ARTIFICIAL INTELLIGENCE APPLICATIONS TO COMPUTER-ASSISTED INSTRUCTION

Project Progress Report No. 2:

WHERE AI CAN FIT IN CAI

Jesse M. Heines, Ed.D.
Staff Consultant

Systems Based Courseware
Educational Services Development & Publishing

November 4, 1982

d i g i t a l

Educational Services
Bedford, MA 01730

1.0 ABSTRACT

This report discusses some of the AI/CAI research I have pursued since I began my assignment Britain's Open University [1]. The basic purpose of my assignment here is to investigate the applicability of artificial intelligence to improving

- the quality of CAI courseware and/or
- the efficiency of the CAI course development process.

The first component of my research has been to try to identify the different ways in which AI can be applied to CAI, that is, where it can "fit". So far, my research has identified examples of systems that apply AI to the building of

- natural language parsers to enhance the student/computer interface,
- powerful programming systems that "know about" the language or application being programmed,
- sophisticated subject matter databases that can be used to generate highly adaptive tutorials in real time and/or that students can query directly,
- models of student behavior that give programs superior ability to adapt to individual student differences,
- rule-based tutorial strategies that can be adapted to different target populations, and
- self-improving tutors with the ability to adapt their own tutorial strategies to maximize teaching goals.

This is, of course, only a partial list of the entire AI/CAI domain, but it is a fair representation of those applications mentioned most often in the literature.

This report discusses each of these applications in turn and comments on their feasibility for influencing CAI development in Educational Services, particularly in relation to the above stated purpose of my assignment. It concludes by expressing ideas for the types of AI/CAI efforts that would be most profitable for Educational Services to pursue at this time, and outlines my current research plans for the remainder of my tenure at The Open University.

---

[1] For information on AI/CAI work that I did prior to my assignment at The Open University, please refer to AI/CAI Project Progress Report No. 1 entitled "Basic Concepts in Knowledge-Based Systems" and dated April 20, 1982.

2.0 TABLE OF CONTENTS

3.0 EDUCATIONAL SERVICES' CURRENT CAI SYSTEM

Educational Services' current CAI system may be represented as shown in Figure 1. This system consists of authoring tools that are only used internally and courseware components that make up the software package ultimately shipped to customers.

CAI authoring tools have been described in a recent extensive study by Randy Levine, "Authoring Tools for Computer-Based Instruction: Review and Recommendation" (1982). Levine discusses eight such tools, the following four of which are currently in use in Educational Services.

- DRAW

  A visual screen and graphics editor for creating libraries of predefined screen displays that are callable from standard programming languages via a PLOT routine. (The libraries thus created are stored in .DLB files as indicated in Figure 1.)

- TEMPLATE

  A set of DCL indirect command files and TECO macros that set up the skeleton of a CAI course for VAX/VMS systems. TEMPLATE sets up a complete course in "press RETURN to continue" mode with dummy displays in DRAW format. The developer edits the program files generated by TEMPLATE to call his or her own displays, perform response judging, and branch appropriately.

- GRAIL

  A small interpreter of basic CAI function commands that can perform most of the functions generally used in TEMPLATE-generated courses. GRAIL runs as an interpreter and was originally developed to save program testing and debugging time by eliminating the need to recompile and relink lesson programs each time small changes were made. Its interpretive nature and small size have since proved valuable for implementing CAI on small machines such as the new PROFESSIONAL series.

- OFAL

  Another interpreted CAI language, but with significantly more programming power and structure than GRAIL. This language has been used for OFIS courses in Reading but is not currently being used for any active course development. It is, however, serving as the basic model for a new interpretive language intended to make TEMPLATE, GRAIL, and OFAL all obsolete.

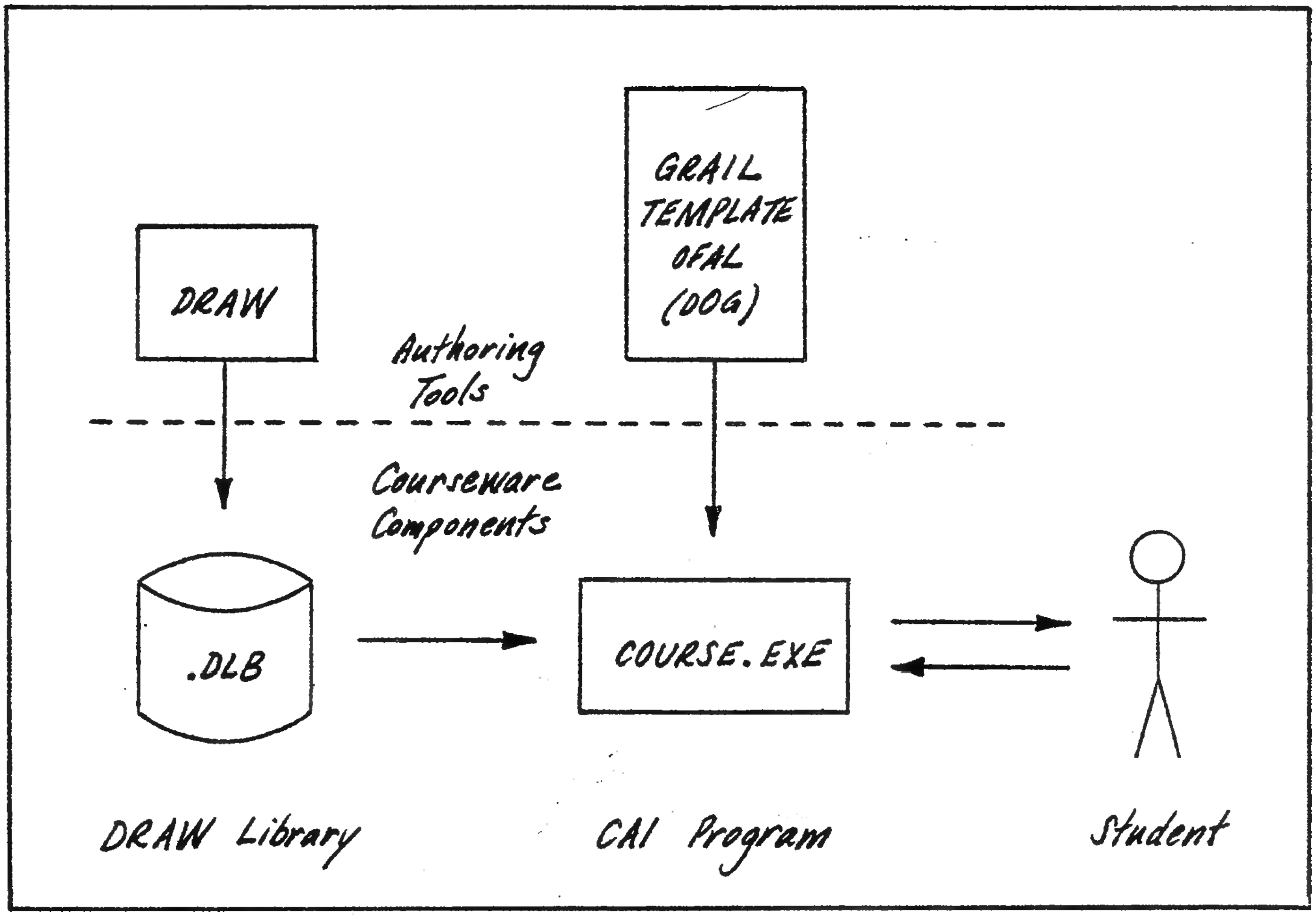

Figure 1

CURRENT EDUCATIONAL SERVICES
CAI DEVELOPMENT SYSTEM

The new language currently under development is called DOG, an acronym for "Daughter of OFAL and GRAIL". DOG is not yet operational, but extensive development efforts are underway. The software team developing DOG is also working on a specification for what they call the DOG "house", a system intended to provide authors using DOG with an enriched programming environment and extensive resources for help.

The three major recommendations in Levine's report are that Educational Services should:

(1) develop a new interpreted authoring language that incorporates the positive features of our earlier authoring languages and corrects most of their shortcomings (this effort is currently underway as the DOG project),

(2) maintain DRAW as the basic tool for creating screen displays (a new, more powerful and more human-engineered version of DRAW is currently under development), and

(3) develop standards for CAI coding.

These recommendations should help increase the efficiency of CAI course development in the following ways. First, course developer training time should be reduced as the result of adopting a single, well-designed CAI authoring language that is runable on all systems. The adoption of a single authoring language should also greatly simplify the implementation of CAI coding standards. Second, courseware conversion costs should be reduced because new systems can be supported by implementing an new interpreter rather than having to design entirely new CAI programming techniques for each new system on which we are asked to deliver courseware. Third, software support costs should be reduced as the result of implementing all courseware in the same language.

None of Levine's recommendations will change the basic flavor of the CAI courseware that Educational Services is currently producing. DOG and its complementary "house" should increase developer efficiency considerably, but the types of programs produced with these tools will be essentially the same as those currently in existence.

This report looks at how AI can be applied to change the basic flavor of our CAI courseware. It examines the different points at which AI technology can be applied, both in the development cycle and in the resultant courseware. The report discusses the changes that could be brought about by each application, and examines the feasibility of producing the required software.

4.0 AI APPLICATIONS TO STUDENT/COMPUTER INTERFACES

Our current major interfaces between students and the computer are the PLOT routine for putting DRAW displays onto the screen and the GETANS routine for handling student input. Both of these routines have proven themselves to be highly robust in actual use and easily adaptable to a wide range of CAI interaction strategies.

If we were to apply AI techniques to enhance these interfaces, the target of that effort would be as shown in Figure 2. The PLOT routine might be enhanced to allow more computer-generated output, while the GETANS routine might be enhanced to accept a wider range of student input. The discussion that follows focuses on the latter of these applications.

4.1 Natural Language Interfaces

The most dramatic way in which AI can be applied to student/-computer interfaces is through the implementation of a natural language interface. A large number of linguists have been involved in basic AI research for many years, and several viable natural language parsers exist. While some of these parsers appear to exhibit startingly intelligent understanding of English sentences, the subject matter domains to which they have been applied are usually quite limited. Nonetheless, the ability of natural language parsers to enrich the student/-computer interface cannot be ignored.

4.2 The Classic Example: SOPHIE

The most famous example of a natural language interface in CAI is the SOPHIE program developed by John Seely Brown, Richard Burton, and Alan Bell (1974) at Bolt Beranek and Newman, Inc. This program provided an interactive environment in which students could practice trouble-shooting a faulted instrument given a diagram of its circuit. Reproduced below is an excerpt from a sample program run published in their paper. All computer I/O is printed in upper case. Student input is underlined and preceded by the symbol >>. The annotations printed in mixed case are by Brown, Burton, and Bell.

Assume that the student has been given an off-line copy of the circuit diagram and a list of the present control settings for the faulted instrument. The student begins by asking questions to isolate the fault.

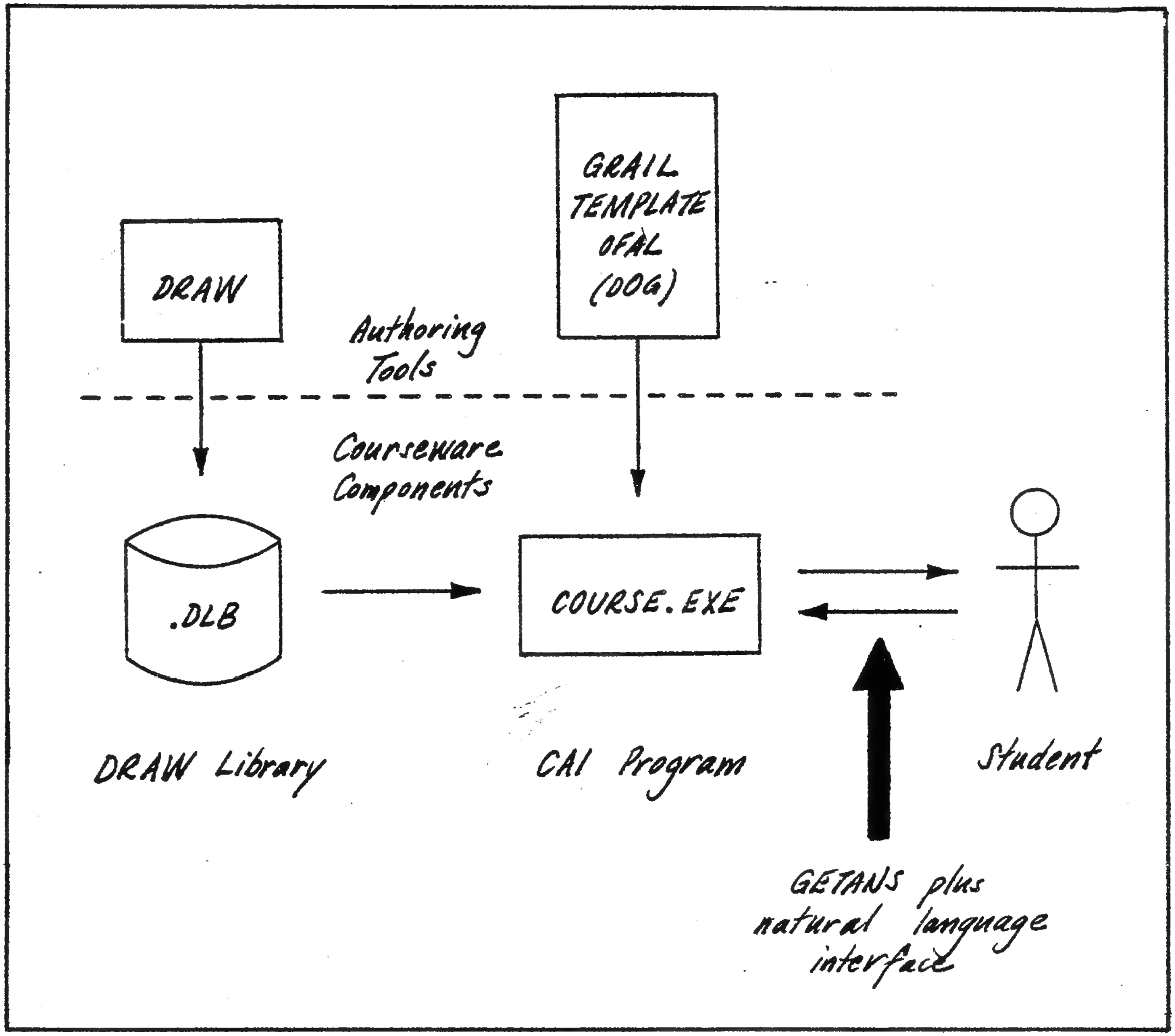

Figure 2

APPLICATION OF AI TECHNIQUES TO
STUDENT/COMPUTER INTERFACES

>> WHAT IS THE OUTPUT?

THE OUTPUT VOLTAGE IS 11.7 VOLTS.

>> WHAT IS IT IN A WORKING INSTRUMENT?

IN A WORKING CIRCUIT THE OUTPUT VOLTAGE IS 19.9 VOLTS.
THAT VALUE IS INCORRECT.

In case the student does not know what the value of a measurement should have been, he can have any measurement repeated in a working instrument. Note the student's use of the pronoun "it" to refer to the entire previous sentence.

>> WHAT IS THE VOLTAGE BETWEEN R8 AND Q1?

THE VOLTAGE AT N1 IS 47.6 VOLTS.

>> WHAT IS THE BSE EMITTER VOLTAGE OF THE VOLTAGE LIMITING TRANSITOR?

= BASE
= TRANSISTOR

THE BASE EMITTER VOLTAGE OF Q5 IS -0.037 MILLIVOLTS.

The spelling corrector has detected and corrected two misspellings. The system must also know that Q5 is the voltage limiting transistor.

>> WHAT ABOUT FOR THE CURRENT LIMITING TRANSISTOR?

THE BASE EMITTER VOLTAGE OF Q6 IS 0.349 VOLTS.

The meaning of this question is completely dependent on the context established by the prior sentence. The system uses this context to determine that he is interested in, not only voltage measurements, but the voltage across a particular junction.

>> WHAT SHOULD IT BE?

IN A WORKING CIRCUIT THE BASE EMITTER VOLTAGE OF Q6 IS 0.593 VOLTS.

Another way of requesting the prior measurement to be repeated in a working instrument, i.e., the meaning of "should" implies that a working instrument should be used.

SOPHIE uses AI techniques in a number of ways, the natural language interface being but one of these. The approach to implementing the parser was somewhat novel, in that it worked by "refining the usual syntactic categories such 'noun phrase' into relevant semantic/conceptual categories such as 'measurement'." The authors claimed that "for our highly constrained domain, this approach is viable."

The key word in the above statement is "constrained". It is clear that a interface such as SOPHIE's could be built for more general subject matter domains, but only at considerable cost. The authors felt that "more complex parsing systems would have helped SOPHIE appear more natural", but opted for trade-offs that allowed them to implement an extensive abbreviation handling capabilities, a spelling corrector and separator for run-on words such as "whatis", and context-dependent references. The paper explains that this last capability enabled the program to handle such entities as:

WHAT IS THE CURRENT THROUGH THE BASE OF Q6?

WHAT IS IT THROUGH THE EMITTER?

"It" refers to "current" and "Q6" is implied but not mentioned.

WHAT ABOUT THROUGH THE COLLECTOR?

In this case, "current" and "Q6" are both implied but neither is mentioned.

4.3 Implications for Educational Services

Like SOPHIE, Educational Services CAI courseware generally deals with relatively restricted subject matter domains. It appears that it would be possible, therefore, to build a natural language interface of this type, at least for courses running on large machines. But the task is not a trivial one: Brown <u>et al</u>. (1974) state that "SOPHIE represents approximately 300K words (36 bit) of INTERLISP and FORTRAN code running on a virtual memory TENEX" (DECsystem-10), although only part of this represents the natural language interface. "The natural language interface to SOPHIE," they report in another paper (1982), "took approximately two man-years of effort, and evolved over the course of four years."

The CAI courses we have developed to date have not had much call for such powerful interfaces. We have had much more need to create parsers for DCL, ReGIS, and EDT command lines than for natural language. Some may argue that this is due to our course designs, but others would argue it is due to the nature of the materials we teach and the types of interactions that we want students to practice on-line. Whatever the reason, the size of the effort needed to create such an interface and the lack of extensive application for it do not seem to warrant applying our resources to this task at this time, especially since the result of such an effort would most likely be highly domain-specific and therefore only be applicable to a small number of our courses.

5.0 AI APPLICATIONS TO CAI AUTHORING TOOLS

The target of AI applications to CAI authoring tools would be as shown in Figure 3. In this case, AI might not appear in the courseware components at all. It would be an authoring tool for course developers that provides powerful facilities and valuable assistance in the creation of CAI courseware.

5.1 AI as a Programming Environment

One of the most striking characteristics of environments in which AI programming takes place is the power of the programming tools. As soon as I began learning LISP here, I was introduced to EMACS, its companion editor. EMACS is not just another screen editor in the style of TV and EDT. EMACS "knows about" LISP and LISP structures and is a significant factor in increasing the efficiency of LISP programming. It can format functions, identify incorrectly nested parentheses, and interpret abbreviations for standard LISP tokens.

EMACS also takes advantage of the facts that LISP is usually implemented as an incremental compiler and that LISP programs are generally built out of a large number of functions. These characteristics allow most LISP programming bugs to be corrected by changing the code in a single function rather throughout an entire program. During the debugging process, therefore, programmers can "mark" those functions that they wish recompiled. On exiting EMACS, the marked functions are automatically compiled and linked into the user's program. This process generally takes but a few seconds, and the new version is ready to be tested.

This LISP/EMACS integration forms an extremely powerful and highly productive programming "environment". After riding for years on the

```
EDIT -> COMPILE -> LINK -> TEST
 ^___________________________v
```

merry-go-round, I found that the shortened

```
EMACS -> LISP
 ^__________v
```

cycle greatly increased the speed with which I could generate and debug code.

An even more tightly coupled editor/incremental compiler system has been developed by Steve Hardy and Aaron Sloman (1982) of Sussex University (Brighton, England). This system,

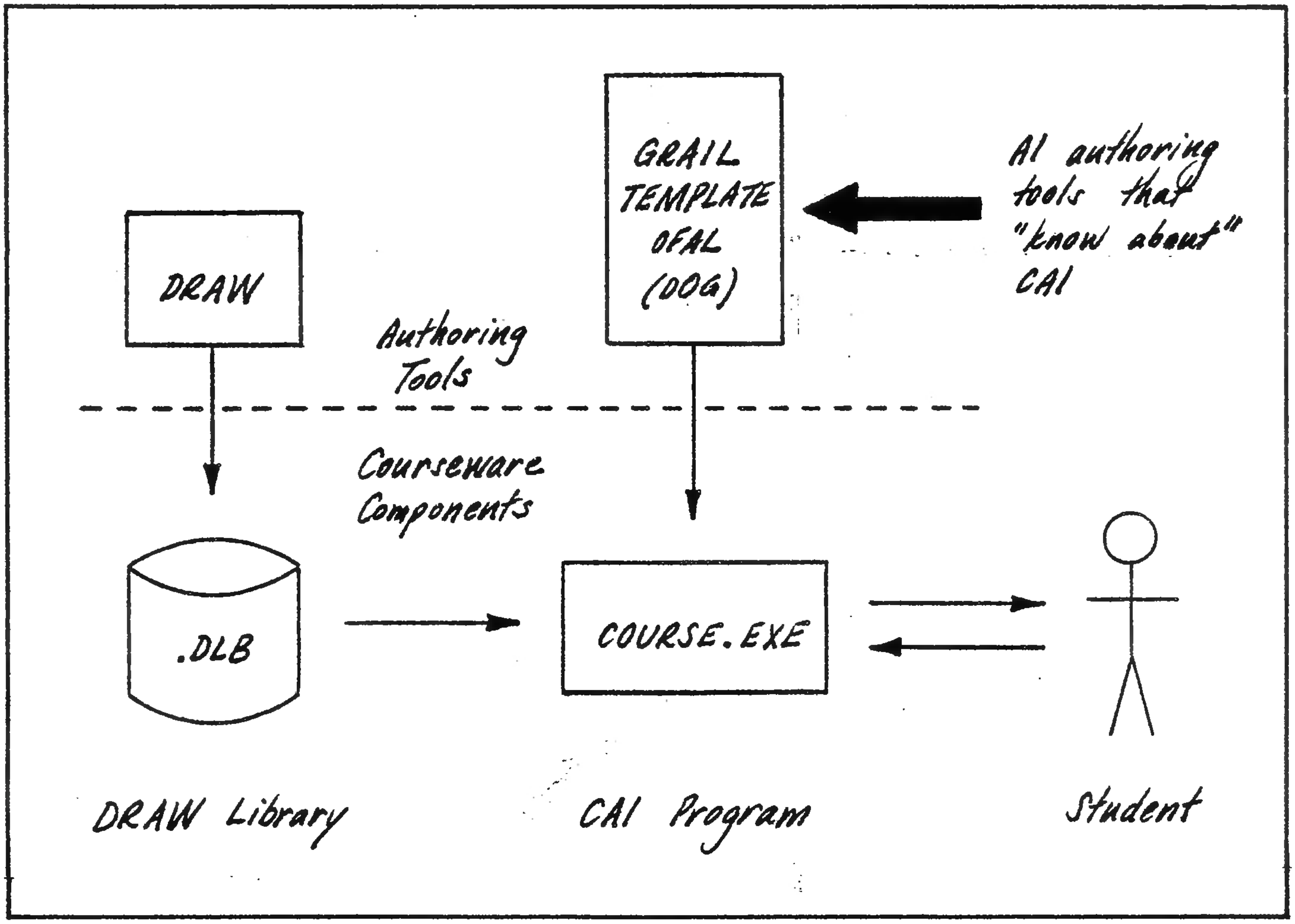

Figure 3

APPLICATIONS OF AI TECHNIQUES TO
CAI AUTHORING TOOLS

called POPLOG, runs under VAX/VMS Version 3.0 and includes incremental compilers for both POP-11 and PROLOG. The POPLOG system is more tightly coupled than EMACS/LISP in the sense that all system functions are available at all times. For example, one can call editor functions from a POP-11 program to format the screen using commands such as "open line", "delete word", etc. Since the editor itself is written in POP-11, it is also highly extensible. One need simply write a new function for the editor and incrementally compile that one function to add it to the editor's capabilities.

The type of environment described above need not be restricted to languages used for AI. There is no reason why other languages can't be integrated as incremental compilers, although certainly function-oriented languages make the task easier. No matter what authoring language we adopt for CAI course development, this language would be far easier to use if it could be implemented as an incremental compiler and be married to an intelligent editor. Such an approach would surely increase our course development efficiency many-fold and allow our programming tools to stand head and shoulders above those of most other industrial CAI development centers.

5.2 AI as a Development Coach

A more application of AI to authoring tools involves imparting the authoring tools with some knowledge of the domain of CAI. This might be done through sets of production rules like those used in expert systems or direct coding. Given this knowledge, AI could function as a very sophisticated authoring help facility and/or as an expert monitoring a developer's work that interrupts at opportune moments when it detects that the author is in need of advice. Each of these applications is discussed in more detail below.

5.2.1 **Intelligent Help Facilities** - Wilensky (1982) has reported on efforts to create a natural language help interface for UNIX. This system, called UC for "UNIX Consultant", exhibits many of the same qualities as SOPHIE, specifically understanding of pronouns and implied ellipses. Wilensky's system basically tells users the correct syntax for UNIX commands. A sample dialogue is as follows:

User: How do I print the file FETCH.L on the line printer?

UC: To print the file FETCH.L on the line printer type "LPR FETCH.L".

User: Has the file FETCH.L been printed yet?

UC: The file FETCH.L is in the line printer queue.

User: How can I cancel it?

UC: To remove the file FETCH.L from the line printer queue you must type "LPRM FETCH.L".

Eisenstadt (1982) feels that Wilensky's efforts are not really worthwhile, for if the system can be made intelligent enough to understand the queries, it might as well execute the commands directly. That is, users should not have to ask "how do I print the file FETCH.L on the line printer?", be told to "type 'LPR FETCH.L'", and then have to type "LPR FETCH.L". If the system can deduce "LPR FETCH.L" from the user's query it might as well just pass this command to the operating system. Interestingly enough, this is exactly what appears to happen in the second query in the above sample dialogue. The user asks "has the file FETCH.L been printed yet?", and the system reponds that it "is in the line printer queue". It does not tell the user to type another command to list the contents of the line printer queue.

While Eisenstadt's point may certainly be a valid one, it is also clear that Wilensky's techniques could be used to get help on much wider subject domains. For example, a system with knowledge of the CAI domain and an understanding of the current screen display might be able to answer author queries of the following type.

- The advice I want to give students at this point will not fit into the standard bottom three lines. What other options do I have?

- If students press the HELP key while viewing this screen, how many forms back will they be positioned when they return?

- What sequence of commands will wipe the screen to reveal a videodisc image?

- How can I create a menu using graphic entities rather than text?

- What form would be displayed to students if they pressed the advice key while viewing this form?

- What forms were overlayed to generate this image?

A system of this type would be expensive to produce. Like other natural language systems, it does not seem worthwhile for Educational Services to pursue at this time.

5.2.2 **Intelligent Monitoring Facilities** - Intelligent monitoring systems, on the other hand, offer more direct applications to the type of system represented by DOG (see Section 3.0). In these systems the computer monitors user actions looking for patterns of incorrect or inefficient command usage. When such patterns are found, the system interrupts and offers the user advice.

A system of this type designed specifically for VAX/VMS has been described by Shrager and Finin (1982). Their system detects several types of inefficient DCL command sequences. For example, it can deduce that if the user enters a sequence of PRINT commands for each file with a specific extension, it would have been more efficient to use the PRINT command with a wildcard in the file name. On another dimension, the system can deduce that actions taken by command sequences such as

```
$ COPY NOTES.* OLDNOTES.*
$ DELETE NOTES.*.*
```

could have been more efficiently carried out with single commands (RENAME in this case).

Another system of this type is currently being implemented by Tony Hasemer (1982) at The Open University. Hasemer's system looks for "clichés" in student programs as an aid to debugging. His system currently operates on programs written in SOLO, a small relational database language used to teach students AI concepts in a psychology course. The two major differences between Shrager's work and Hasemer's are (1) that the former operates interactively while the latter is intended for use after the program has been written, and (2) that the former is designed to look for a number of general patterns simultaneously while the latter looks for specific patterns unique to a model of a correct program.

Shrager's system recognizes only five different patterns of inefficient command usage, and Hasemer's current system is tailored specifically to one model SOLO program. Recognition of a wide range of patterns, therefore, appears to be quite difficult. Like natural language systems, the building of a system viable for Educational Services use may be technically feasible, but the task would be arduous. Unlike the limited payback of natural language systems, however, the payback of intelligent monitoring facilities could be very large, specifically in regards to the implementation of CAI coding standards as recommended by Levine. This AI application also has implications for building intelligent tutoring systems, as described in the next section.

6.0 AI APPLICATIONS TO TUTORIAL PROGRAMS

AI applications to tutorial programs can be represented as shown in Figure 4. This application requires major changes to the CAI program itself, as represented by the additional facilities that the .EXE program will have to access. These facilities include a subject matter database, a student model, and a rule-based tutorial strategy. Applying AI in this manner could result in basic changes to the type of CAI courseware that Educational Services is currently producing. A number of prototype programs already exist, and some of the concepts embodied in these systems are presented below.

6.1 Subject Matter Databases

Tutorial programs that interface to subject matter databases (SMDBs) generally construct interactions from logical inferences they draw from the database. Some of these systems also allow students to query the database directly to ascertain both facts and relations.

An intelligent tutor interfaced to a subject matter database might be conceptualized as shown in Figure 5. This structure resembles the one used in GUIDON, an intelligent CAI program built on a MYCIN-like expert system (Clancey, 1982). The SMDB in these types of programs typically has a complex relational database structure like that in AI production rules. The intelligent tutors needed to access such SMDBs are usually large and run on mainframe systems.

As discussed in the my first AI/CAI Project Progress Report, the building of these databases is often a more difficult task than writing the programs to access them. For this reason, a substantial amount of research is now going on in various cognitive science centers to understand the task and bring it down to manageable proportions. If we were to decide to try to implement an intelligent tutor of this type, it would probably be wise to invest time to build an intelligent authoring tool to simplify the task of building the database. Such a tool would interact with a subject matter expert as shown in Figure 6.

An intelligent tutor of this type would definitely not run on a small system. However, Steve Hardy (1982) of Sussex University has suggested that it might be possible to take advantage of a technique developed by Gerald Sussman (1974) to get the system down to a size that could run on a smaller machine. Sussman developed a program called "HACKER" that transforms a small declarative database that must be accessed by a "smart" program into a larger but conceptually simpler

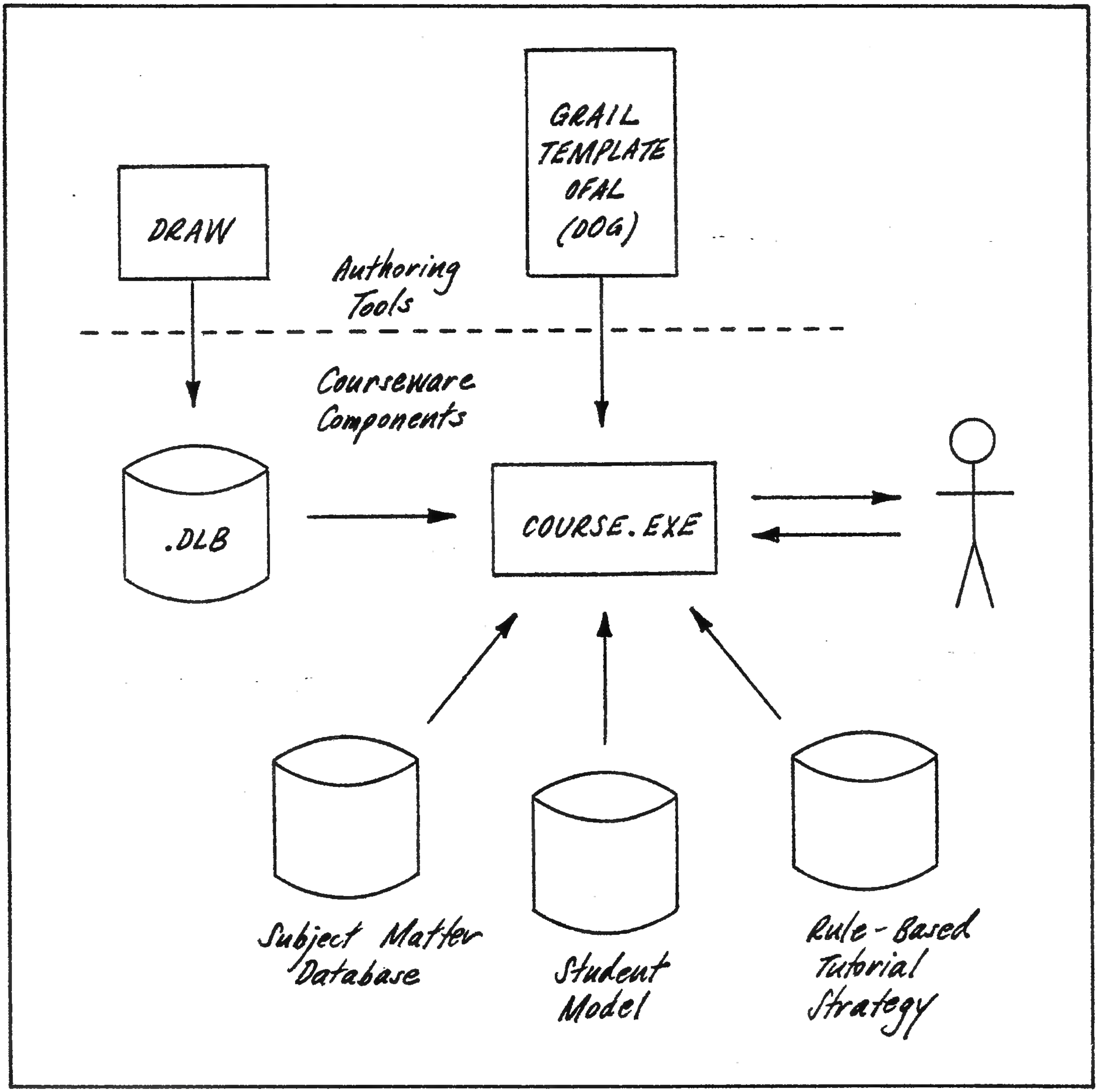

Figure 4

APPLICATIONS OF AI TECHNIQUES TO
TUTORIAL PROGRAMS

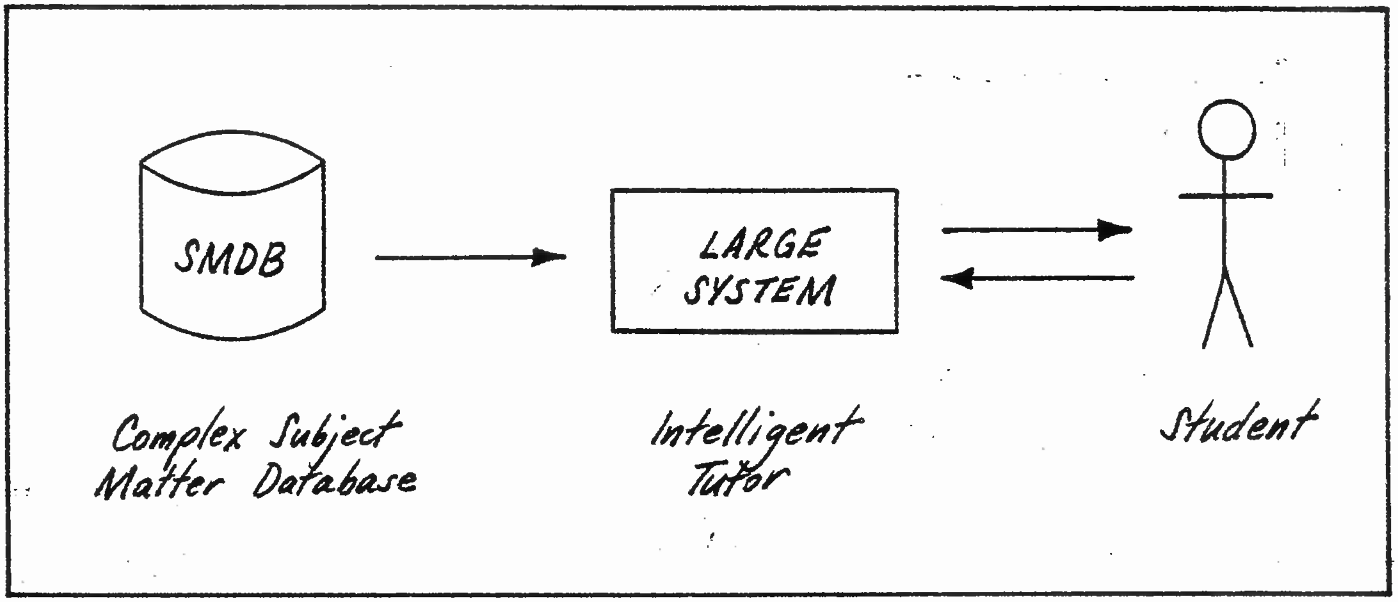

Figure 5

AN INTELLIGENT TUTOR INTERFACED
TO A SUBJECT MATTER DATABASE

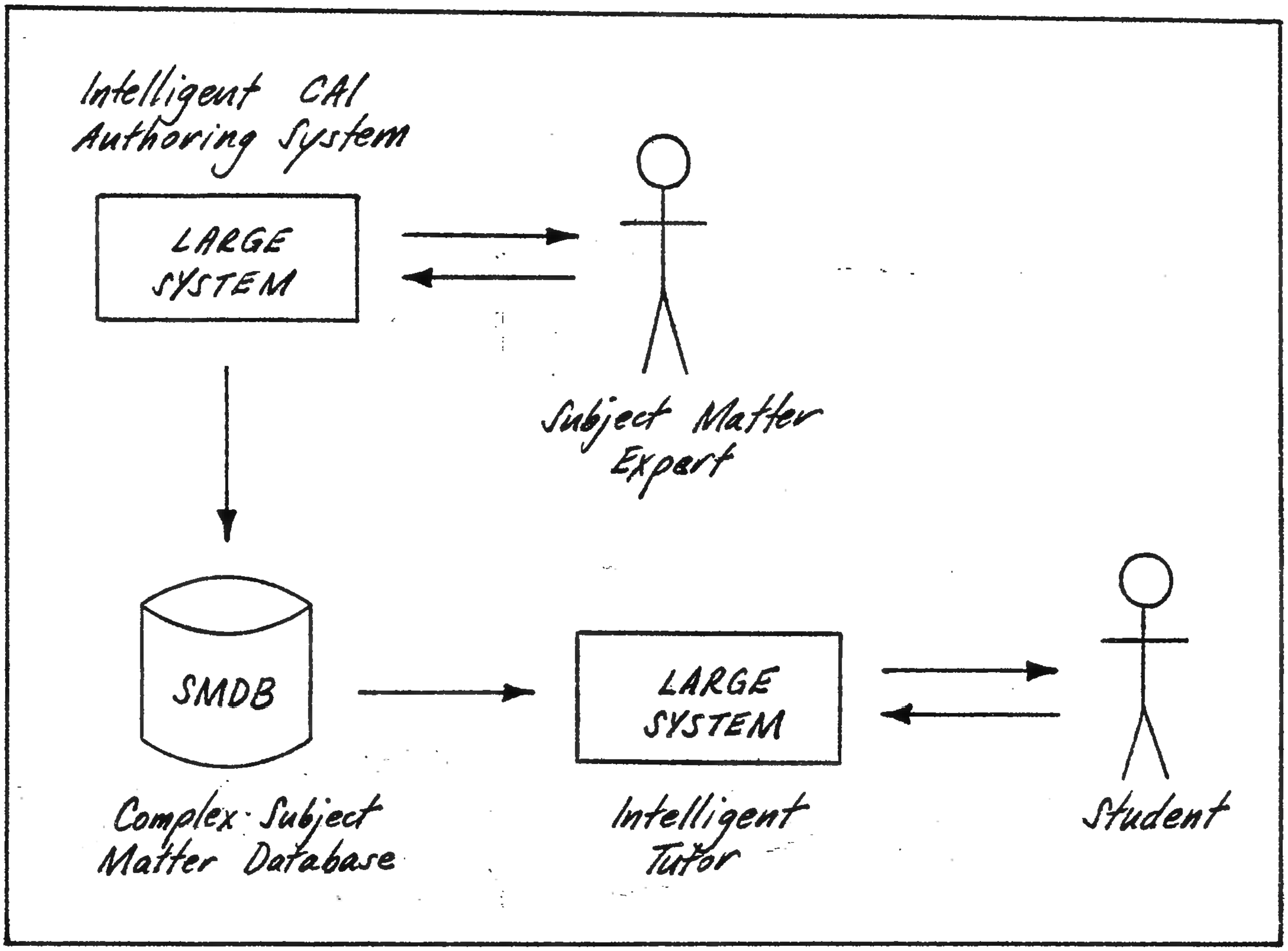

Figure 6

A SOFTWARE TOOL FOR BUILDING
A SUBJECT MATTER DATABASE

procedural database that can be accessed by a more conventional program. The key to Sussman's work is that the resultant procedural database possesses all of the information in the original declarative database, but in a form simple enough to be used by a much smaller driver program. This technique, if feasible, might result in the CAI system depicted in Figure 7.

6.2 Adaptable Student Models

The second basic component of intelligent tutoring systems is an explicit student model (see Figure 4). This model is the key to these systems' responsiveness to individual student differences, in short, to their adaptability. Tim O'Shea (1982) explains that typical CAI programs "adapt to success", that is, they go on to higher levels of abstraction or problem difficulty when students master more elementary sections. The fallacy in this approach, he asserts, is that it is usually wrong to assume that problems can be graded from hard to easy and that problems can be stated so that they test discrete areas of students' misunderstandings. More often, students have problems caused by a combination of misunderstandings, and the interactions between these usually cannot be expressed hierarchically. Stevens, Collins, and Goldin (1982) have demonstrated this phenomenon empirically in studies at Bolt, Beranek & Newman, Inc.

Consider, for example, a simple task that has two component concepts, A and B. A program program that adapts to success might be set up to teach students Concept A and then go on to Concept B when Concept A is mastered. Such a program ignores the connections between the two concepts. That is, a student might be able to pass tests that ostensibly evaluate each of the concepts individually, but not a test that evaluates situations in which the concepts interact. O'Shea contends that far more subject areas exhibit connected concepts than disjunct ones.

AI student models attempt to express students' skills in a manner that matches their actual behaviors much more closely. These models must be sufficiently complex to account for concept interaction and partial acquisition of skills. Indeed, the ideal student model would be one that could predict student behavior on any specific class of problems. There are two advantages to building student models at this level. First, tutorial programs can be tailored by adjusting the student model without affecting any of the other program components. Second, the existence of an accurate student model and a similarly structured model of the subject matter to be learned can yield considerable insight into the tasks one should teach to fill in the concepts missing in the

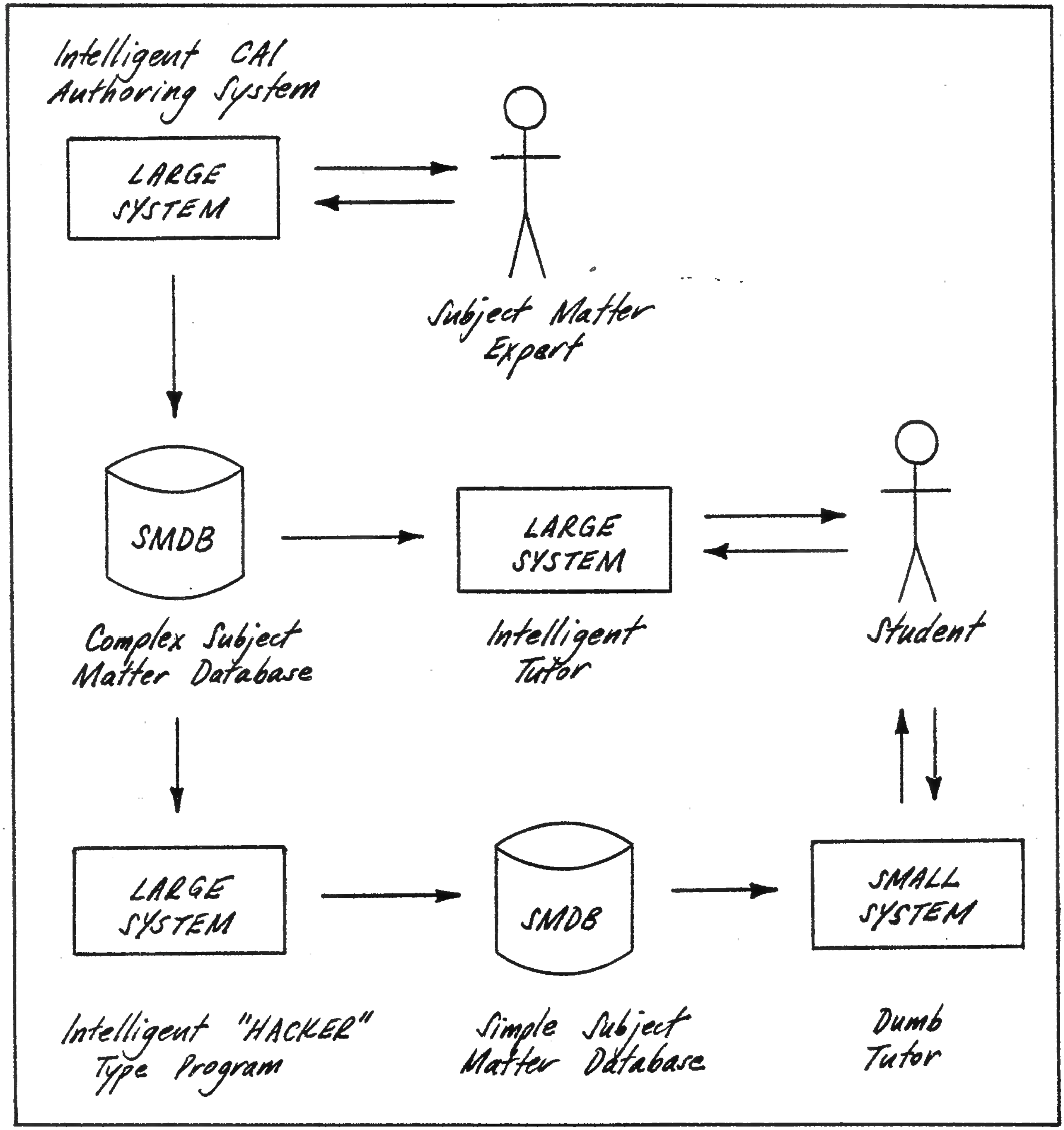

Figure 7

AN INTELLIGENT CAI SYSTEM INCORPORATING A "HACKER" PROGRAM

(after Hardy, 1982)

student's understanding. While a number of programs have been successful in demonstrating the validity of student models in limited domains (Burton and Brown, 1982), a large amount of work remains to be done in this area because even non-computerized models of students' cognitive processes are still relatively primitive.

6.3 Rule-Based Tutorial Strategies

The final basic component of intelligent tutoring systems is a rule-based tutorial strategy (see Figure 4). Like the student model, isolating the tutorial strategy allows CAI programs to be tailored by adjusting the way or the order in which materials are presented without affecting other characteristics of the program.

Some of the most extensive work on rule-based tutorial strategies has been done by O'Shea (1979). I described the basic concepts of O'Shea's work in AI/CAI Project Progress Report No. 1 (see pages 35-38 in that report), so I will not repeat them here [2]. My previous report did not, however, discuss the self-improving aspect of O'Shea's system, a characteristic that allowed the system to increase its adaptability to a particular group of students. The system's tutorial strategy was expressed as a series of production rules that governed program branching based on student histories and their mapping on an explicit student model. A number of discrete program goals were defined, and the system tried to maximize its achievement of these goals without negative side effects. (A negative side effect was defined as increasing its ability to achieve one goal while dramatically decreasing its ability to achieve the other goals.)

With its tutorial strategy expressed as a series of production rules, this system was able to alter its teaching strategy without affecting other program parameters. The truly fascinating aspect of the program, however, what that it performed and evaluated experiments itself based on the given instructional goals. Changes that resulted in positive improvements in teaching performance (as measured by _t_-tests of students' scores) were automatically incorporated into the teaching strategy, which changes that resulted in deteriorated teaching performance were discarded.

---

[2] Two errors in that report should be corrected. First, O'Shea is Director of the Open University's Microcomputers in Schools project, not its Institute of Educational Technology. Second, O'Shea's system _does_ (or did) exist: it was implemented in LISP and ran on a CDC 6600/6400 at the University of Texas at Austin.

6.4 Intelligent Instructional Interfaces

When it comes to using CAI teaching students about computers as we do in Educational Services, the three AI applications described above must be evaluated in slightly different terms than those in which they were originally developed. John Ulrich of DEC's Knowledge Engineering Group sees two main problems with the type of CAI embodied in courses such as EDTCAI. First, he feels that students may have trouble understanding these courses because they teach things before students fully appreciate the value of the material being taught. Second, he feels that the instructional context of these courses is often significantly different from that in which students will eventually work.

As a remedy to these problems, Ulrich has suggested that we build what he calls an "instructional interface". Such an interface would be similar to that described by Shrager and Finin (see Section 5.2.2 of this report), but it might go off into CAI sequences if students so requested rather than just presenting short help messages as in Shrager's and Finin's system. Ulrich feels that putting this interface between naive users and the operating system or common utilities would provide an effective way to teach them only what they need to know and to do so in the same context in which those skills will eventually be used.

This idea is an interesting one, though rather ambitious. It would involve essentially the same problems encountered by Shrager and Finin, and I am doubtful as to whether significant problems in student's misunderstandings could be identified simply by monitoring their work in this manner without incorporating a number of queries to refine the system's assertions. Ulrich's concerns are worth considering, however, and might be addressed by implementing a slightly modified concept.

Assume for a moment that the EDT editor could be modified so that (1) it ran in a software-definable screen window, and (2) it could be run as a subprocess controlled by a program that could monitor all of its I/O. (For a discussion of the implications of these capabilities, see Zimmerman and Heines, 1981.) If these modifications could be achieved, it would be possible to let students run EDT in a corner of the screen and interact with them about their activities in other parts of the screen. The monitoring program would then form an intelligent instructional interface like that suggested by Ulrich. It could trap inefficient command uses like those described by Shrager and Finin, but it could also assign users sample tasks that they would perform in an environment matching that of EDT itself very closely. When users first begin using this interface the EDT window might be small. As they become more proficient, the window could be allowed to grow until it filled the entire screen and the instructional interface was essentially removed.

Such an approach would have distinct advantages over most current AI/CAI systems because it could take advantage of the techniques that have already proven themselves effective in our current CAI materials. The most important of these techniques is graphics. Virtually none of the classic intelligent tutors have made any use of graphics, although this fact is quickly changing due to the capabilities of the Symbolics LISP machine (Stevens, 1982) and its competitor at Xerox.

One graphics application in the intelligent instructional interface for EDT described above might make the contents of the cut and paste, word, and character buffers visible for new EDT users outside the main editor window. (Wendy Mackay first implemented this visual technique in the original EDTCAI course.) Implementing the interface on a VT125 would also allow it to do things such as pointing to user errors via the separate graphics plane without disturbing the editing context in the text plane. This effect would be similar to having a human tutor point to something on the screen as he or she explained the problem.

The combination of these facilities with an effective student model and tutorial strategy could be very powerful. Tasks such as editing can be easily modelled and their component activities easily quantified. Ulrich suggests that one could build models of naive users, users with 2-3 weeks' experience, and intermediate/advanced users and tailor the tutorial strategies and amount of hand-holding accordingly. Though by no means trivial, implementation of a program of this type should be relatively straightforward, assuming that EDT could be interfaced to a CAI program with the required levels of windowing, I/O filtering, and control facilities.

7.0 RESEARCH EFFORTS TO BE PURSUED

My work at The British Open University has actually focused more quickly on specific applications than I at first thought it would. The central location of The Open University has also put me in touch with faculty at other British colleges and universities, including Brunel, Sussex, Loughborough, Edinburgh, and Imperial. It has been easy for me to find researchers to exchange ideas with, and this fact has helped put a lot of my reading of the literature into perspective.

Tim O'Shea has arranged for me to have direct access to the University's DECsystem-20, and, to my very pleasant surprise, the Computing Services staff has loaned me with a GIGI terminal in exchange for some help with ReGIS. The DECsystem-20 here has three flavors of LISP and versions of PROLOG and OPS, so I have been able to study these AI development tools as well. Given the availability of these tools and the steadily refining focus of my research, I now plan to try to do some prototype implementation during my stay here rather than just planning to write a specification for incorporating AI into Educational Services current CAI activities. Efforts in this area should make it easier for me to write a specification from a more knowledgable and experienced position when I return to my post in Bedford next April.

My current plans are therefore as follows:

- continue reviewing the AI/CAI literature and studying AI languages (concentrating on LISP) through the end of December,
- focus in on a very small, prototype AI/CAI implementation project that can be completed by the end of my stay here and have this project completely defined and specified by the end of January, and then
- begin writing code for the prototype software by the beginning of February, incorporating graphics wherever feasible.

My current thoughts on a prototype project involve looking at two areas: a course on editing as discussed in Section 6.4 and a course on ReGIS graphics programming. Editing appears to have a wider initial audience, but the systems problems involved with windowing and I/O filtering when EDT is run as a subprocess appear to be exceptionally difficult to tackle. The required systems software can be much more easily built for a course on ReGIS due to its more limited subject domain. Tackling this subject matter therefore looks like it would allow greater concentration on incorporating AI techniques into the course's presentation and exercise routines. Experience gained in this area might then be applied later to developing a larger graphics course to teach the ACM SIGGRAPH Core standard or the new Professional CTIG subroutine calls. Either of these could form the basis for an attractive Educational Services product once AI/CAI techniques have been sufficiently developed.

It is impossible to say at this point whether either of these prototype projects could be completed within my tenure here (I return to my post in Bedford in April, 1983). But if it is not, it should be possible for me to continue and complete the work in Bedford because I will be working on all DEC equipment here. I anticipate that my next AI/CAI Project Progress Report will therefore be written in January, 1983, and that it will contain further reviews of the AI/CAI literature as well as a specification for the prototype implementation project I have decided to pursue.

8.0 REFERENCES CITED AND RELATED READINGS

Brown, John Seely, Richard R. Burton, and Alan G. Bell, 1974. "SOPHIE: A Step Toward Creating a Reactive Learning Environment." International Journal of Man-Machine Studies, 7: 675-696.

Brown, John Seely, Richard R. Burton, and Johan de Kleer, 1982. "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III." In Intelligent Tutoring Systems, ed. D. Sleeman and J.S. Brown, pp. 13-24. New York: Academic Press.

Burton, Richard R., and John Seely Brown, 1982. "An Investigation of Computer Coaching for Informal Learning Activities." In Intelligent Tutoring Systems, ed. D. Sleeman and J.S. Brown, pp. 79-98. New York: Academic Press.

Clancey, William J., 1982. "Tutoring Rules for Guiding a Case Method Dialogue." In Intelligent Tutoring Systems, ed. D. Sleeman and J.S. Brown, pp. 201-225. New York: Academic Press.

Eisenstadt, Marc, 1982. Personal conversation at The British Open University, November 1, 1982.

Hardy, Steve, 1982. Personal conversation at Sussex University, October 1, 1982.

Hardy, Steve, and Aaron Sloman, 1982. "A Multi-Purpose Development Environment." Mimeographed paper available from the authors at Cognitive Studies Programme, School of Social Sciences, University of Sussex, Brighton, England.

Hasemer, Tony, 1982. Personal conversation at The British Open University, October 28, 1982.

Levine, Randy, 1982. Authoring Tools for Computer-Based Instruction: Review and Recommendation." Internal report, Digital Educational Services.

O'Shea, Tim, 1979. "Self-Improving Teaching Systems." Ph.D. Dissertation, University of Leeds. Basel, Boston, Stuttgart: Birkhauser.

O'Shea, Tim, 1982. Personal conversation at The British Open University, November 3, 1982.

Shrager, Jeff, and Tim Finin, 1982. "An Expert System that Volunteers Advice." Proceedings of the 1982 National Conference on Artificial Intelligence (sponsored by the American Association for Artificial Intelligence), Pittsburgh, Pennsylvania, August 18-20, 1982, pp. 339-340. Los Altos, CA: William Kaufmann, Inc.

Stevens, Albert, 1982. "The STEAMER Project." Presentation and demonstration at the Conference on Computer-Based Education Research, University of Deleware, June 3-4, 1982.

Stevens, Albert, Allan Collins, and Sarah E. Goldin, 1982. "Misconceptions in Students' Understanding." In Intelligent Tutoring Systems, ed. D. Sleeman and J.S. Brown, pp. 13-24. New York: Academic Press.

Sussman, Gerald, 1974. "A Computational Model of Skill Acquisition." Ph.D. Dissertation, Massachusetts Institute of Technology.

Wilensky, Robert, 1982. "Talking to UNIX in English: An Overview of UC (UNIX Consultant)." Proceedings of the 1982 National Conference on Artificial Intelligence (sponsored by the American Association for Artificial Intelligence), Pittsburgh, Pennsylvania, August 18-20, 1982, pp. 103-106. Los Altos, CA: William Kaufmann, Inc.

Ulrich, John, 1982. Personal conversation at DEC, October 18, 1982.

Zimmerman, Michael J., and Jesse M. Heines, 1981. "Implications of Window Management for Computer-Based Instruction." Educational Services Technical Report No. 10.