mistake, the system records it. When the user is able to complete an instruction correctly, it stores that "fix" with the example. If the user later makes a syntactically isomorphic mistake, the system displays the familiar earlier example; e.g. (SETQ A (CONS 5) Expected ")" corrected to— (SETQ A (CONS A 5)).

Goodness; An overall goodness metric is also kept.

The lattice of relationships between skill domain parts (concepts, forms tokens, required knowledge, basis sets for a language, what the user is doing, and the state of the user model characteristics) is the basis for selecting user help.

The rule system uses this lattice of knowledge together with curricular knowledge in the form of rules to control user help. Reasoning and planning about how information interacts, the way the system updates the adaptive user model, even the system's adaptation algorithms which change the behavior of the system are kept in rules. By changing these rules, a researcher can tailor help for different skills and pedagogical ideas.

Each skill domain part has a help knowledge frame. These frames can include four levels of help defined for descriptions, syntax, and examples.

The help frame structure is a refinement of Rissland's taxonomy of examples [34]. Rissland described the importance of distinguishing examples for different kinds of users; this is extended to distinguish and include examples of use, syntax forms for what is legal, and description text telling how and when to use something.

The completed thesis will include a description of how to add user model frame slots.

6.4 Taxonomy of helpful information:

The help taxonomy consists of four levels of help, each of which can have syntax, example or description help.

Starter knowledge includes only simplified basic information. Confusing optional parameters and ways of using things are elided for novice or introductory help.

Reference knowledge is a more accurate description to familiarize users with standard uses of things.

Model knowledge is a completely descriptive explanation of what something is, and how to use it.

Anomalous knowledge is machine-level descriptions one might find in reference manuals.

6.5 AI Learning Strategies Used

As data is collected into user model frames, it changes the way presentation rules cause the system to respond to a user. To the extent that its responses change, the system learns.

In order to learn in real time, the system limits itself to opportunistic and simple hill climbing learning. The taxonomy of learning shown in Machine Learning [29] is used to organize the ways that the system can change its behavior.

Learning from examples is the practice of using specific solutions already accomplished in more complex situations. This technique is used by the system to collect and help for user defined functions. The system collects the function syntax by watching the user define it. The system collects function examples when the new function is used.

Learning by analogy is using knowledge collected in one situation in a different, analogous situation. The system uses its lattice of skill domain parts to access knowledge of related information a user knows in analogous situations. Analogy explains things in terms of skill domain parts the user already knows.

Learning from instruction is the practice of introducing knowledge a user "gives" into a system. The system uses learning from instruction to bring new function syntax for user defined functions into known syntax.

Learning by programming is simply the practice of having a developer add knowledge to a system. Expert systems and most state-of-the-art AI in education systems rely exclusively on developer modification to change response behavior. The Lisp Tutor [1], and the Lisp Critic [20] improve their performance in this way. The system is designed to allow a researcher to easily add facts and rules improving the adaptive user model system without writing Lisp code. Observations of students using the system give the researcher ideas of how to change the way the system treats a user in different situations. These ideas are put into additions and changes in presentation text or presentation rules.

The system premiers a collection of technology enabling research on real time adaptive help in an interactive programming environment. It includes the ability to classify syntax and static semantics issues embodied in the keystrokes of a user entering a program. It uses rule sets to update a fine grained user model and to analyze this model and present information to the user in real time.

7 An AUM Help Development Shell

COACH's structure is designed to allow a courseware designer to change the skill domain information presentation approach or adaptation strategy with minimum effort. To demonstrate and evaluate this capability, I asked a talented 17 year old high school student without programming experience to attempt to adapt the system to teach UNIX. In his 10 week internship the student learned enough UNIX to identify 20 key UNIX commands, wrote help text for these commands, defined delimiter and token types needed for the system to parse UNIX commands, and wrote syntax definitions for all identified UNIX commands.

I wrote two new Flavor methods (functions) to make carriage return into delimiter and to make change the "X" anything token. I also changed the character parse table altering the delimiter from ")" to carriage return and eliminate the ";" for comments. At the time, the system only ran on the Symbolics computer. A command caller which interfaced to a UNIX workstation over a telnet connection was proposed. Waiting for the IBM PC-RT version, a more logical workstation to use as a UNIX front end made sense.

The courseware developers' process of converting the system to teach in a new task domain requires the following steps:

1. Identify a task domain. The system is designed to work with textual interpreted interfaces. To have actual output from the interpreter it has to to be able to communicate with the Symbolics or RT running MACH, which the system runs on.
2. Identify delimiters and other token types which the system does not have.
3. Write token handling methods for the domain's token types not already supported.
4. Change the token table for parsing delimiters.
5. Identify commands in the skill domain for which help will initially be made available.
6. Write skill domain syntax in the system syntax language.

A primitive COACH will now exist for the skill domain. The system will be able to check user syntax, look for undefined functions, variables, learn about user examples and add help for new and system functions. It can also decrease and increase help and change its levels of help. It doesn't know about relationships between syntactic units, concepts, basis sets or required knowledge. Identifying the relationships between parts of the domain allows the developer to add deeper knowledge about the skill domain. This can be accomplished with the following steps:

1. Identify Skill domain concepts.
2. Identify basis sets in the skill domain.
3. Identify required knowledge for skill domain parts.
4. Write description, syntax and example text for three levels of each skill domain part.

The completed thesis will include details of adapting the system for the UNIX domain.

8 Implementation Status

The implementation is written in Flavors. Versions of it run on Symbolics Genera 6 through 8 [23]. A conversion procedure allows the Symbolics version to identically run on Mach Lisp with CLOS on an IBM PC-RT. On a Symbolics 3640, with eight megabytes of memory, the system is able to keep up with user's typing for reasonably sized functions. Performance problems occur when a user leaves mistakes unfixed. In these cases, the system must attempt to re-parse all forms in a routine every time a user types a character.

The Common Lisp system has courseware for approximately 40 common Lisp functions. It automatically constructs help for all other functions by accessing system argument list data and accumulating examples from users.

The UNIX system has courseware for 20 popular UNIX commands. Its output has yet to be connected to a UNIX shell.

The completed thesis will expand on these descriptions.

Question	Non-adaptive Mean	Adaptive Mean	p–value
How often do you look at the help screen while solving a problem?	3.16	3.91	0.04
How useful is the help screen?	2.44	3.20	0.11
Is the system better than a normal environment?	3.31	3.4 9	0.70
Comfort using the system?	1.97	2.97	0.05
Comfort programming in Lisp?	1.90	3.36	0.01

Table 2: Data from questions 1-5 on comment sheets.

9 Evaluation

In a five session user study and Lisp course, the system, which gives automatic adaptive help was found to improve both performance and perceived usability over a version which only offered non-adaptive, user requested help. In this study, significant differences were found in groups which had Adaptive Automatic help versus subjects who only had mouse selectable help.

The amount of knowledge that the students were exposed to included complex data structures, recursion etc., possibly 2/3 of what students of a full semester Lisp course would be exposed to. The terse nature of the tutorial masked the quantity of information.

The goal of requiring them to use the help system to solve their problems was achieved. The data demonstrate differences in self assessment and perceived performance between users of the adaptive automated help system as compared to users with manual help. Even though large differences were found between the groups, the non-adaptive group performed extremely well considering the amount of time that they had (table 2). Even the non-adaptive users believed their user interface was a significant improvement over the usual tools that are available.

The pressure of learning so much Lisp in such a short time without any teacher help was a strain for all subjects. Although all students did complete the study, one of the manual help group subjects required extensive persuasion to continue the study after the third day.

The students wanted feedback from the experimenters. When they were told that they would not receive help they felt discouraged. A pilot study in 1988 showed that the pressure of the amount of material to be learned was greater when presented in a non–self–paced manner, influencing this study to be self–paced.

While data collected from the comment sheets from the two groups shows indistinguishable motivation and self described performance between them, the automatic adaptive help group utilized all available materials (the automatic help, the Lisp Tutorial, and user requested help), felt more comfortable with Lisp, had a higher morale and **wrote five times as many functions on average** than the group with manual help.

While both groups had the same access to the tutorial and on-line help, the manual help subjects were most likely to look only at menu selected help, in contrast to the automatic adaptive help group, which used tutorial materials as well as asking for help (table]3). The reason for this could be a lower moral among the manual help subjects; they missed the student/teacher interaction of a classroom environment. Although the self–perceived motivation of the manual help group was not much different than the other group, the data clearly show that the manual help group had less confidence in the programming environment and Lisp.

The automatic adaptive help system succeeds in raising moral, and motivation, which in turn leads to an attitude that is more conducive to learning.

The completed thesis will include data and detailed analysis of user studies of the framework. Appendices will show materials used in the study.

Learning Materials Used	Manual	Automatic
Asks for Help	6	6
Uses presented Help	0	6
Refers to Tutorial	1	6
Uses Trial and Error	1	0

Table 3: Number of people using different methods to solve problems.

10 Future Directions

10.1 Future Research Goals

As a testbed for coaching ideas, the system positions a researcher for a host of follow on investigations. Below is an outline for a few research directions which could be productive continuations of the work undertaken for this thesis:

1. Exploring details of how various adaptive mechanisms impact users.
2. Using the adaptive paradigm to make agents to save users from rote work.
3. Using the adaptive paradigm in graphical interfaces with menus, drawing and/or other non-textual interactions.
4. Integration of different help media like video graphics or audio into the system.
5. Demonstrating the adaptive paradigm for use with tutorial curriculum based educational situations.

Implementation work which will make the system more available and usable are somewhat separate from research objectives. These directions are exciting because they hold possibilities for making the technology widely useful and available. The following implementation work could make my work more widely available.

1. Integrating adaptive help technology with other program development tools in a programming environment.
2. Making the system run on other platforms.

The system is more than a specific implementation of an adaptive paradigm in a help system. It is a shell which allows researchers to fill in the details of when, how, where and why adaptation improves help applications.

10.2 Future *Research*

- How do **adaptive mechanisms impact** the users? Experiments with users motivated selection of adaptive strategies. A researcher can change strategy rules to change the system's adaptive strategies. Studies concerning which strategies are best could be set up to address issues in education, cognitive psychology or cognitive science. When first exposed to a performance help level, is it better to show a user examples, syntax and description, or would it be better to simply focus on an example? Should the novice be shown as much information as possible when just getting started? Presently the system shows a user relevant things it thinks the user should understand but has not yet digested. These questions should, in fact, be expanded to give insight into issues of cognitive models. This framework is uniquely positioned to explore such issues.

- The use of **agents** in the adaptive paradigm; Should the computer tell a user how to do something, or do it for them?

  The simplest use of the adaptive paradigm is the advisory one; the framework tells a user how to do things. In an agent paradigm, on the other hand, if the computer knows something like an open parenthesis needs to be input, it types it. In such a paradigm, when the computer sees some way of simplifying what a user needs to do to accomplish something, the computer builds a primitive to do that to come to the aid of the user. The computer is building a private, helpful set of things it does for the user, a language the user and the computer both know. To the extent that the things the computer tries to do to the user's program are what the user really needs and wants, and to the extent that either the user can ask for it or the computer can recognize the need for it easily, the agent is helpful.

- How would the adaptive paradigm work for **graphical interactions**?

  The framework was created for a textual domain, but it could be adapted to give help in a graphical domain. Brad Myers' [25] seminal automated interface construction system, Peridot, addresses issues in adaptive graphical help systems. Peridot assumes certain graphical goals of a user wanting to build a graphical interface. It uses mixed initiative to test its hypothesis in designing a user interface. An explicit user model keeps track of what kinds of user interface techniques a user has applied and familiarity with the spatial tools. Such a framework could be modelled after this thesis's framework. A system in which the graphical language was defined in a systematic way as a visual language using Selker's elements of visual language [39] could parse visual graphic commands as COACH parses textual commands. It could give help in textual terms. More interesting is the idea of giving graphical advice. This can be done using a number of techniques: an advisor mouse like Peridot's could draw, light things up, and blink in ways that are distinguishable.

- Integrating **multiple** help **media** into the adaptive help paradigm.

  Video segments could replace text for explaining things. Hardware and demonstrations of integrating computers and video exist and can be harnessed [12]. The Anchored Instruction researchers at Vanderbilt University have proposed to use COACH to build a multimedia help system for their research.

- Integrating **tutorial curricula** with the adaptive paradigm.

  This thesis demonstrates an adaptive user model-based teaching aid which follows the goal-directed user assumption. While most of our lives are spent trying to do things with our own goals, we all go through a period of schooling, a time when others motivate our goals. The coaching scenario supports a user's goals leaving the syllabus and motivation of goals up to a user or a human teacher.

  The framework could be extended to include more directed teaching materials as demonstrated in systems like the Lisp tutor [30]. Such a system would have curricular goals, a syllabus and a set of ways of evaluating a user relative to these goals. In addition to giving programmed lessons as other tutors do, such a system would be able to follow and help users in their programming, even when they are not doing exactly what the system wants them to do.

10.3 Future *System Development*

As well as being a research system, COACH can be used in real work. Below I briefly outline two kinds of work which would make the scenario more useful and available to users.

- Integrating adaptive help into Standard User Work Environments.

Anyone using the Hemlock Emacs-like editor can already integrate this thesis's framework into his or her Lisp programming work. Several integration projects could improve the framework's widespread use:

  - User interface environment aids

    Rules could be added to add agents which look for spelling errors, search for variables, uses in other functions, function definitions, data type uses, etc.

  - Efficiency improvements.

    The courseware facts and user interface grammars could be moved into improved speed-of-access data structures for increased performance on large user languages.

- Exporting COACH for use on different computers.

  An automated mechanism exists for supporting this port as the system evolves. Since the Mach Lisp tools can be used on non-Mach machines, the system should be able to run on Common Lisp implementations which support the CLX interface to X11 without modification. It would be nice to make transport systems to Common Lisp environments which run on other machines. This might require interfacing the system to other text editors and window systems. I am also exploring porting a COACH system to a personal computer platform.

11 Bibliography

References

[1] J. R. Anderson A. T. Corbett. Feedback timing and student control in the lisp intelligent tutoring system. In *Proceedings of The International Conference on Artificial Intelligence*, pages 64–72. IOS, 1989.

[2] S. M. Alexander and V.Jagannathan. Advisory system for control chart selection. *Computer And Industrial Engineering*, 10(3), 1986.

[3] N. S. Borenstein. *The Design and Evaluation of On-line Help Systems.* PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[4] R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2, 1978.

[5] R. Burton. *Intelligent Tutoring Systems*, chapter 4. Academic Press, 1982.

[6] R. L. Campbell. Developmental levels and scenarios for smalltalk programming. Technical Report RC68222, IBM T. J. Watson Research Center, Yorktown Heights, NY, December 1989.

[7] R. L. Campbell. Online assistance: Conceptual issues. Technical Report RC15407, IBM T. J. Watson Research Center, Yorktown Heights, NY, December 1990.

[8] J. G. Carbonell. Computer models of human personality traits. Technical report, Computer Science Department, Carnegie-Mellon University, Pittsburgh PA, 1979.

[9] J. Carroll and A. Aaronson. Learning by doing with simulated intelligent help. *CACM*, 31(9), 1988.

[10] T. O. Ellis and W. L. Sibley. The grail project. *Spring Joint Computer Conference, Boston, MA*, 1966. verbal and film presentation.

[11] G. Fischer, A. Lemke, and T. Schwab. Knowledge-based help systems. In *CHI Proceedings*, 1985.

[12] S. Gano. Movie manual. Technical report, MIT Media Lab, Cambridge MA, 1982.

[13] D. R. Gentner. A tutor based on active schemas. *Computational Intelligence*, 2, 1986.

[14] C. Hewitt. The challenge of open systems. *Byte Magazine*, pages 223–342, April 1985.

[15] R. C. Houghton. On-line help systems: A conspectus. *CACM*, 27(2), 1984.

[16] K. Lawrence. Artificial intelligence in the man/machine interface. *Data Processing*, 1:231–236, 1984.

[17] J. Mackinlay. *Automatic Design of Graphical Presentations.* PhD thesis, Computer Science Department, Stanford University, Stanford CA, 1986.

[18] T. W. Malone and M. R. Lepper. *Making Learning Fun: A Taxonomy of Intrinsic Motivation For Learning.* Erlbaum, NJ, 1987.

[19] Z. Manna. *Mathematical Theory of Computation*, chapter 5-3. McGraw-Hill, 1972.

[20] T. Mastigolio. Tutors coaches and critics. Technical report, Computer Science Department, University of Colorodo, Boulder CO, 1989.

[21] E. Mays, Chidanand Apte, James Griesmer, and John Kastner. Experience with k-rep: An object-centered knowledge representation. In *The Fourth Conference on Artificial Intelligence Applications, Proceedings.* IEEE Computer Society Press, 1988.

[22] M. Minsky. Frames. Technical report, AI Laboratory, MIT, Cambridge MA, 1976.

[23] D. Moon. *Users Guide To Symbolics Computers.* Symbolics Inc., 1987.

[24] N. M. Morris and W. B. Rouse. Adaptive aiding for human-computer control: Experimental studies of. Technical Report AAMRL-TR-86-005, Armstrong Medical Research Laboratory, Wright-Patterson Air Force Base, OH, 1986.

[25] B. A. Myers. Visual programming, programming by example, and program visualization: A taxonomy. In *CHI '86 Proceedings*, pages 59–66, 1986.

[26] P. Pirolli. A cognitive model and computer tutor for programming recursion. *Human-Computer Interaction*, 2, 1986.

[27] J.S.Brown R. Burton. *Intelligent Tutoring Systems*, chapter 2. Academic Press, 1982.

[28] Jr. R. F. Grise. *ANGEL: A Pleasant User-Interface For An Interactive Computing Environment.* PhD thesis, Cybernetic Systems Department, San Jose University, San Jose, CA, 1986.

[29] T. M. Mitchell R. S. Michalski, J. G. Carbonell. *Machine Learning: An Artificial Intelligence Approach.* Tioga Publishing Company, 1983.

[30] B. J. Reiser, J. R. Anderson, and R. G. Farrell. Dynamic student modeling in an intelligent tutor for lisp programming. *IJCAI*, 1, 1985.

[31] P. Reisner. Human computer interaction: What is it and what research is needed. Technical Report RJ5308, IBM Almaden Research Center, Almaden CA, 1986.

[32] M. E. Revesman. *Validation and Application of A Model of Human Decision Making For.* PhD thesis, Industrial Engineering and Operations Research, Virginia Polytechnic, 1983.

[33] E. Rich. Users are individuals: Individualizing user models. *Int. J. Man-Machine Studies*, (18):199–214, 1983.

[34] E. Risland. Understanding understanding mathematics. *Int. J. Man-Machine Studies*, 2(4), 1978.

[35] J. Schofield, D. Evans-Rhodes, and B. Huber. Artificial intelligence in the classroom: The impact of a computer-based tutor on teachers and students. *Social Science Computer Review*, 1990.

[36] O. Selfridge. Personal communication, 1985.

[37] T. Selker. Cognitive adaptive computer help; a user manual. Technical report, IBM T. J. Watson Research Center, Yorktown NY, 1988.

[38] T. Selker. Cognitive adaptive computer help (coach). In *Proceedings of The International Conference on Artificial Intelligence*, pages 25–34. IOS, 1989.

[39] T. Selker and L. Koved. Elements of visual language. *1988 IEEE Workshop On Visual Languages*, October 1988.

[40] T. Selker, C. Wolf, and L. Koved. A framework for comparing systems with visual interfaces. In *Interact '87 Proceedings*. North Holland Amsterdam, September 1987.

[41] D. Sleeman and J.S.Brown, editors. *Intelligent Tutoring Systems*. Academic Press, 1982.

[42] J. C. Spohrer and E. Soloway. Alternatives to construct-based programming misconceptions. *CHI '86 Proceedings*, 1985.

[43] P. Suppes. Some theoretical models for mathematics learning. *Journal of Research and Development in Education*, (1):4–22, 1967.

[44] L. Massinter W. Teitelman. The interlisp programming environment. *Computer*, 14(4):25–34, 1981.

[45] L. Weiss. Conceptual model of an intelligent help system. Technical Report DDC/LW-15, ESPREE, May 1987.

[46] J. Whiteside and D. Wixon. Improving human-computer interaction: A quest for cognitive science. Technical report, Digital Equipment Corporation, Maynard MA, 1986.

[47] A. Y. Zissos and I.H. Witten. User modeling for a computer coach: A case study. *Int. J. Man-Machine Studies*, (23), 1985.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598