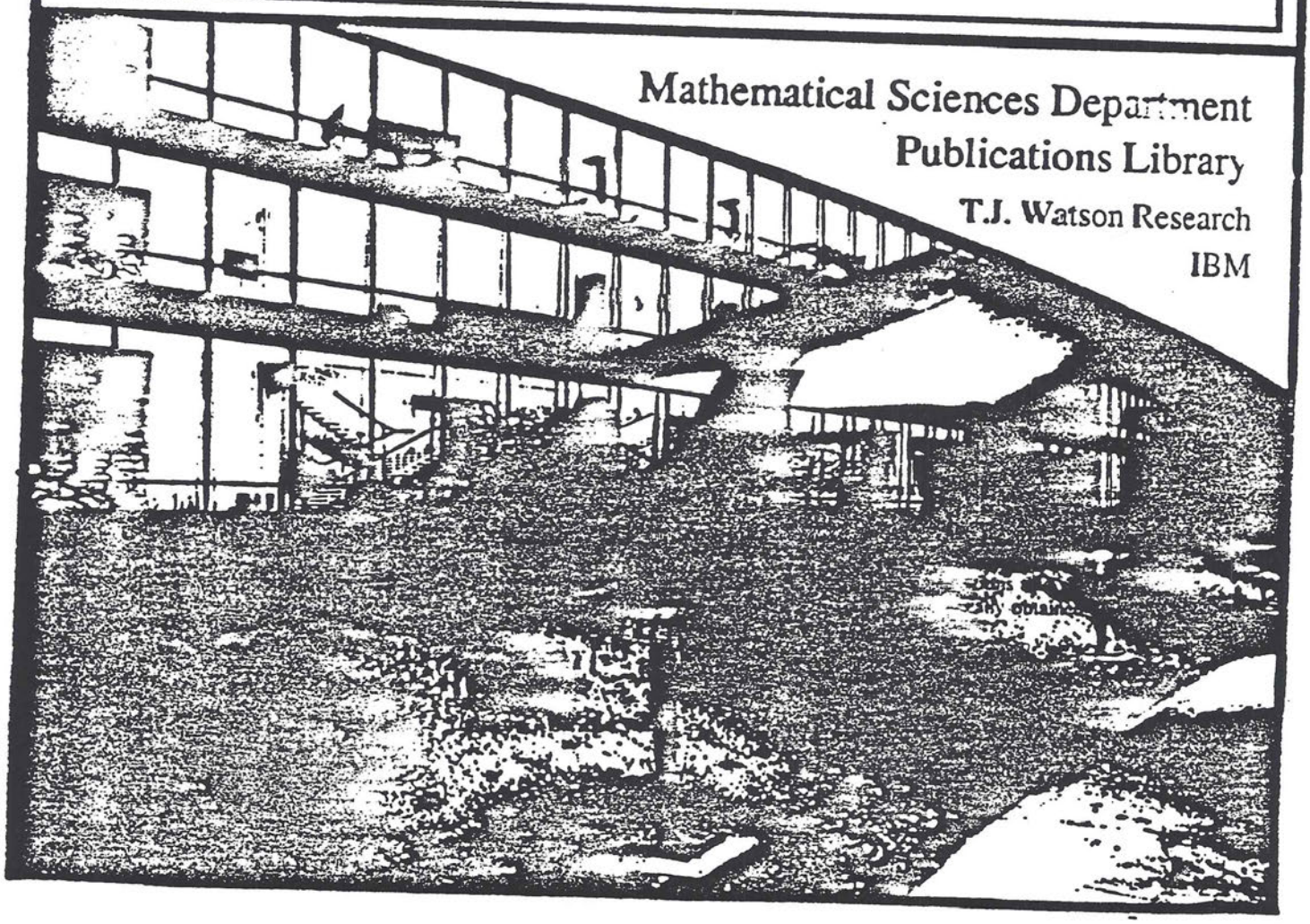
RC 14130 (#63295) 10/27/88
Computer Science 8 pages

Research Report

The COgnitive Adaptive Computer Help (COACH) Interface

Ted Selker

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



RC 14130 (#63295) 10/27/88
Computer Science 8 pages

Research Report

The COgnitive Adaptive Computer Help (COACH) Interface

Ted Selker

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

The COgnitive Adaptive Computer Help (COACH) Interface

Ted Selker
Thomas J. Watson Research Center, IBM
P.O.Box 218
Yorktown Heights, N.Y. 10598
Selker@IBM.COM
914-945-2217

Keywords: *adaptive, artificial intelligence, machine learning, production system, user model,intelligent tutoring system.*

Abstract

This paper describes COACH, a system designed to mimic the kind of empathetic help a human coach can provide. The system builds an adaptive user model to attempt to understand a user's needs. COACH watches a programmer's progress, advising with syntax, examples and language underpinnings as needed. COACH ceases advising in situations where its user has demonstrated expertise. COACH text reappears when that user once again needs help. Whether giving help for user-initiated explanations, or acting as an advisor for COACH-initiated explanations, COACH presents information at the level which a user has shown competence.

COACH is an architecture for experimenting with adaptive user models. The language COACHed on, the way COACH adapts and the pedagogical paradigm can be changed by writing a language definition table, courseware facts and COACH learning rules.

The system runs on The Symbolics and IBM PC-RT computers. GOACH has been set up to teach Common Lisp COACH. User studies are being run. COACH is also being set up to run for UNIX, and other interface languages.

Introduction

Many new computer users have personal goals and are not always motivated to follow a didactic lesson plan. Behavioral studies show that help systems are more effective when they are available from within the computer program (integrated) and help response is based on where the user is in the program (context-dependent) [1]. Other research in Intelligent Tutoring Systems [2, 4, 7, 9], motivate our work on systems providing integrated, context-dependent, adaptive coaching.

Burton and Brown [8] built mixed initiative electrical circuit trouble-shooting coaches (Sophie1,2 and 3). Sophie 3 compared students to a model of an expert circuit designer. The system reasoned about user problems, assuming the difference between novices and experts to be attributed to bugs in user's otherwise expert approach. In their work, user exploration was promoted as a way of improving task relevance of a syllabus.

COgnitive Adaptive Computer Help (COACH) expands on Burton and Brown's approach by being designed for use with large domains and expecting different needs of novices and experts to come out of their inherently different user models. We believe a coaching approach is useful for open systems where the goal of the user is not assumed. The key contributions of COACH are the application of an *Adaptive User Model* to the *advisory coaching paradigm* in a system which can be reconfigured as a *testbed* for experiments in Intelligent Tutoring.

COACH analyses a user's syntactic and static semantic understanding of the system being used. It integrates this analysis into possible coaching responses. As a user needs help, the system advises (coaches) them. User expertise in COACH is relative to the tools the user is expert with. A user may be expert with lists but lousy with arrays, COACH's help adapts giving detailed help for arrays but not lists.

A system could help a user as an agent performing tasks in the users behalf or as an advisor merely telling the user about things the user might need to do. [6]. COACH's *paradigm* is *advisory* since it does not type for the user; it does not lock the keyboard; it never types where a user could *type*; help information is provided in a separate window pane.

The COACH Interaction Paradigm; How users learn

The COACH interaction paradigm consists of a coaching process advising a user as the user types a program. Coaching text can be called with user menu selection, but like real coaches help often is provided by the coach by its own initiative.

Based on user experience and expertise, COACH rules decide whether to show description, syntax, or examples. If a person has read descriptions many times the system expects that the person does not need to read the same description again. If the person has improved, then the syntax level will change and the examples will change. If the person has seen the syntax and used it accurately, then the system will not show the syntax again. The examples do not change until the person has adequately mastered the level for which the syntax has been described. This example of control reasoning is adaptive and described by rules.

COACH includes an incremental interpreter with backup. Each new character a user types changes coach's expectations for what can be typed next. It uses an adaptive user model to give advice appropriate to the users expertise and experience.

COACH's user model is explicit. COACH uses frames, facts, rules and learning technology to represent a model of the user and the language with which being

interacted with. COACH learns, picking up examples, and keeps track of what a person does right and wrong.

A person types into a input "listener" window. The system responds with advice in the help and token help windows, shows results in an "effects" window and allows other interaction through menus

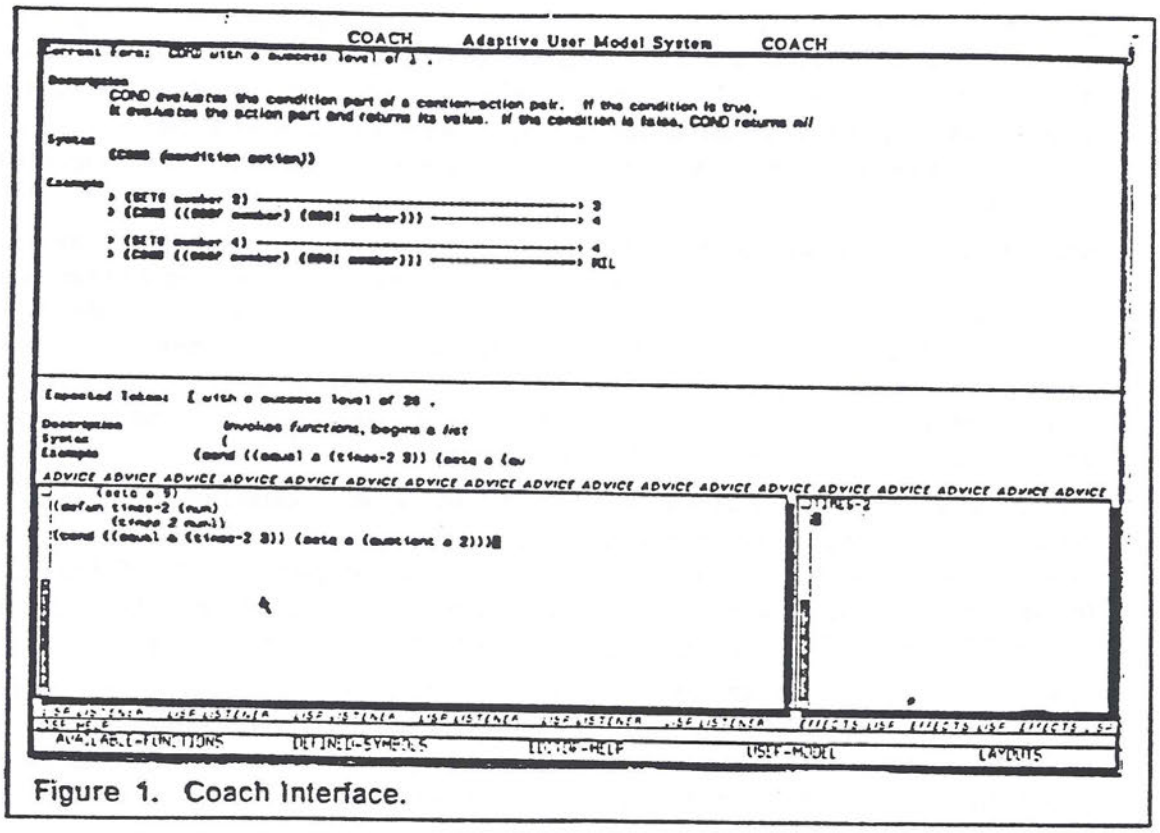


Figure 1. Coach Interface.

The listener window shows the user's work. This window is an interpreter of the language the user is trying to write in. It uses the popular Emacs commands to allow a person to edit input.

The "effects" output window shows the effects of the input the user typed. A help menu can be displayed permenantly or poped up at any time.
This is the way a user explicitly asks for help.

The COACH help window displays information designed for the user's current needs. It shows a user legal syntax, pertinent examples and helpful concepts.

The token help window is closest to the listener pane, the token window formats specific, local information such as misspelled variable names and word or character level problems. Like a syntax-directed editor this part of the screen can walk a user through syntax. The kind of keyword and variable spelling corrections Teitelman's DWIM system [10] could handle are dealt with in this area. Farther away from the listener window COACH provides general information. Examples of the function that the user is trying to use, descriptions of what it is good for and information about how it relates to other things this user knows are provided, at the level COACH finds appropriate for this user.

COACH treats a person that knows all about some part of the language, and little about another part of the language, as a novice only in the parts where little is known.

For example, if a person knows about COND and is beginning to use IF: since there is a relationship between COND and IF through the concept of predicate, the system can show that the predicate COND, which is already known is similar to IF. Similarly, when a person has trouble using COND, and already knows how to use IF, the system should extend the IF experience to COND, showing the relevance of IF experience. If the person has already mastered something which is at one level, and is learning something similar, COACH recognizes that less instruction will be needed.

COACH Architecture; how Coach works

An adaptive user model drives the COACH system. COACH rule and fact knowledge is structured to permit real time time reasoning on a user model even in a complex language like Common Lisp. COACH is built as a set of interacting objects which work together using knowledge rules and knowledge from a user's current user model to help that user.

Adaptive User Model

An adaptive User Model is a formal description of a user relative to some domain which changes to improve or track a user's change in knowledge in that domain. COACH's user model consists of connectivity of relationships which are known for a specific user, examples generated and characteristics of experience, expertise which the user infers or the user has demonstrated. This architecture is used to create a user model for a language, providing help on the *user factors*:

1. language **tokens** e.g. "CONS", "(".
2. language **forms** e.g. (CONS arg1 arg2).
3. **basis-sets**; *sets* of factors, all of which must be known to do a kind of task
   e.g. (Basis-set simple-LISTS Atoms S-expression CONS CAR CDR).
4. **required-knowledge**; sets of *factors* all of which must be known to use an element in a language, e.g.(Required-Knowledge CONS S-Expressions Atoms)and
5. key **concepts**; fundamental ideas in a language, e.g. EVAL.
6. help **menu selection**

The system has a list of concepts that a person needs to know. COACH'S first tested domain is Common LISP. If a person is typing LISP, the system notices: Are the tokens being typed correctly? Are keywords typed after the open parenthesis in an S-expression? Are forms being made correctly? If the function CONS is being used, has the person shown themselves to know required knowledge? The system has concept information in the rule base describing concepts ranging from EVAL and QUOTE to what a program is. Concepts are described relative to what they are used for. When a user needs to use a concept and is not yet proficiency with it COACH will consider including it in help information.

COACH's user model records the following *user model characteristics* for each user factor:

1. **number**: The number of uses that have been made of each user factor.
2. **best**: The most sophisticated use a user has made of a user factor.
3. **experience**: How much it has been used.
4. **latency**: How long since the user has used this.
5. **slope** How fast a user is learning or unlearning something. How much the sophistication of use been changing with usage and over time.
6. **examples**: Examples of errors and fixes to those errors, When a person makes a mistake, the system records that as an example; when the person is able to

complete an instruction correctly, it stores that "fix" with that example. Then, if the user later makes a syntactically isomorphic mistake, the system displays the familiar example. e.g. (SETQ A (CONS 5) Expected ")" corrected to— (SETQ A (CONS A 5)) GoodnessAn overall metric is also kept.

Slope and examples made a large impact on users in our early system called Lexical Expression Language Parser Help (LELP HELP) in 1984.

When it has the opportunity, the system adds new user-defined functions to its table and adds help information to the newly defined functions. Many characteristics that are recorded are actually scalars to help the system react at the speed that a person is typing.

Each user factor can have four levels of help defined for descriptions, syntax, and examples, The levels are an extension to Edwina Risland's taxonomy of examples [5] to descriptions and syntax:
help taxonomy:

1. **Starter** knowledge is used as a novice level, only simplified basic information is provided.,
2. **reference** knowledge, a more accurate description which tries to lead users into more complexity.
3. **model** knowledge, is completely descriptive explanation of what something is, and how to use it.
4. **anomalous** knowledge is machine readable syntax, the very complicated, idiosyncratic examples and descriptions that one might find in reference manuals.

COACH Knowledge

The following COACH knowledge drives the system and is re-definable by a researcher or designer: . The language COACH is helping a user with is defined by *language knowledge*:

1. **Token definitions** for the language a user will be coached on are defined in a table with associated token methods.
2. **Syntax definitions** for the language are defined in a syntax language: e.g.. (PLUS * N)
3. **Help information** for a language we call courseware. Courseware is a set of facts.
   - Courseware about the language the user is learning is separated as much as possible from courseware about learning.
   - Concepts in the domain that the system is teaching are also kept as a list of facts.
   - The relationships between concepts and syntax definitions are kept as facts.

The way COACH presents courseware is defined by **curricular knowledge:**

- **Reasoning and planning rules** about how information interacts are kept as rules.

The way COACH change the adaptive user model is defined by **adaptive knowledge:**

- **learning rules;** rules describing how to change the behavior of the system when a person performs an action.

These elements provide a system that can be tailored for different languages and to test out Intelligent Tutoring ideas.

COACH structure

The Coach shell consists of five objects which interact together to make the system.

- The **coach-frame** manages the Emacs-like editor, the menu, screen real estate and dispatches input key and mouse events.
- The **coach-reader** handles token level interpretation of what the person is typing.
- The **coach-parser** handles the lexicon, and builds the syntactic unit the person is typing. Both the COACH reader and the parser serd the COACH user information about how the user is doing.
- **coach-user** relies on PS, the production system, to make decisions based on the information it is gathering and the user model it has built to decide how to advise the user.
- **coach-ps**, the production system, it reasons.

COACH Learning Strategies; how COACH learns.

In order to learn in real time COACH limits itself to opportunistic and simple hill climbing learning. By reference to the taxonomy of learning shown in Machine Learning [3], the ways that COACH uses learning may be classified:

- The system uses learning by analogy by looking for and explains in terms of user factors that the user already knows. It reinforces successes, for future use. The network of relationships between user factors, language concepts required knowledge and user defined factors is the basis of the substrate upon which these analogic relationships are determined.
- Learning from instruction is another method COACH uses. The parser takes statistics on the style of interaction to which the user responds to and the level of the user.
- Learning by programming is used by the designer to incorporate into COACH newly acquired knowledge of how to change the way the system treats a person in different situations. This is similar to the syllabus notions that have been used in classical intelligent tutoring systems.
- Learning from examples is used by the system for including help for user defined functions. The system generates reference syntax with no optional arguments, anomalous syntax with machine readable syntax. The system records examples of use of new functions, classifying them by the sophistication of syntax for use in demonstrating use of the function. The function author is also free to generate descriptions, key examples and descriptive syntax help.

Discussion

A pilot user study with professional, Lisp nieve programmers, allowed us several encouraging observations. When working with COACH people experimented, read help text a lot more and finished problem sets faster than when the same help information was only available in text. Post exercise sessions tests did not show a dependence on COACH help to retrieve information. We are running more user studies to document the effects of coach on students in detail.

COACH is written in Flavors. Versions of it run on Symbolics Genera 6 through 7.2 on Mach Lisp with CLOS on an IBM PC-RT. The Common Lisp implementation has carefully crafted courseware for common LISP functions, approximately 40. It automatically constructs help for all other functions defined on the interpreter which the system evaluates its input. On the 3645, with four megabytes of memory, the system

is able to keep up with the person's typing Additional Language elements do not slow down the system. performance problems occurs when a user insists on leaving mistakes unfixed. In these cases, the system must attempt to re-parse all forms in a routine every time a user types a character.

SUMMARY

COgnitive Adaptive Computer Help is an architecture for including user background in computer-human interactions. It uses learning and reasoning to direct its role as an advisor/coach of computer users.

COACH as a front end allows users to forego the requirement of reading manuals before using a system. COACH assumes a directed user; people sit down at a computer with a goal in mind, something that they want to do, and they have experience which they are bringing to bear on this situation. As systems become more complex, people will master parts of it useful for a specific task at a specific time. Can a cohesive syllabus even be designed that covers 25,000 commands to do everything from make noise to control page faults in a system. COACH designed to have a minimum of syllabus and a maximum of interaction variance.

COACH is segmented into the COACH-frame with its user-interface package, the LISP parser editor which has the tokenized Lisp parser, the knowledge base for presenting Lisp and teaching knowledge to the user, the reasoning system which controls the system, the set of rules which run the system, and the learning system. The adaptive user-model is stored as a separate file for each individual user.

COACH is useful to novice users because it shows them syntax before they make mistakes. It is useful to intermediate programmers because it helps them through the task of putting modules together by showing them the syntax in a context-dependent way. It is useful for advanced users showing them syntax in a machine-readable form and when they are looking at new parts of the language.

COACH is designed to be a vehicle for demonstrating research ideas in adaptive user model technology. A COACH System User Manual is available for learning how to use the COACH system to make an adaptive user model based research system. Currently COACH is being used at the T. J. Watson Research Center for research on teaching Lisp, COACH Courseware is currently being built for UNIX and K-REP as well.

Bibliography

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

2. John Carroll and Amy Aaronson. Learning by Doing with Simulated Intelligent Help. *CACM*, 31(9), 1988.

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

4. B. J. Reiser, John R. Anderson, and Robert G. Farrell. Dynamic Student Modeling In An Intelligent Tutor For Lisp Programming. *IJCAI*, 1, 1985.

5. E. Risland. Understanding Understanding Mathematics. *Int. J. Man-Machine Studies*, 2(4), 1978.

6. O. Selfridge. Personal Communication, 1985.

7. D. Sleeman and J.S.Brown. *Intelligent Tutoring Systems*. Academic Press, 1982.

8. D. Sleeman and J.S.Brown. *Intelligent Tutoring Systems*, chapter 4. Academic Press, 1982.

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

10. W. Teitelman and L. Massinter. The Interlisp Programming Environment. *Computer*, 14(4):25-34, 1981.