THE DEVELOPMENT AND EVALUATION OF INSTRUCTIONAL STRATEGIES FOR AN INTELLIGENT COMPUTER-ASSISTED INSTRUCTION SYSTEM

Elliot M. Soloway

COINS Technical Report 80-04

January 1980

This report contains the technical sections of a proposal to the U.S. Army Research Institute for the Behavioral and Social Sciences.

This work was supported by the Army Research Institute for the Behavioral and Social Sciences, under ARI Grant No. DAHC19-77-G-0012.

EXECUTIVE SUMMARY

The promise of "computers in education" is one-on-one, individualized instruction in which the computer system tailors the instructional dialogue to meet the needs and abilities of each student. To meet this challenge, we must go beyond the traditional Computer-Assisted Instruction (CAI) and programmed instruction approaches, and make the computer system significantly more "intelligent." To date, the software technology necessary to achieve this objective has not been available. However, recent work in Artificial Intelligence, a branch of Computer Science, has resulted in the development of "expert systems." These systems exhibit performance comparable to that of a human expert in applications such as disease diagnosis, spectral analysis, and mineral prospecting. We believe that the technology used to develop these types of expert systems will allow us to construct a CAI system which exhibits expert abilities, i. e., an Intelligent CAI (ICAI) system.

During the past three years, we have made steady progress towards achieving this goal. A key aspect of this work has been the identification of the essential roles, functions, and engineering requirements needed by such a system. To this end, we have explored Instruction and Learning by (a) building a computer system which learns, (b) developing a classification scheme for problems which makes explicit the underlying structure, and (c) empirically investigating problem solving skills. Based on this experience, we have outlined the design of the "ideal" Student Environment, which can respond intelligently to the needs of each student, and the "ideal" author Environment, which can facilitate the transfer of the author's expertise into the system. For example, The Student Environment needs to understand the subject matter, possess a model of the student's understanding, and possess a data base of alternative instructional strategies; these components can work together to tailor the instructional dialogue for each student. Likewise, the complimentary

function of the Author Environment is facilitated by a similar structure. Finally, we have developed a rudimentary, prototype system, MENO-I, to examine the feasibility of this approach.

We propose to build directly on the work highlighted above and (1) construct MENO-II, a complete Intelligent CAI Student Environment system; and (2) evaluate and compare the effectiveness of two state-of-the-art instructional strategies (Coaching and Socratic Tutoring) as implemented in MENO-II. The first 18 months of this two year project will be devoted to completing the detailed design, implementation, and formative evaluation of MENO-II. The summative evaluaton will occur in year two; half the students in the progamming language course, the subject domain of MENO-II, will be exposed to MENO-II equipped with a Coaching strategy, while the other half will be exposed to MENO-II equipped with a Socratic Tutoring Strategy. A comparison of the performance of the two groups of students on various tests will be made; in particular, the degree to which students are able to transfer what they learn will be assessed.

Based on our experience over the last three years, and on the sophisticated software technology being developed for expert systems, we feel that the goal of an Intelligent CAI system is realizable now. The steps we have outlined provide for both an orderly development and an effective evaluation of such a system. The projected cost of this two year project is $151,000.

I. TECHNICAL DISCUSSION

I.1 Description of Current Research 2

I.1.1 Pursuing the Goals of Intelligent CAI: The Promise of Artificial Intelligence 2

I.1.2 The Components of the IDEAL ICAI System 4

I.1.2.1 The IDEAL Competence Model 7

I.1.2.2 The IDEAL Student Environment 9

I.1.2.3 The IDEAL Author Environment 11

I.1.3 A Description of MENO-I: An ICAI System for the Programming Language LISP 11

I.1.3.1 The Competence Model of MENO-I: The System's Knowledge Base 12

I.1.3.2 The Student Environment of MENO-I 18

I.1.4 A Problem Solving Taxonomy 23

I.1.5 The Utility of Programming Languages for Enhanced Problem Solving: Empirical Evidence 27

I.2 Proposed Research: Objectives and Description 32

I.2.1 Instructional Strategies for MENO-II 33

I.2.1.1 A New Language for Specifying Instructional Strategies 40

I.2.2 Modeling the Student's Understanding: The Performance Model 42

I.2.3 An Improved Dialogue Facilitator: A Sophisticated Menu System 46

I.2.4 Evaluation of the MENO-II ICAI System 49

I.2.4.1 Formative Evaluation: Feedback from Experiments . . 49

I.2.4.2 Summative Evaluation 51

I.1 Description of Current Research

I.1.1 Pursuing the Goals of Intelligent CAI: The Promise of Artificial Intelligence

What are the goals of Computer-Assisted Instruction? From our perspective, it is the creation of an ENVIRONMENT in which a STUDENT can explore a subject domain in a manner tailored to the needs and capabilities of the individual student so as to facilitate the student's understanding of that domain. We would also like that environment to facilitate the instruction of knowledge acquisition and problem solving skills. Clearly, these goals transcend CAI; they are the goals of education [Simon 1978]. One also needs to create an ENVIRONMENT for the AUTHOR of a CAI course; this environment should facilitate the transfer of an author's expertise into the computer for use as a CAI course. A serious problem in the wide-spread utilization of CAI is the current reality of each new CAI course requiring a major investment of time on the part of authors; it should be possible to help authors to evolve new courses based on previous courses.

What is Artificial Intelligence (AI) and how can it contribute to the development of rich STUDENT and AUTHOR ENVIRONMENTS? AI is a branch of computer science, one of whose goals is to develop techniques which imbue computers with the ability to interact with humans in a flexible and robust manner [Feigenbaum 1977]. The promise of the marriage between AI and CAI, then, is precisely the creation of the STUDENT and AUTHOR environments; AI techniques promise to enable a CAI system to respond to both the basic and subtle needs of students and authors [Carbonell 1970, Peelle and Riseman 1975].

The creation of such environments does not rest solely on progress in AI alone. Breakthroughs in education must occur before we can build truly effective instructional systems. However, in order to develop techniques

powerful enough to provide the capabilities suggested above, AI must come to grips with some of the basic questions of intelligence: what is the nature of knowledge -- its organization, representation and content; and what are the processes involved in understanding and learning. As workers in AI, we are not so presumptious to think that we will unlock all the secrets of intelligence. Yet, the metaphor of the computer, and the ability to quantify intelligence as "computer programs", may be the tools needed to enable researchers to push the understanding of these questions to a deeper level.

Whether or not we learn enough to build the ideal intelligent CAI system, we certainly feel that improvements over traditional CAI can and have been made {1}. In the next few chapters we shall outline our efforts towards achieving fundamental advances in the state of computers in education. While we shall fall short of the ideal, the system which we have created, and the system which we propose to create, should provide facilities necessary for the next generation of intelligent CAI systems.

In the next section we first briefly highlight the components of an IDEAL ICAI system. A more detailed description of each major component will be provided.

{1} See the collection of papers in the special issue of International Journal of Man Machine Studies, 11, 1979.

I.1.2 The Components of the IDEAL ICAI System

There are three top-level components in the IDEAL ICAI system (Figure 1).

- The Competence Model for the Subject Domain
- The Student Environment
- The Author Environment

The Competence Model is the system's understanding of the subject matter and as such is the heart of the system. In the Student Environment, it serves as the ideal against which the student's understanding is compared; the difference between what the student knows and what he is supposed to know can be utilized by the instructional strategy. Also, the Competence Model does not contain a parsimonious description of the subject area; quite the contrary, the knowledge is rich in metaphors and analogies in order to tap into a whole range of different student knowledge states. With respect to the Author Environment; the Competence Model serves as a base from which an author in concert with the environment can evolve a new course; for example, if a Competence Model for high school algebra existed, it should be used in the construction of a new Competence Model for, say, computer programming in BASIC.

The Student Environment, as depicted in Figure 1, is composed of four major components: the Dialogue Facilitator, the Instruction Monitor, the data base of alternative Instructional Strategies, and the Performance Model of the student's evolving understanding. The Dialogue Facilitator facilitates the interactions between a student and the system; if, for example, natural language were used as the medium of communication, then the Dialogue Facilitator would parse student responses and generate the system's comments. What the system will say is determined by the Instruction

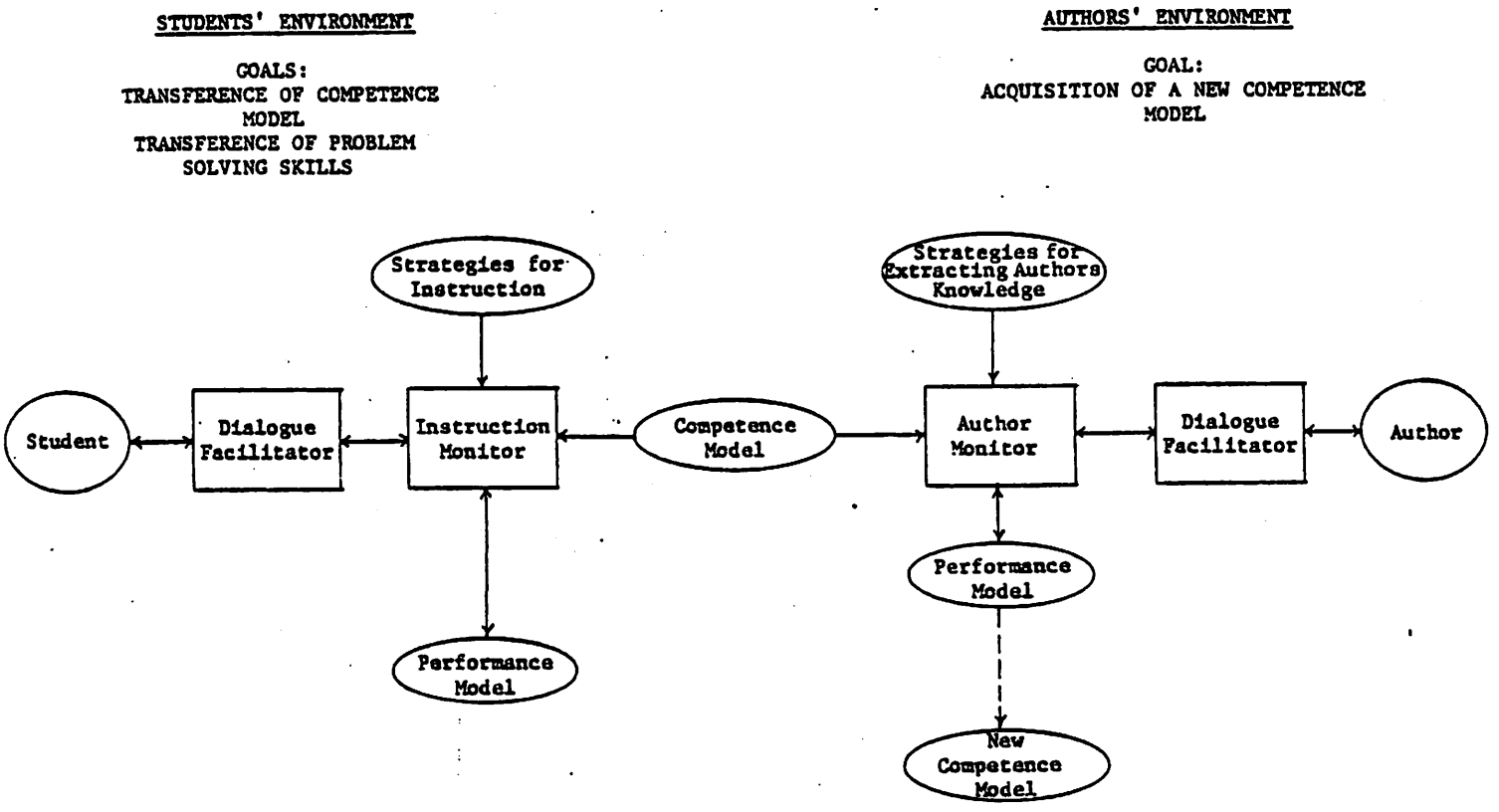


Figure 1

Components of an IDEAL ICAI System

Monitor; this process compares what the system believes the student knows, stored in the Performance Model, with what the student is supposed to know, stored in the Competence Model, and then chooses a specific instructional strategy, from a data base of alternative ones, to use in tutoring the student.

We view the role of the Author Environment as analogous to that of the Student Environment; instead of facilitating the transfer of knowledge from the system to the student, the Author Environment must facilitate the transfer of knowledge from an author into the system. Thus, the composition of this environment is symmetric to that of the Student Environment. The Dialogue Facilitator again manages the communication medium. The Author Monitor constructs a model of what it understands the author to be saying about a subject area; this Performance Model of the author will become a new Competence Model. The current Competence Model can be utilized by the Author Monitor in conjunction with a data base of strategies used for ferreting out what the author knows, to more precisely evolve the new CAI course, e.g., analogies to other courses can be made through the Competence Model and used in this enterprise.

The diagram in Figure 1 illustrates the interconnection of the components of the IDEAL ICAI system. In the following sections we shall describe in more detail the rationale for and capabilities of each of these components.

I.1.2.1 The IDEAL Competence Model

The main functions of the Competence Model are:

1. To provide a description of the subject matter under consideration.
2. To provide relationships to other, analogous subject matters.

The knowledge represented in this data base is a map of the subject area; quite literally, this knowledge base provides the system with "competence" in the subject area. Also, the analogies and metaphors contained in the Competence Model add richness and robustness to the instructional process by providing reference to concepts and relationships outside the given subject area; this will be useful in dealing with students from differing backgrounds.

Content alone is not enough; the knowledge must be structured and organized to facilitate effective and efficient access. Two design principles are relevant here. First, the knowledge should be structured axiomatically; that is, a hierarchical, tree structure, starting with primitive concepts at the bottom and building defined concepts strictly in terms of primitives and other defined concepts [Suppes 1957, Stelzer and Kingsley 1975].

There are four major advantages to an axiomatically structured knowledge base. First, the system knows what it knows -- the defined concepts -- and, it knows what it does not know -- the primitive concepts. Thus, the system can give explanations only for the defined concepts. Second, the primitive concepts serve as the PRESUMED knowledge base of the student. That is, the author assumes that a student taking his course will already understand the primitive concepts. If it turns out empirically that the author's guess as to the primitives is incorrect, then the author

can simply define concepts previously considered primitive. {1} Third, in developing a Competence Model, the author must actively confront what he believes to be the tacit (presumed) knowledge of a student in order to effectively decide which concepts should be primitive and which defined [Collins 1978]. Fourth, the precision of the axiomatization will aid the system in its task of building the Performance Model.

The second organizing principle is that the concepts should be grouped according to the role which they serve in the subject matter. For example, the ICAI system to be described shortly, tutors students in a computer programming language (LISP). A natural decomposition of such a "language" subject area is in terms of syntax (how an element is constructed), semantics (what an element does), and pragmatics (when an element can be used). In fact, we contend that it is the pragmatics of a language which is of overriding importance; for example, the pragmatics of a programming language contains the general problem skills that are really the goal of the instructional process (see also Goldstein and Papert 1977, and Brown, Collins, and Harris 1978). Defining in general, the types and numbers of levels for any arbitrary subject domain is an open question; a Theory of Subject Matter Structure would be required to resolve this issue.

A Competence Model possessing the content and form described above would aid the ICAI system in building a model of the student's understanding. That is, we can view the Performance Model of the student as some transformation of the Competence Model, e.g., if a student displays a misunderstanding of a concept, the system can infer based on the Competence

{1} Note, that the notion of what is "primitive" is relative to the subject domain; we are not here arguing pro or con concerning "absolute primitives" [Schank 1973].

Model what other concepts the student probably doesn't understand either. Based on the differences between the Performance Model and the Competence Model, the Instruction Monitor could then select analogies, metaphors, or other types of descriptions for presentation to the student which would be tailored to the specific misconceptions, and knowledge state of the student. With regards to facilitating an effective Author Environment, the system's current Competence Model will serve as the basis for building bridges to new subject areas (and hence new Competence Models). These issues will be explored more fully in the next two sections.

The issues raised in the construction of Competence Models are ones that are constantly under investigation in Artificial Intelligence under the ruhbrick "Knowledge Representation". In this area, questions such as how knowledge can be represented inside the computer [Hendrix 1976, Newell 1973, Minsky 1975], what the content of such knowledge should be [Schank and Abelson 1977, Minsky 1975, Schmidt 1976], and what the organization and structure of that content should be [Schank and Abelson 1977, Minsky 1975, Newell 1973] are being explored. Progress in this area will continue to have a significant impact on the design and construction of Competence Models.

I.1.2.2 The IDEAL Student Environment

In order to realize the full potential of computer based instruction -- personalized instruction -- the ideal ICAI system must build a model of what it perceives is the state of the student's understanding, i.e., the Performance Model. Such a model can then be used, by the Instruction Monitor in concert with the Competence Model and the data base of alternative Instructional Strategies, to tailor the instruction process to

the particular needs of a student. This is a dynamic process since different strategies may be necessary to accommodate the expected and observed changes in the student's understanding and learning style.

Note that the task of building such a model is non-trivial; it requires a learning system which can, through observations, hypothesize and verify knowledge that is consistent with the foregoing behavioral observations, and in addition, can successfully predict future behavior patterns [Soloway 1978].

The IDEAL CAI system will need a data base consisting of alternative Instructional Strategies. For example, a description of the Socratic Method, and a description of instruction-by-example might be two strategies included in this data base. One way to view instructional strategies is as types of algorithms. The work of Collins [1977], in defining the Socratic Tutoring Method as a set of production rules, as well as our own work on defining the primitives of an instructional language [Soloway 1979a], and the earlier more simplistic and basic languages such as COURSEWRITER can be seen as steps in this direction.

A Dialogue Facilitator is needed in order to manage the communications interface between the user and the system. The choice of medium for this communication is an open question; some have argued for natural language [Codd 1974], while others have argued for a sophisticated menu-based system [Newell 1977]. There will probably be no universal right answer to this question. The key issue will be in matching up the abilities of the components in the ICAI system with their communications requirements. For example, a system which can not answer arbitrary questions need not have natural language communications interface.

I.1.2.3 The IDEAL Author Environment

The key problem for the Authoring Environment is facilitating the creation of new CAI courses; the author must transfer his expert knowledge to the system's Competence Model. This process is complimentary to that of the Student Environment, there the Competence Model must be transfered to the student. This similarity in function is reflected in the design of the two environments; Figure 1 illustrates their compositional symmetry.

An intelligent Authoring Environment should be able to effectively interact with potential authors who have little or no understanding of the ICAI system; authors should not be required to be CAI experts. Rather, the Authoring Monitor should, via the Dialogue Facilitator, be able to query the author and extract from him knowledge in his area of expertise. The techniques for this process are stored in a data base and are selected by the Authoring Monitor based on a number of factors such as past utility, requirements of the Competence Model, etc.

On a more sophisticated level, the system may inquire of the author whether or not the subject matter being developed has any analogies to pre-existing subject matter domains. If so, the system may be able to draw on its current Competence Model in order to build a new Competence Model, thus raising the possibility of automatic generation of new Competence Models from old ones. In this way, new information can be linked to old, and new courses can be evolved based on previous ones. This cumulative effect is the key to solving the problem of CAI course development.

I.1.3 A Description of MENO-I: An ICAI System for the Programming Language LISP

In this section, we shall describe a system, MENO-I, which is

implemented and running at the University of Massachusetts. It consists of a knowledge base -- the Competence Model -- for the instruction of introductory LISP, and a set of questions with which a user, via the Dialogue Facilitator, can interrogate that knowledge base (Figure 2). These questions directly reflect the structure of the knowledge base; they also subtlely instruct the student in a good problem solving technique -- ask critical questions. Currently, there is no instructional strategy overseeing the student's exploration. In this section, we shall highlight the salient characteristics of MENO-I and limit our description to what is actually implemented; in Section I.2 we shall propose to extend MENO-I along the lines of the ideal ICAI system described earlier into a more complete intelligent Student Environment, MENO-II.

I.1.3.1 The Competence Model of MENO-I: The System's Knowledge Base

The subject area for MENO-I is that of LISP, a LISt Processing programming language. The Competence Model for this domain was built in accord with the design goals for an IDEAL Competence Model: (1) that it be axiomatically structured; (2) that the concepts be organized in levels thereby reflecting the use (or role) of those concepts in the subject matter; (3) that the Competence Model be densely populated and richly interconnected. {1}

With respect to the first design goal, a partial axiomatization of LISP was developed for MENO-I. The results of this effort are reflected in the Competence Model of MENO-I; 120 primitive and defined concepts have implemented which covers approximately 25% of a four week classroom course

{1} These issues are discussed in more detail in Stelzer, Soloway, and Woolf [1979], Stelzer [1978], and Woolf [1979].

on LISP. An example of a defined concept is the syntax of an ATOM.

> An ATOM is a string of characters that must-begin-with a letter.

In this description string, characters, must-begin-with, and letter are primitive concepts, presumed to be known by the student already. The actual implementation of this axiomatization is in the form of a semantic network [Hendrix 1976]. The network description for the ATOM definition is depicted in Figure 3 (syntax level, lower left hand corner).

In addition to being axiomatically structured, the concepts in the Competence Model of MENO-I are also organized into coherent groups or levels based on the role which a concept plays. Since the subject domain at issue is that of programming LANGUAGEs, we borrowed from the linguists their characterization of language as being composed of:

Syntax - describes how the concept is constructed

Semantics - describes what a concept means

Pragmatics - describes how a concept is used.

For example, all the concepts which deal with how elements of the language are constructed -- its syntax -- are collected together in the Syntax Level. Many concepts have definitions at all three levels; in Figure 3, the syntax, semantics, and pragmatics of the concepts ATOM and LIST are illustrated.

While we have recently been concentrating on identifying the pragmatic descriptions of programming concepts, only 1/3 of the 120 concepts currently in MENO-I's Competence Model have pragmatic descriptions attached to them. Most textbooks on programming languages present the syntax and semantics of the language. However, they usually do a poor job of telling the reader when to use a particular construction, or what roles various components play in the workings of the language. Without a doubt, these are difficult

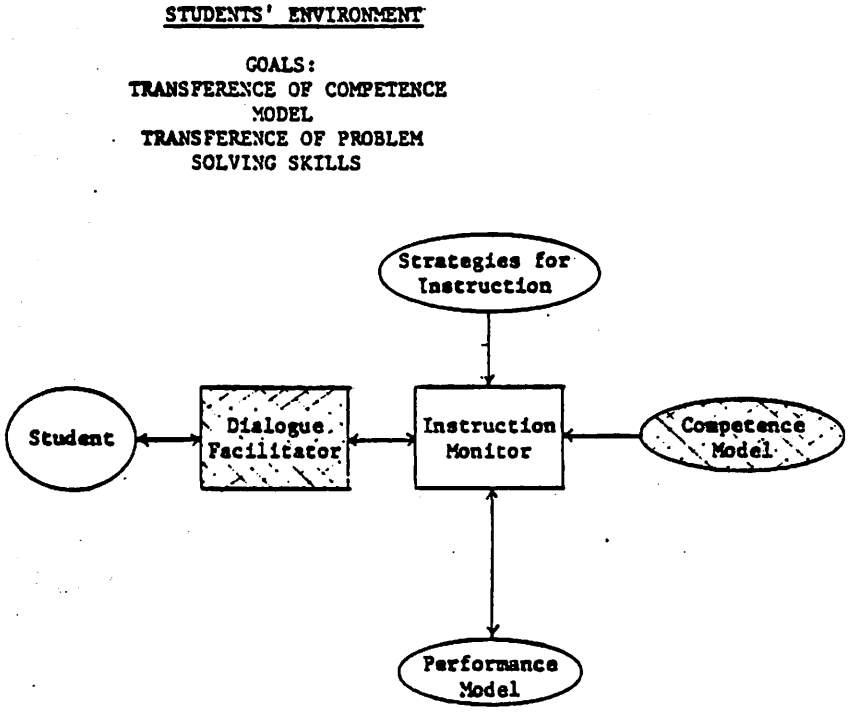


Figure 2

The Components of MENO-I

MENO-I consists of a Dialogue Facilitator and a Competence Model for the subject area of computer programming in LISP. These components are indicated by the shaded boxes above.

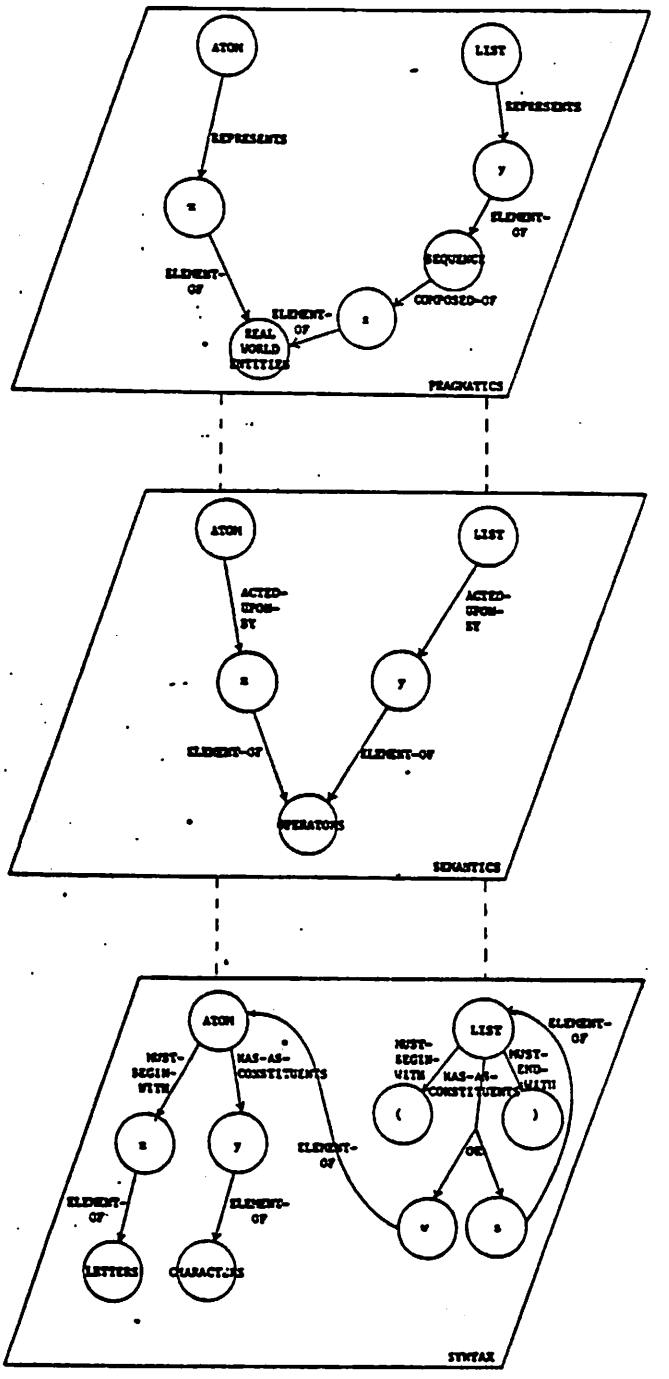


Figure 3

Grouping Concepts into Descriptive Levels:
Syntax, Semantics, Pragmatics

Depicted here are the concepts "ATOM" and "LIST" and the network representaions of their descriptions. The English translation of the network for the Pragmatics of ATOM would be: An ATOM represents those things which are considered to be (elements-of) the class of real-world entities.

questions to answer. But, without recourse to the three tiered structure we are proposing, it is not even clear that these questions even get asked!

The third design goal, a densely populated and richly interconnected Competence Model, facilitates the generation of analogies. Analogies play the role of relating knowledge from different subject areas. For example, if a student with a background in mathematics is attempting to learn LISP, the system should be able to make analogies, where appropriate, to concepts in mathematics; Figure 4 depicts an analogy between the LISP concept `ATOM' and the mathematical concept `NUMBER'. Currently, MENO-I can generate about 17 analogies drawn from knowledge bases for LISP, Chemistry, Mathematics, English.

The MENO-I algorithm for generating analogies is a graph-matching one. If a knowledge structure in mathematics, say, can be mapped onto a knowledge structure in LISP, then an analogy can be made between the two knowledge structures. This technique is one of those identified by Stelzer [1979a] in his formal investigation of the mathematical status of analogy. There are some serious computational problems with finding analogies using this technique [Machtey and Young 1978]. However, we have limited the scope of the matching in order to circumvent this computational problem -- at the risk of losing some analogies that are in the knowledge base, but are too costly to be found.

The analogy depicted in Figure 4 is a "linguistic" one; it is expressed in words. However, we <u>do</u> have the capability of displaying "pictorial" analogies also. For example, a pictorial analogy for the LISP concept `LIST' is illustrated in Figure 5. The algorithm for this type of analogy draws on the previously mentioned one. However, instead of printing "words", a computer program stored at the concept node in question is

```
analogy :atom:

WHAT TYPE OF ANALOGY FOR A ATOM WOULD YOU LIKE? (PICTORIAL OR LINGUISTIC)
ling
IN WHAT FIELD ? (ENGLISH,MATH,CHEMISTRY)
math

A ATOM CAN BE THOUGHT OF IN TERMS OF MATHEMATICS

ATOM   <IS TO>  LISP
         AS
NUMBERS  <IS TO>  MATHEMATICS

THEY ARE BOTH THE BASIC BUILDING ELEMENTS IN THEIR SUBJECT FIELDS
```

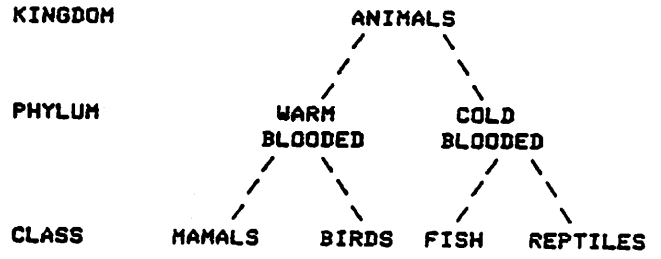
Figure 4

A Linguistic Analogy Generated by MENO-I

```
analogy :list:
WHAT TYPE OF ANALOGY FOR A LIST WOULD YOU LIKE? (PICTORIAL OR LINGUISTIC)
pic
IN WHAT FIELD ? (LISP,ENGLISH,MATH)
eng

A LIST CAN BE SEEN IN TERMS OF AN ILLUSTRATION

     PHYLOGENETIC TREE
```




```
(ANIMALS (WARM-BLOODED (MAMALS BIRDS)) (COLD-BLOODED (FISH REPTILES)))
```

Figure 5

A Pictorial Analogy Generated by MENO-I

executed which results in the printing of a picture.

I.1.3.2 The Student Environment of MENO-I

The Student Environment of MENO-I contains only a Dialogue Facilitator; the student is provided with the ability to explore the Competence Model by asking it questions. For example, one can ask "how is concept X constructed?" A reasonable answer to this question would be to return to the description of the syntax of that component (Figure 6a). Similarly, one could ask about the semantics -- "what does concept X do?" (Figure 6b) -- and one could ask about the pragmatics -- "what role does concept X play?" (Figure 6c). Also, one could ask "what is concept X?" (Figure 6d). A reasonable answer to this question could make reference to the classes to which concept X is a member, i.e., the superclasses of concept X. Figure 7 lists the set of questions available to the student (those asterisked are currently not implemented). This list of questions evolved based on our classroom experience with real student questions, and on our needs in constructing the actual Competence Model; that is, the list of questions which the student can ask reflects the questions we needed to ask ourselves in order to build the knowledge base originally.

The student can also ask the system to generate an example of a concept, or test whether or not a student supplied response is a correct example of some concept [Vermeer 1979]. In Figure 8a we see a student asking to see examples of an ATOM, while in Figure 8b we see the student testing himself by typing in proposed examples of the concept ATOM. Note, the system does not respond to an incorrect student input by simply saying "wrong"; rather, it attempts to diagnose where the student's error is and inform him of the problem. In Figure 8c we see the system testing the

```
-how is an :atom: constructed?

     ATOMS    HAS-AS-MEMBERS        CHARACTERS
     ATOMS    MUST-BEGIN-WITH-A     LETTERS
     ATOMS    MUST-END-WITH-A       NOT-APPLICABLE
```

(a)

A Student Requests to See the Syntax of an ATOM

```
-what does an :atom: do?

     ATOMS    ARE-ARE-ACTED-UPON-BY MEMBERS-OF-THE-OPERATOR-CLASS
```

(b)

A Student Requests to See the Semantics of an ATOM

```
-what is the role of an :atom:?

     ATOMS    ARE-USED-TO-REPRESENT REAL-WORLD-ENTITIES
```

(c)

A Student Request to See the Pragmatics of an ATOM

```
-what is an :atom:?

     ATOMS    MEMBER-OF-THE-CLASS DATA-ELEMENTS
     ATOMS    MEMBER-OF-THE-CLASS LANGUAGE-ELEMENTS
```

(d)

A Student Requests to See the Superclasses to which an ATOM Belongs

Figure 6

Examples of MENO-I Responses to Student Requests

In the above, student input is denoted by lower case letters, while MENO-I output is denoted by upper-case letters.

Simple Version	Alternative Versions
PRAGMATICS :?:	WHAT ROLE DOES :?: PLAY
SEMANTICS :?:	HOW DOES AN :?: WORK HOW DO :?: WORK WHAT DOES AN :?: DO WHAT DOES A :?: DO WHAT IS THE MEANING OF AN :?: WHAT IS THE MEANING OF A :?: WHAT DOES :?: DO
SYNTAX :?:	HOW DO YOU CONSTRUCT A :?: HOW DO YOU CONSTRUCT AN :?: HOW IS AN :?: CONSTRUCTED
SUPERCLASS :?:	WHAT IS A :?: WHAT IS AN :?: CLASS OF :?:
SUBCLASSES :?:	WHAT ARE SOME SUBCLASSES OF :?:
EXAMPLE :?:	GENERATE AN :?: GENERATE A :?:
CHECK :?:	CHECK MY EXAMPLE OF A :?: CHECK MY EXAMPLE OF AN :?:
ANALOGY :?:	ANALOGY FOR A :?: ANALOGY FOR AN :?:
* BIG-PICTURE	* WHERE AM I IN THE NETWORK
* RELATIONSHIP-OF :x: :y:	* WHAT IS THE RELATIONSHIP OF CONCEPT :x: TO CONCEPT :y: * HOW IS CONCEPT :x: RELATED TO CONCEPT :y:
* RELATIONSHIPS :?:	* WHAT ARE THE RELATIONSHIPS OF CONCEPT :?:

Figure 7

Questions Which a User Can Ask of MENO-I

The Dialogue Facilitator understands the above questions; it can respond with answers to them by drawing upon on the knowledge base. Note that these questions are independent of our particular subject area; they reflect the structural relationships inherent in a subject matter.

```
-example :atom:?

    MEALS


-example :atom:?

    HAKLZO
```

(a)

A Student Requests Examples of a Concept: ATOM

```
-check my example of an :atom:

    ENTER EXAMPLE OF ATOM

:ari
    ARI - IS AN ATOM
```

(b)

MENO-I Checks a Student's Examples

If a student types an incorrect example, MENO-I dos not simply respond "INCORRECT"; rather, it tries to explain why the student's input is incorrect.

```
-check my example of an :atom:

    ENTER EXAMPLE OF ATOM

:5ari

    5ARI - BEGINS WITH A NUMBER; IT IS NOT AN ATOM


-check my example of a :functional-expression:

    ENTER EXAMPLE OF FUNCTIONAL EXPRESSION

:(first* alpha beta)

    TOP LEVEL STRUCTURE LOOKS GOOD

    FIRST* - IS AN ATOM

    ALPHA - IS AN ATOM

    BETA - IS AN ATOM

    (FIRST* ALPHA BETA) - IS A LIST

    THE OPERATOR "FIRST*" REQUIRES 1 ARGUMENT;
    YOUR EXAMPLE IS SEMANTICALLY INCORRECT
```

(c)

A Running Commentary in the Checking Process

The motivation behind the commentary is to make evident to the student the process which underlies the checking/generation of an example.

Figure 8

Illustrating the Generation and Checking of Examples

student's proposed example of a functional expression. {1} The "running commentary" supplied by the system as it analyzes this more complicated concept is meant to provide the student with a sense of the active procedure behind the generation/testing of this concept.

In the list of Figure 7, we have tried to identify questions which a student can use to explore and illuminate concepts in the Competence Model. While these questions will be explicitly used by the student attempting to learn a specific programming language, we contend that he/she will also be picking up good problem solving skills. In fact, we plan to test whether or not the ability to ask questions transfers when the student is confronted with a new programming language (see Section I.2.5). Thus, we hope that not only will the student be learning problem solving skills in mastering a programming language such as LISP, but also, the student will be learning "how to learn" by mastering how to interact with and ferret out information from a knowledge base.

This concludes the description of the components of MENO-I. In the next two sections, we shall describe our futher exploration of the knowledge needed for problem solving and computer programming; we plan to integrate the knowledge gained in this enterprise into the system's Competence Model. First, we shall describe our work in identifying the pragmatic knowledge involved in solving a class of problems. Next, we shall highlight the results we obtained from empirical studies comparing computer programming versus traditional algebra as a language for solving word problems.

{1} A "functional expression" in LISP is the term used for an operator (function) followed by the appropriate input arguments.

I.1.4 A Problem Solving Taxonomy

In developing knowledge for inclusion in MENO-I's Competence Model, a major emphasis has been on the identification of underlying common structure. This methodology led to a significant insight when we examined the problems and programs often used in the instruction of LISP; namely, problems which appear very different when stated in English, nonetheless have computer program solutions which appear quite similar. A closer inspection of the problems and programs resulted in the development of a taxonomy of problems, in which there are currently three categories: predicates, selectors, and modifiers. It turns out that ONE program template (schema) is all that is necessary in order to generate programs for ALL the problems just mentioned. This is an interesting and surprising result which we plan to investigate further [Soloway 1979b].

We shall try to give an intuitive feel for each of the problem types by posing problems that one might have if one had a garden and one wanted to retrieve some facts about that garden's history {1}. Figure 9 gives a layout for the garden in question for one year, 1978.

A predicate problem is one which asks a question and requires an answer of TRUE or FALSE. Typical predicate problems for the garden domain might be:

Was broccoli always planted in the North-bed?

or

Was the yield of corn high in all the beds that had broccoli in them the year before?

{1} We use gardening as an example subject domain since it is relatively intuitive; the taxonomy is most certainly independent of this domain, and can be easily transferred to other domains if desired.

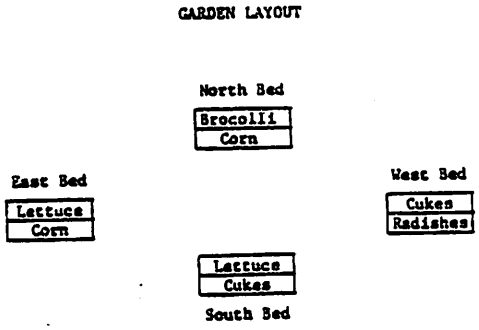


Figure 9

Garden Layout for 1978

```
? (psps-i)

   "DO YOU WANT A PREDICATE, A BUILDER, OR A SELECTOR?"
>predicate

   "IS IT AN 'AND' OR AN 'OR' PREDICATE?"
>or

   "WHAT IS THE FUNCTION NAME?"
>test-crop-or

   "WHAT IS THE VARIABLE LIST?"
>(bed-data crop)

   "WHICH IS THE CDRING VARIABLE?"
>bed-data

   "WHAT IS THE TEST ON THE IDEAL ELEMENT?"
>(equal crop (get-crop (car bed-data)))

   "HERE IT IS ....."

   (DE TEST-CROP-OR
         (BED-DATA CROP)
         (COND  ((NULL BED-DATA) NIL)
                ((EQUAL CROP (GET-CROP (CAR BED-DATA))) T)
                (T (TEST-CROP-OR (CDR BED-DATA) CROP))))


? (print-bed-history bed-n)
(1976 NS LETTUCE HIGH BED-N)
(1977 NS LETTUCE HIGH BED-N)
(1978 NS BROCOLLI MEDIUM BED-N)
(1978 NS CORN MEDIUM BED-N)

? (test-crop-or bed-n 'lettuce)
T

? (test-crop-or bed-n 'cukes)
NIL
```

Figure 10

Solving a Problem by Generating a Computer Program

In the above, the problem solving system generated a program to solve a user's problems. He wanted to know if lettuce had ever been planted in the North Bed, the answer being Yes (T); he also wanted to kow if cukes had ever been planted in the North Bed, the answer being No (NIL).

While these questions may appear syntactically different, they are nonetheless asking the same kind of question; namely, both require a TRUE or FALSE answer. The programs which solve the above problems in fact are identical except for "content" issues such as corn vs. broccoli, etc.

A selector problem is one which requires that a search be made for those entities which have the specified properties; this problem type requires as an answer the names of the entities which meet the specification. A problem of this type might be:

Which beds had cucumbers in them in 1978?

Here, the required answer is a list of only the beds in which some entity existed. Those beds which did not meet the specification are ignored.

A modifier problem is one which requires a search through some entities for those which meet some specification; the desired subset is then transformed in some manner. The answer returned here is the union of those entities not initially selected together with those entities which were subsequently transformed. A problem in this class might be:

> In order to project how the garden might look in 1979, replace lettuce by radishes.

The answer to this question would simply be a list of all the beds with their original contents, except that whenever lettuce appeared, now radishes would appear.

In order to evaluate the utility of this taxonomy -- and in order to make it very precise, we performed the following two experiments: (1) we used the taxonomy in classroom instruction of LISP, (2) we wrote a computer program which helps a user generate programs to solve the user's problems. With respect to the classroom evaluation, we felt, and the students agreed, that the taxonomy helped them to "move up a level" in their problem solving/programming ability by giving them a tool with which to look at new

problems. That is, the students saw that they could use the taxonomy to examine a new, and apparently different problem, and transform it into one which they already knew how to solve. This is precisely the generalization of problem solving skills that we set out to teach! Clearly, a more structured evaluation is required before any conclusions with regard to improved human problem solving skills can be drawn. Nonetheless, we were very encouraged by the students immediate grasp and use of the taxonomy.

We also developed a computer program, PSPS-I {1}, which interacts via questions with a user in order to ferret out the underlying issues in a problem, and based on the ansers to those questions generates a LISP program which solves the user's problem. For example, in Figure 10 we illustrate a typical interaction with PSPS-I. The problem that the user wants to solve is:

Were all the vegetables in the North-bed cucumbers?

PSPS-I first asked the user a series of questions, and then produced a LISP program to solve the problem. This program was then "run" (evaluated) on the particular bed in question with the resulting answer NIL (FALSE), i.e., no, not all the vegetables in the North-bed were cucumbers.

At this stage, the majority of the problem solving work is still done by the user; however, the stage is set for us to make the system smarter, thus requiring less technical knowedge by the user. Also, PSPS-I currently generates programs in LISP. However, the taxonomy is unquestionably independent of LISP. Therefore, we feel that via analogy mappings, we could generate program solutions in languages such as BASIC or FORTRAN.

{1} PSPS-I stands for Problem Solving by Program Synthesis.

I.1.5 The Utility of Programming Languages for Enhanced Problem Solving: Empirical Evidence

A number of research groups have claimed that computer programming is a good source for problem solving techniques. The LOGO group was an early and consistent proponent of this view [Papert 1971a, 1971b]. The Learning Research Group at XEROX-PARC with their language SMALLTALK hold a similar view [Goldberg and Kay 1977]. However, to date, there has been little empirical evidence pro or con for this position. Recently, the MENO Group, in conjunction with the Cognitive Development Project in the Department of Physics here at the University of Massachusetts, has performed a series of psychological tests which seem to indicate that, in fact, this hypothesis is correct. We shall briefly outline the motivation of the experiments, and then describe the results obtained [Clement, Lochhead, Soloway 1979].

The physics group, mentioned above, discovered that a surprisingly large number of college freshman engineering students had difficulty with the ratio problem depicted in Figure 11. In fact, out of 150 students 37 percent missed it! When an equation of the sort $nX = mY$ was required (Figure 12), the percentage of incorrect answers (73 percent) rose considerably higher. From video-taped interviews it appeared that the students understood the problem; they were able to describe verbally the relative sizes of the groups, and they even could draw pictures representing the different sizes. Nor does the difficulty seem to stem from a misunderstanding of the English description; the students were also given the same type of problem in pictorial form with similar performance resulting.

One of the hypotheses we developed in order to explain these results was that the students who gave the incorrect answer of `6S=P' for the algebra problem in Figure 11, did not interpret `6S' as an active process in

PROBLEM:

Write an equation using the variables S and P to represent the following statement: "There are six times as many students as professors at this university." Use S for the number of students and P for the number of professors.

Percent Correct	Percent Incorrect
63	37

n = 150

Correct Answer: 6P = S

Typical Incorrect Answer: 6S = P

Figure 11

A Ratio Problem Used to Assess Student Understanding of Math

A surprising number of college freshman engineering studens missed this seemingly simlpe ratio problem.

PROBLEM:

Write an equation using the variables C and S to represent the following statement: "At Mindy's restaurant, for every four people who order cheesecake, there are five people who ordered strudel." Let C represent the number of cheesecakes and S represent the number of strudels ordered.

Percent Correct	Percent Incorrect
27	73

n = 150

Correct Answer: 5C = 4S

Typical Incorrect Answer: 4C = 5S

Figure 12

A More Complex Ratio Problem

With a more complex ratio problem, the number of incorrect answers increased even more dramatically.

the standard mathematical way, namely `6 times S'. That is, they seemed to think that since the student group was larger -- which it is -- then in order to symbolize that picture mathematically, one put a `6' in front of the `S' thereby indicating the large size of the student group.

We then hypothesized that if the students were put in a situation which more strongly compelled them to take an active view of `6S', then their performance on this type of problem might improve. The obvious choice for such an active environment is computer programming. For a variety of reasons, not all of which have yet been identified, computer programming seems to suggest to students that they view numbers, variables and operations thereon, as active processes. However, we felt that, a priori, it would seem that writing a computer program to solve a problem such as the one in Figure 11 would be more difficult than writing an equation for the same task. Nonetheless, we felt that possibly the active environment would override the other complications which might arise in this environment.

On the basis of several experiments, it turned out, in fact, that significantly more students were able to solve problems of the ratio sort in the programming context, than were able to do so in the traditional algebraic (equational) context. (The results of one experiment are presented in Figure 13.) We have begun to do video-taped interviews to isolate the causes for this result; preliminary protocols indicate that the programming context does "set the stage" differently than does the equational one. A student solved a ratio problem wrong using an equation, and on the very same piece of paper solved a similar ratio problem correctly using a computer program!

We find this result to be very exciting. On the one hand, it is what many in the computer science/artificial intelligence community have for a

Program

Given the following statement:

"At the last company cocktail party, for every 6 people who drank hard liquour, there were 11 people who drank beer. "

Write a computer program, in BASIC or FORTRAN, which will output the number of beer drinkers when supplied (via user input at the terminal) with the number of-hard liquour drinkers. Use H for the number of people who drank hard liquour, and B for the number of people who drank beer.

	Percent Correct	Percent Incorrect	
Equation	45	55	
			$p < .05$
Program	67	33	
	n = 101		

Figure 13

Problem Solving Enhanced in a Programming Environment

The results of the above experiment indicate that students seem to be able to solve problems better in a programming environment as opposed to the more traditional equational (algebraic) environment.

long time believed. On the other hand, no hard evidence had been gathered to support this intuition. Furthermore, the implications for mathematics education are quite revolutionary. The "theorem-proof" style of math education is seriously called into question; greater emphasis on making mathematics active is required.

I.2 Proposed Research: Objectives and Description

In the last chapter we outlined the theory behind and the implementation of MENO-I, a rudimentary Intelligent CAI System. In this chapter we shall first describe how we plan to extend this work and build MENO-II, a full fledged ICAI Student Environment which will be able to tailor instruction to the needs of the different students. Next, we shall describe how we plan to evaluate and compare the two dominant ICAI instructional strategies, Coaching and Socratic Tutoring, as implemented in MENO-II. In particular, the following specific objectives will be described in the next four sections:

- Implement two state-of-the-art instructional strategies; develop a language for specifying instructional strategies.
- Implement the mechanisms for building a Performance Model of the student.
- Implement a student-system communications interface based on "menu-selection."
- Evaluate and compare the effectiveness of the two instructional strategies.

In the next section we shall first examine the two major instructional strategies which have emerged in the ICAI literature, showing first their common root, and then their differences. Next, by way of hypothetical dialogues, we shall show how these two strategies will be implemented in MENO-II. Finally, we shall highlight our design of a new language in which instructional strategies can be specified.

I.2.1 Instructional Strategies for MENO-II

Objective

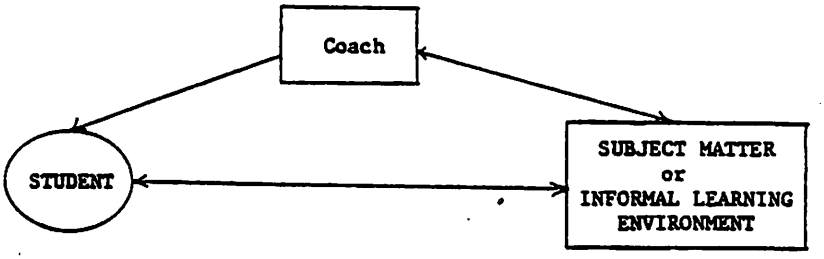
Implement two state-of-the-art instructional strategies; develop a language for specifying instructional strategies.

The recognized goal of CAI is individualized instruction, wherein each student interacts "with his own computer." While the decreasing cost of computer hardware will enable a one-on-one student-computer ratio, the real question arises as to which instructional strategies are best suited for this type of instruction? The "lecture hall style," in which a professor addresses 30 to 300 students, does not seem to take advantage of the ability to converse with and focus attention on individual students. Two strategies which do seem to take advantage of the one-on-one relationship is that of an athletic coach (Goldstein [1976, 1979], Burton and Brown [1979]) and that of a Socratic Tutor [Collins 1977]. A good coach tunes and sharpens the skills of an individual player; personal interaction is the key, since this process requires the subtle molding of students' skills. No standard procedure based on the "average" student would be as effective. The coach must also limit his comments and provide an environment in which the student can himself discover his strengths and weaknesses.

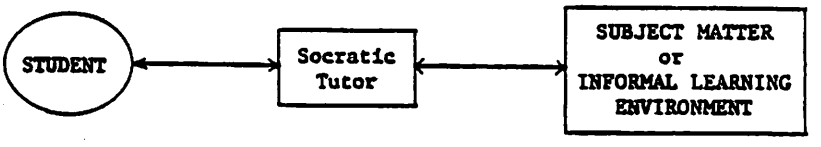
Socratic Tutoring, with its origins in the dialogues of Plato, is also based on the interaction/dialogue between two or more individuals. In this setting, however, the tutor proceeds in a more systematic manner conveying the subject matter to the student, while probing the student to uncover the student's misconceptions and confusions. Thus, it is not too surprising that these two instructional strategies should be at the heart of a number of ICAI systems [Goldstein 1979, Burton and Brown 1976, Miller 1979, Bates, Brown and Collins 1979], where the clear objective is the individualization of instruction.

While these strategies have a common root, they diverge on how they facilitate instruction individualization. The key difference is the degree to which the instructional strategy permits the student to DIRECTLY interact with the subject matter/learning environment (e.g., a subject matter such as LISP programming, or an informal learning environment such as a game). In the Coaching approach, the student is permitted almost total freedom to explore his environment directly, while in Socratic Tutoring, the student's probings are all filtered through the tutor. Figure 14 captures this distinction pictorially; in the Coaching Strategy, there is a direct link from the student to the environment with the Coach observing the students investigative attempts and intruding only at judicious moments. However, the Socratic Tutor fields all inquiries by the student, so the student's link to the environment is through the Tutor.

In Figure 15 we list a number of other differences between the two strategies; these differences either follow directly from the key difference mentioned above or are at least consistent with it. For example, since in the Coaching Strategy the student can interact with the environment himself -- and is, in fact, encouraged to do so -- the Coach directs or structures the instructional dialogue much less and possibly in a less systematic way than does the Socratic Tutor. Typically, the Coaching Strategy has been embedded in a game environment. Thus, the game has been the structuring/driving force in the environment, since the student is trying to resolve the competitive situations and win the game. The twists and turns of the game itself may control the situation and thus information and skills may not be introduced in a highly structured manner. Typically too, the Socratic Tutor has been embedded in a non-game environment (e.g., rainfall prediction [Bates, Brown and Collins 1979]), and thus the Tutor



(a)



(b)

Figure 14

A Pictorial Comparison of
Coaching and Socratic Tutoring

Differences	COACH	Socratic Tutor
Key Difference: 1. degree to which student DIRECTLY interacts with subject matter/informal learning environment	high	negligible
2. degree to which student is directed by instructional strategy	low	high
3. degree to which strategy responds to student errors and misconceptions	high	medium
4. degree to which instructional strategy intrudes or probes student	low	high
5. degree to which student evaluates his/her own answers and progress	high	low
6. degree to which the instructional strategy provides explicit answers to student errors or queries	low	medium

Figure 15

Differences Between Coaching and Socratic Tutoring

itself becomes the driving force by posing problems for the student to solve. In this case, the Tutor can expand into the subject matter much more systematically than can the Coach. While the Socratic Tutor does respond to student errors (difference 3), they are THE major signal to the Coach to possibly intercede. Difference 4 makes a complimentary point; while the Socratic Tutor is allowed to actively probe the student, in order to pinpoint the student's difficulty, the Coach, who is trying to keep his intrusions to a minimum, is not permitted this luxury.

All the differences listed in Figure 15 simply highlight the intuitive differences that come to mind based on our experience with Coaches and Tutors. Note, any particular implementation of these two strategies may vary the differences between them, accentuating or diminishing them. Furthermore, only Socratic Tutoring has been worked out in consistent detail and generality [Collins 1979, Stevens and Collins 1977]; a number of Coaching philosophies have been put forth [Goldstein 1979, Burton and Brown 1979, Miller 1979]. In the above discussion we have tried to remain as faithful to these many sources as possible.

We plan to implement these two strategies for MENO-II. This will provide the basis for (1) an evaluation and comparison of the two strategies, (2) the development of a general "language" in which instructional strategies, such as Coaching and Socratic Tutoring, can be specified.

We will first implement a Socratic Tutor for MENO-II. The hypothetical dialogue listed in Figure 16 illustrates how this strategy will look in our domain. The systematic explication of a concept, in this case the syntax of an ATOM, is the trademark of the Socratic Tutor. While Socratic Tutoring has been much more well specified than Coaching, and while claims for

ST: ATOMS IN LISP SERVE THE SAME ROLE AS NUMBERS IN ARITHMETIC
[state an analog]

ST: DO YOU THINK 'ALPHA' IS AN ATOM?
[ask for a prediction about a particular case]

S: I don't know.

ST: IT IS.

ST: DO YOU THINK THAT ALL THE SYMBOLS OF AN ATOM ARE CHARACTERS?
[form a general rule based on an insufficient factor]

S: Yes.

ST: DO YOU THINK '2BETA' IS AN ATOM?
[pick a counter example to the general rule]

S: Yes.

ST: 2 IS A CHARACTER, BUT THE FIRST SYMBOL OF AN ATOM MUST BE A LETTER; THUS '2BETA' IS NOT A LEGAL ATOM.

ST: DO YOU THINK 'ALPHA4' IS AN EXAMPLE OF AN ATOM?

⋮

Figure 16

Socratic Tutoring: A Hypothetical Dialogue

subject independence have been made for this specification, nonetheless changes will need to be made in order to effectively utilize this approach in our domain. For example, the major test of the Socratic Tutor has been in a domain containing causal relationships between physical processes (weather changes); our domain, however, has facts (e.g., the pragmatics of an ATOM) and abstract processes (e.g., LIST concatenations). Also, an assumption is made in Socratic Tutoring -- as well as in Coaching -- that the student already knows something about the domain; the task of instruction is then the refinement of the student's preconceived notions. However, in our case, a student will most likely not know anything about LISP; thus tutoring rules will have to be developed which utilize analogies to tap into domains which the student does know about in order to make connections to the new domain (see line 1 in the hypothetical dialogue, Figure 16).

Another hypothetical dialogue (Figure 17) illustrates the Coaching strategy in our domain. The less directive, more stand-offish, nature of the Coach is evident in this dialogue. While it is not evident from the dialogue, the student's queries for examples of a concept, and his request for a check of his generated example are routed to the Competence Model; the Coach watches the student and notes his/her actions, but the student is directly in contact with the subject matter.

On a closer inspection of both dialogues, the reader may notice that the questions asked by either strategy are similar, and in fact, the queries posed by the student are similar to those of the strategies. This observation leads into the development of a general language in which to state instructional strategies, the topic of the next section.

I.2.1.1 A New Language for Specifying Instructional Strategies

There are two general observations which can be drawn from the hypothetical dialogues. First, both the Coach strategy and the Socratic Tutor strategy made use of the same "questions" to interrogate and instruct the student; the major difference between the two strategies, in computational terms, is the control structure {1} for each strategy. The second observation is that the strategies AND the students in both dialogues used the same questions; that is, the students were also allowed to interrogate the instructional system {2}.

Taken together, the above two characteristics of the instructional dialogue serve as design criteria for a new and general language we plan to develop which will enable authors to specify instructional strategies. The objective in this design lies in having a common language which a student can use to explore the knowledge base, AND which an author can use to encode an instructional strategy. The basis of this common language is the set of "questions" (see Figure 7) which are available in MENO-I only to the student. That set needs to be augmented in order to facilitate additional types of interactions; nonetheless the key idea that questions should be at the heart of an instructional dialogue -- and thus, an instructional language -- is consistent with our earlier discussions of good instructional strategies for individualized instruction. Note, the development of this

{1} A control structure in a programming language specifies the order in which an event is to occur; it includes the ability to specify that an event take place on the condition that some other event occur. In FORTRAN or BASIC, the DO-loops or the FOR-NEXT loops, the IF-THEN statements, and the subroutine calls are types of control structures.

{2} This was more true of the Coaching strategy.

C: DO YOU THINK 'ALPHA' IS AN EXAMPLE OF AN ATOM?

S: I don't know.

C: IT IS.

S: What do I do now, help.

C: WHY NOT ASK TO SEE A FEW MORE EXAMPLES OF ATOMS.

S: Generate an example of an ATOM.

C: BETA2

S: Generate another example of an ATOM.

C: HBZQR

S: What should I do?

C: WHY NOT ASK ME TO CHECK YOUR ATTEMPT AT GENERATING AN EXAMPLE OF AN ATOM?

S: O.K., is 2BETA an ATOM?

C: NO

C: WHY NOT ASK TO SEE THE DEFINITION OF THE SYNTAX OF AN ATOM?

Figure 17

Coaching: A Hypothetical Dialogue

language will proceed concurrently with and based upon the implementations of the specific instructional strategies. {1}

A major benefit will accrue from this common language; namely, we will have made a giant leap from being only able to provide instruction in the subject domain itself; to being able to provide instruction in the instruction process itself! Since the language in which an author encodes instructional strategies is also available to the student, the student will be able to literally see the instructional strategy being employed by the system. He might then come to understand that strategy and use it himself to continue exploring the knowledge base. The student, in this situation, will have "learned about learning".

Finally, the specification of an instructional strategy is an educational question which can be explored empirically. Thus, we feel that this new language will provide a powerful tool in which alternative instructional strategies can be specified and evaluated. The educator will then be able to precisely define what is meant by "instruction by example", "instruction by lecture", etc.

I.2.2 Modeling the Student's Understanding: The Performance Model

Objective

Implement the mechanisms for building a Performance Model of the student.

Approach to Objective

An effective ICAI system must match up and balance the capabilities of its components. Therefore, while our main initial effort will be in

{1} We refer the reader to a working paper [Soloway 1979a] for a more detailed description of this new instruction specification language; an extended hypothetical instructional sequence is also presented there.

developing the Instructional Components for MENO-II, such effort would be wasted if the system did not have access to a rudimentary model of what the system believes is the state of the student's understanding. That is, without some idea of what the student knows or doesn't know, the instructional end of the system would be operating in the dark. Thus, we propose to develop a rudimentary Performance Model along the lines of one employed in the WHY System of Bates, Brown and Collins [1979]; the Overlay Model of Carr and Goldstein [1977] is another example of this type of Performance Model.

The Performance Model for the student will be a copy of the system's Competence Model. The two models will differ in that the Performance Model will be labeled with tags which indicate how well a concept is thought to be understood by the student, and how that judgment was made (Figure 18). For example, when the student comes to the course initially, his Performance Model will have only the labels "Assumed Known" on all the primitive concepts, and "Unknown" on all the others. As the student is exposed to the subject matter, the labels on the concepts should change to "Known", "Inferred Known", or "Told". Thus, the changing of the tags on the concepts illustrates the dynamic, evolving nature of the Performance Model and is meant to realistically reflect the student's evolving understanding.

The axiomatic structure of the Competence Model will aid us in the labelling process of the Performance Model. For example, from the structure of the Competence Model the system can easily determine which concepts are "primitives" and hence are assumed to be known by the student when he/she comes to the course (Assumed Known). Also, since the "derived" concepts are explicitly defined in a strictly hierarchical fashion, the system can make inferences about what is needed to be known in order to understand a later

Known - The student has explicitly demonstrated knowledge of this concept.

Assumed known - This concept is so trivial that the student is assumed to know it.

Inferred known - The student has demonstrated knowledge from which one may easily infer this concept.

Told - The tutor has told the student this information.

Mentioned - The student has mentioned this concept in an incorrect context, so he/she knows something about it but not how it relates to the overall process.

Unknown - The student has not yet demonstrated knowledge of this concept.

Figure 18

The Performance Model: Assessing The Student's Knowledge State

The student model may be thought of as a copy of the Competence Model in which each node is annotated with one of the above comments. (taken from Bates, Brown, and Collins [1979])

concept (Inferred Known).

While the Performance Model described above should be sufficient for the needs of MENO-II, we have no illusions as to its rudimentary nature. Thus, we plan to explore the development of more sophisticated Performance Models. In particular, if the student's understanding is different than the instructor's, i.e., the Performance Model is not a copy of the Competence Model, then how can the Performance Model be built up, and how can links from the Competence Model to the Performance Model be made? The key to this problem, we feel, is in the explication of the notion of analogy. A good instructor builds on what the student already knows by first making some initial connections to the student's concepts, then applying an analogy mapping from the student's concepts to his own understanding in order to generate new knowledge structures more in concert with what the student knows, and finally modifying his instruction to fit the new knowledge. For example, some students do not understand "tree structures" but do understand "pointer structures." While an instructor may have planned his instruction around "tree structures", a good instructor will recognize the student's weakness and strength, make an analogy from tree structures to pointer structures, and continue the instruction in terms of pointer structures.

How analogies can be discovered between two already existing structures, and how they can be used to generate a new structure from an old, are open questions. We have taken a first step towards of solving these questions by developing a formal, mathematical model of analogy [Stelzer 1979a]; we plan to continue developing that approach, and we also plan to begin turning that formal model into executable computer programs. The ability to effectively make and find analogies will be exceedingly useful in the construction and utilization of more realistic Performance

Models.

I.2.3 An Improved Dialogue Facilitator: A Sophisticatd Menu System

Objective

Implement a student-system interface based on "menu-selection."

Approach to Objective

In building systems which interface with people, one immediately runs into the communication barrier; what "language" should interface the man and the machine? It is a commonly held opinion among researchers (e.g., see Codd [1974]) in this area that a natural language, such as English, would be the best language for the user. However, the subtleties and flexibility of natural language expressions pose significant theoretical and practical difficulties for algorithmic description. In fact, natural language research is an area in and of itself [Schank 1977]. Since we want to direct our energies to issues more specifically related to CAI, we shall propose a different approach to coping with this problem.

We feel that the developing "menu-system" technology will serve our needs as a "front-end" interface. Such a system would present the user with information about some concept, and a set of alternative actions which the user could initiate. For example, in our problem, a menu system might display information about an `ATOM', and permit the user to ask the questions described earlier about ATOMS. We have chosen a particular menu system, called ZOG [Robertson, et al. 1977], to customize for our project. In what follows, we shall describe how this menu system will serve as a Dialogue Facilitator in MENO-II [Bonar 1979].

Figure 19 depicts the structure of a typical "menu frame" which would be painted on a display type terminal; Figure 20 depicts an instructional

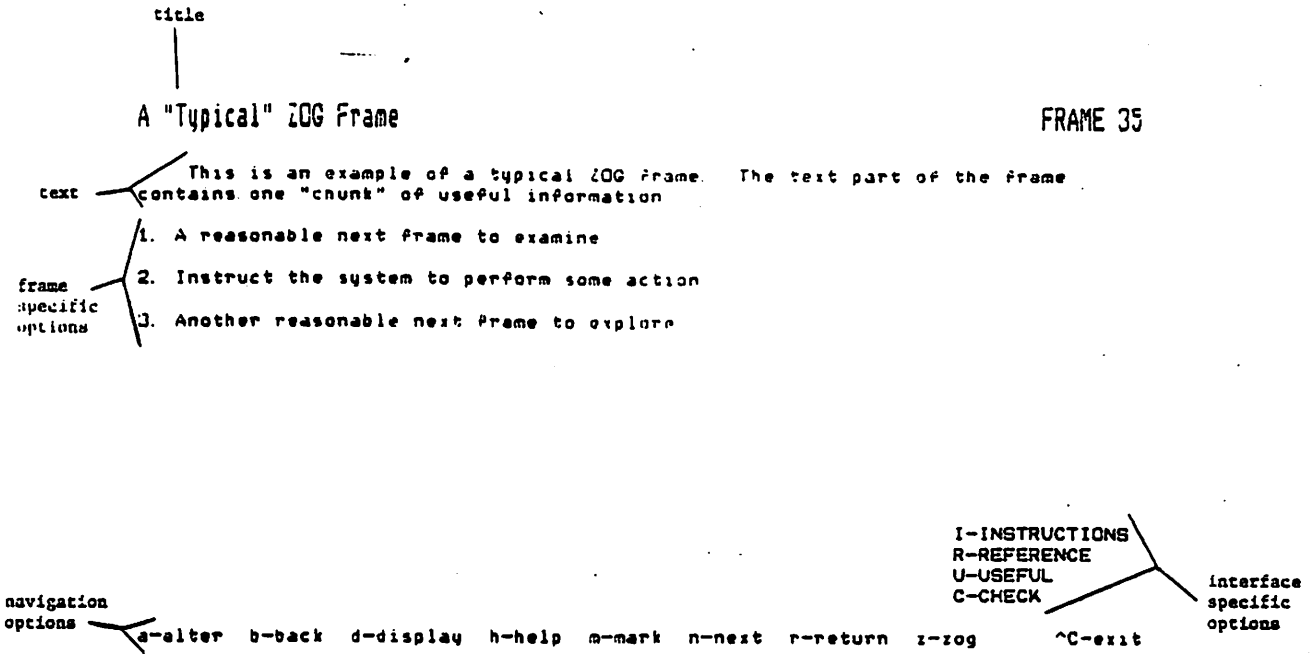


Figure 19

The Organization of a MENO Frame

The above illustrates how the information in a frame will look to a student when displayed on a CRT by MENO-II.

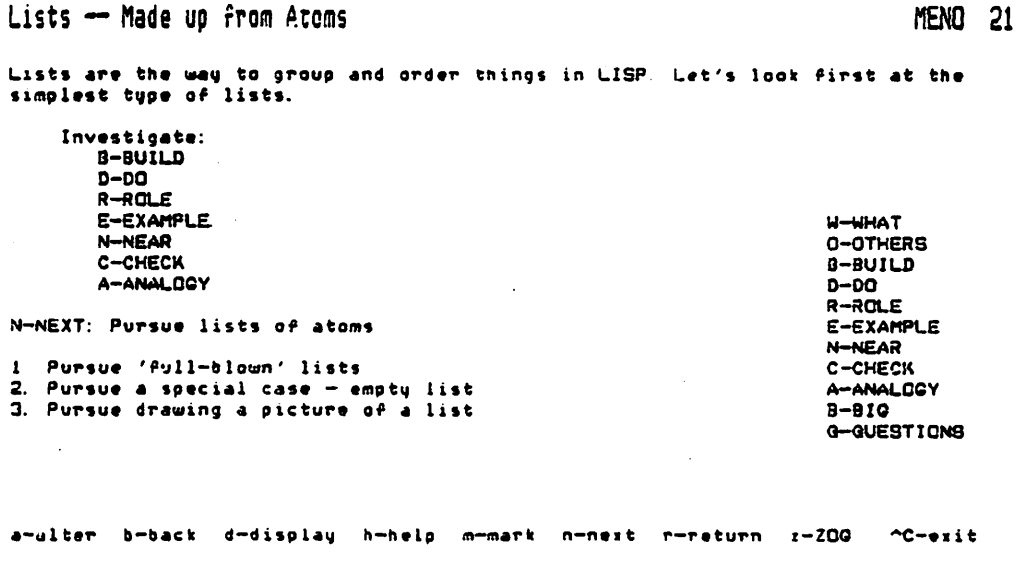


Figure 20

An Example Frame: LISTS

Descriptive information about a concept, LIST in this example, is only part of what is a available in a frame; the student can choose an action from the menus at the right hand side and bottom of the frame. These choices allow the student to explore the knowledge base himself, ask illuminating questions, or permit the instructional strategy to continue the instructional process.

frame for the concept `LIST'. Information is grouped into "frames" which consist of a concept, with actions that can be taken by the user in response to the presented material. In our case the presented material would be a concept node in the Competence Model, and the actions would be the set of questions appropriate to that concept. These questions will always appear on the right hand part of the screen. The information in the center of the screen is dependent on the instructional strategy. For example, if the student did not know what types of questions to ask about a concept, then the student could select one (or some) which are suggested by the particular instructional strategy; these are listed in the middle of the screen under INVESTIGATE. Also, the student could simply go on to the next concept in the lesson, by selecting the next option.

Tying the presentation of the material to a menu system does not mean that the result will be a programmed-instruction or "canned" course. The student always has the option to go anywhere he/she would like to in the Competence Model by simply asking questions about the concepts; the student need not elect to choose the NEXT-CONCEPT option. Nor is the concept which will be presented next decided before the student takes the course, as it is in programmed-instruction. Rather, the instructional strategy can dynamically select the next concept based on whatever information is available. The menu system in no way inhibits or restricts the user with respect to his flexibility to explore the subject matter; it only serves as a communications link.

I.2.4 Evaluation of the MENO-II ICAI System

Objective

Evaluate and compare the effectiveness of the `Coach' and `Socratic Tutor' instructional strategies.

As we mentioned earlier, the two instructional styles which have emerged in ICAI are the Coach [Goldstein 1976] and the Socratic Tutor [Collins 1977]. No comparison of the effectiveness of these strategies has been conducted. We are in the unique position to perform just such a comparison; the architecture of MENO-II will permit us to "plug-in" a Socratic Tutor module or a Coach module while keeping ALL other components constant. Thus, we propose to run one group of students on MENO-II using the Socratic Tutor module, and one group on MENO-II using the Coach module; performance measures from the two groups will then be compared.

There will be two types of evaluations: formative and summative [Wagner and Seidel 1978].

I.2.4.1 Formative Evaluation: Feedback from Experiments

Three experiments will be conducted during the first 16 months of our work. A course of four weeks in duration on LISP programming will be given {1}; in the first two experiments MENO-II will be used in conjunction with traditional classroom instruction, while in the third experiment, MENO-II will be used to provide the total instruction. A variety of evaluation techniques will be used in order to give us feedback on

- the adequacy of MENO-II
- the adequacy of the tests which we plan to use during summative evaluation

{1} In the chapter on the Project Schedule we discuss how we shall obtain the students for these courses, and who shall be in charge of the classroom instruction.

Complex computer systems, such as MENO-II, need to be tuned and balanced in order to maximize their utility; components must themselves be modified, but more importantly, their capabilities must be coordinated and integrated. Student evaluations resulting from experience with MENO-II will be important in this regard. For example, in order to assess the quality of MENO-II's Performance Model, we plan to use video-taped interviews of students actually engaged in interacting with MENO-II. A member from our research team will carefully interview the student as he/she interacts with MENO-II; the interviewer will attempt to assess changes in the knowledge state of the student, e.g., where the student's confusions are, what concepts he feels he understands. We will then compare the interviewer's guesses with MENO-II's guesses as reflected in the Performance Model. Also, we will video-tape two students working together; 1 student will explain to the other student the `what' and `why' of his actions. Again, comparisons to MENO-II's behavior will be made. This latter technique frees up the interviewer, and has been quite successfully utilized here at UMass by a member of our research team, Dr. J. Lochhead, in previous research in cognition and education [Lochhead 1979].

The video-tape interviews will also provide us with anecdotal observations concerning the performance of other components of MENO-II. For example, where can the layout of the "menu" of the Dialogue Facilitator be improved; does the Competence Model contain adequate explanations and sufficient analogies; to what degree is the instructional sequence being tailored to the individual characteristics of the students, etc.

We will also ask the students to take pre- and post-performance tests and fill out a subjective evaluation questionaire which we plan to use during the summative evaluation. This will help us debug and tune those

evaluation tools.

I.2.4.2 Summative Evaluation

The summative evaluation will be devoted to comparing the effectiveness of the Coach and Socratic Tutor strategies. During this period no changes to MENO-II will be made. The experiment will consist of running half the students, in the programming language course, on MENO-II equipped with the former strategy, and the other half on MENO-II equipped with the latter strategy. Data will be collected on student performance and data will be collected on computational aspects of the ICAI system itself. With respect to student performance, three types of information will be gathered:

1. Performance of students on tests dealing with subject area.
2. Transfer of learned skills to other related subject areas.
3. Subjective perceptions of the students concerning various aspects of MENO-II.

The performance of students will be measured by their scores on a final test. Two types of skills will be tested for: the ability to recall factual information, and the ability to coordinate a number of concepts and apply them to problems not presented in the course. A problem of the first sort would be:

> Which of the following are syntactically correct examples of ATOMS:
>
> (a) ALPHA
>
> (b) ZBETA
>
> (c) ALPHA$BETA

Problems of this sort require knowledge of the syntactic, semantic, and pragmatic aspects of the subject area. A problem of the second sort would

be:

> Write a program which returns the nth item in list.

Problems of this sort require both the recall of syntactic, semantic, and pragmatic information and, more importantly, the coordination of these components.

The type of problem which we feel will test the degree to which students can transfer what they have learned is as follows:

> The following expression is a statement in the programming language SNOBOL. What do you need to know in order to understand it?
> Line WPAT = `EMPTY' :S(ALPHA)F(BETA)

This type of question can not be answered by rote factual recall. Rather, it requires that a student abstract from his/her understanding of LISP some general principles of programing languages and problem solving. Also, the student must abstract from the specific learning experience, the strategy or algorithm for "going about learning".

Standard statistical analysis will be carried out on the data collected from these tests. We anticipate that students exposed to one instructional style may do better on one test yet do worse on the other test, when compared with students exposed to the second instructional strategy. For example, a priori we feel that the Coaching style, which fosters student probing of the subject area, will increase students' transference ability, but possibly at a cost of insufficient practice and review in the subject matter itself.

We would also like to gather information on the preferences and attitudes of the students, and correlate that with the backgrounds and abilities of the students. As a tool for this analysis we may use the standard course evaluation questionaire used here at the University; of

course, we will modify that questionaire to reflect our interests in computer related aspects. Video-taped interviews may also prove insightful in this regard.

We will also gather data on various computational aspects of MENO-II. In particular, two types of information will be collected:

- utilization of computational resourses
- subjective observations on the requirements of the various components in MENO-II as a function of the instructional strategy.

It is clear that the cost of computer hardware is decreasing rapidly, and thus computer resources, such as "compute time," may need not to be optimized. However it may still be interesting to compare the utilization of such resources by the two instructional strategies. Besides a comparison of the efficiency of the strategies, resource utilization might show bottlenecks and problem areas in the instructional strategy itself; this type of analysis has become quite popular and useful in debugging and tuning other types of software systems (e.g., operating systems, language translators) [Svobodova 1976].

A more novel question to ask is: what requirements does a particular instructional strategy make on the other components in the ICAI system. For example, does the Socratic Tutor tend to require a better Dialogue Facilitator; are natural language capabilities necessary? Need the Competence Model be more structured for effective Socratic Tutoring than for Coaching? Since the Coaching strategy discourages explicit probing of the student, does this entail that a Coaching strategy requires a more sophisticated Performance Model? At this stage, answers to these questions may only be subjedtive; however, such judgments may be the basis of further, more empirical, evaluation.

In conclusion, we should remind the reader that we will be evaluating two computer programs, which embody two instructional philosophies; while we will of course try to remain as faithful to the philosophies in our implementation as possible, final rejection of one philosophy over another requires more evidence than we can muster in only two years. Nonetheless, if successful, we will have the first empirical evaluation and comparison of two state-of-the-art ICAI instructional strategies.

II. DELIVERABLES

II.1 Deliverables from Current Research

Four types of products have resulted from our current research:

1. The specification of the components comprising the Student and Author Environments in an IDEAL ICAI system.
2. The implementation of a rudimentary ICAI system, MENO-I.
3. An empirical evaluation of computer programming (the subject matter of MENO-I) as an aid to problem solving.
4. A set of reports documenting our results.

In this proposal, we have outlined the design of an IDEAL Intelligent Computer-Assisted Instruction (ICAI) system which will achieve the goal of CAI; high-quality, individualized instruction. Such a system will require a great deal of "intelligence"; it will need to understand the subject matter under consideration, the student's learning style and knowledge state, and possess strategies for instruction in order to perform at the desired high level.

The IDEAL ICAI system has served as a plan for the design and implementation of a particular ICAI system, MENO-I. In the following, we describe the software which comprises MENO-I and which is currently running on our computer here at the University of Massachusetts.

1. Competence Model: This component represents MENO-I's understanding of the subject area; it is a knowledge base consisting of concepts and relationships about programming and programming languages in general, and LISP in particular. In instructional terms, this knowledge base contains approximately 25% of a four week course on Computer Programming Using LISP. It is capable of making analogies to other subject areas (e.g., English, Mathematics, Chemistry) in order to accommodate differences in individual students. Besides this richness in breadth of knowledge, the Competence Model is also rich in its depth of understanding of the material; the key concepts are described at three different levels: syntactic, semantic, pragmatic.

2. Dialogue Facilitator: This module processes questions posed in restricted English by the student and accesses the Competence Model in order to respond with an answer. The list of permissable questions is meant to reflect good problem solving behavior; we have tried to identify questions which will reveal the essence of a concept. For example, one can ask "how is concept X used?" or "what does concept X do?" One can also ask for and receive examples of a concept. Or, the student can ask that his own example be checked; if the student's input is incorrect, the Dialogue Facilitator in concert with the Competence Model tries to explain why the student's statement was incorrect.

3. A Automatic Problem Solving System: In order to further identify problem solving knowledge for eventual inclusion in the Competence Model, we developed a Taxonomy of Problems. Currently, this taxonomy serves as the basis for a computer program which can automatically generate computer programs as solutions to problems presented by a user.

The Dialogue Facilitator and the Problem Solving System are written in LISP; the Competence Model is written in GRASPER, a graph processing language extension to LISP. The total package requires approximately 65K on a CDC 6600. Response time varies between 2-6 seconds depending on system load.

As an additional aid in explicating the tacit knowledge involved in problem solving and programming, we conducted several empirical experiments comparing the problem solving behavior of students in a computer programming context with students in a traditional algebraic (equational) context. The surprising, yet robust result, we obtained was that students do better solving word problems when they are asked for a computer program solution, than when they are asked for an algebraic solution --- even though the same solution is required!

Our work has produced a number of reports; a complete list is given in Chapter V. In order to collect and focus attention on our work, we have begun an ICAI Working Paper Series; we are currently in the process of establishing a mailing list and a distribution policy for these reports.

One working paper has already been submitted to a journal, while two others are being reworked for journal submission.

II.2 Deliverables from Proposed Research

Three types of products will result from the successful completion of our research objectives:

1. A working Intelligent Computer-Assisted Instruction System, MENO-II.
2. An evaluation and comparison of the effectiveness of two state-of-the-art instructional strategies: Coaching and Socratic Tutoring.
3. A set of reports documenting our results.

MENO-II will be capable of tailoring an instructional dialogue to meet the needs and abilities of different students. It will be composed of the modules in the Student Environment of our proposed IDEAL ICAI system: a Competence Model, representing the system's understanding of the subject matter (computer programming using LISP); a Performance Model, representing the system's understanding of the student's evolving knowledge state; a Dialogue Facilitator, a communications interface between the student and the Instruction Monitor; an Instruction Monitor and a data base of Instructional Strategies which monitor and direct the instructional dialogue.

MENO-II will serve as controlled testbed for the evaluation and comparison of two instructional strategies: Coaching and Socratic Tutoring. During a period of formative evaluation, student feedback from video-taped interviews, performance tests, and questionnaires will be used to tune and balance the components of MENO-II. For the summative evaluation, the group of students in the class will be divided, with half being exposed to MENO-II equipped with a Coaching strategy, and the other half being exposed to MENO-II equipped with a Socratic Tutoring Strategy. The comparison between the two groups will be based on tests of student performance in the subject

area itself, and tests of the ability to transfer learned skills to other subject areas. Also, quantitative measures of computational resource usage will be gathered and analyzed. The compatibility of the particular instructional strategy with the rest of the ICAI system will be appraised.

This research will generate reports which will be directed at a number of different fields. We plan to publish a description and analysis of the ICAI system itself in computer science/artificial intelligence journals such as Artificial Intelligence, International Journal of Man-Machine Studies, IEEE Journal on Systems, Man, and Cybernetics. We plan to publish the descriptions of the evaluation of the ICAI system in education oriented journals (e.g., Instructional Science). Finally, we plan to publish comprehensive articles on both the system and the evaluation in journals which are more interdisiplinary, e.g., Cognitive Science, International Journal of Man-Machine Studies. Journal articles will be preceeded by a working paper in our ICAI Working Paper series, and may also be presented at various conferences.