CHAPTER 2

APPLYING THE PARADIGMS IN THE CREATION OF CAI

In the last chapter, we saw that before computers can reach optimal performance in education, more research will have to focus on variables within CAI, and subject-matter instructors will have to become involved in CAI implementation (having first appropriately revised their conceptions of educational paradigms). Then, based on whatever paradigms are found to be appropriate in guiding the use of computers in education, on what is learned through research about various aspects of CAI, and on what we already know and are finding out about how learning takes place in general, effective software will have to be designed.

This chapter will pursue the principles already set forth in showing how they were applied in designing the interactive programming for the present project. In so doing, a process of creation for the non-programming subject-matter instructor contemplating authoring courseware will be defined. This process has the following four stages: (1) deciding on mode of delivery, (2) considering options for possible instructional paradigms appropriate to CAI, (3) working from a definite theory of learning, and having clear terminal goals in mind, and (4) considering the advice and experience of others in settling on the particulars of programming. In this chapter, these four

stages will be examined in greater detail in light of lesson creation for this thesis project.

2.1 Deciding on Mode of Delivery

2.1.1 The Effect of Microcomputers on CAI Development

There are two considerations here, one concerning hardware and the other software. With regard to the former, educators have benefited immensely from the introduction of easily obtainable microcomputers by Steve Wosniak and Steven Jobs in the late 1970's. Durkee (1982:170) credits this development with having prompted "a second wave of research in CAI." Rowe (1983:190) concurs, noting that the "great tide of CAI enthusiasm, which crested in the early seventies" is experiencing a resurgence as "young educators" are attracted to microcomputers. Burnett and Miller (1981-2:208) say that because of the shift away from dependence on large mainframes to stand alone systems, "it is now possible for virtually any interested institution, or individual, to begin developing and using computer-based instructional materials . . . "

This contention was evidenced at the 1983 TESOL Convention in Toronto, where there was demonstrated innovative courseware which had been programmed on Sinclair microcomputers available in the United States for under $100. As further evidence of this fact, consider the present thesis project, which was conceived and developed on a home

microcomputer completely free, except for the experimental portion itself, of any institutional backing.

2.1.2 The Pros and Cons of Authoring Systems

Having decided on hardware, the courseware author must decide how he plans to code the lessons into the machine. The primary decision is whether to use a standard programming language or an authoring system. Use of either is a trade-off. Programming languages will allow the courseware author to relate more intimately with the computer than will authoring systems, which have been designed especially for ease of use by the non-programming lesson author. Increased complexity of the programming language provides greater versatility and therefore more options for format, types of interaction, and the like. However, a programming language might be much more difficult to learn and use, particularly for a non-programming subject-matter instructor.

The case has been made (Merrill, 1982) that the limitations of CAI authoring languages are not compensated for by ease of learning when compared to programming languages such as Pascal or Basic. Authoring languages provide templates which allow courseware authors to concentrate on lesson content more than on programming, but a template "forces the author to turn out courseware which conforms to the template. Such templates can have the effect of enhancing the quality of courseware produced by the novice,

while restricting the quality of the courseware produced by a creative author." (p. 77)

Merrill claims that the disadvantages of authoring languages lie in the reasons that non-programming instructors use them in the first place. Instructors use authoring languages because they provide (1) a reduction of the domain of possible commands, (2) commands and strategies "based on an implied instructional strategy," and (3) commands or routines which perform higher level tasks, relieving the author of both the necessity and capability of learning how to create such functions himself. (p. 76)

As to the first point, Merrill argues that when the domain of commands is simplified, so are possible outcomes. As to the second, although Merill allows that in PILOT, authors are not forced to use tutorial strategy, and that more creative programming is possible, he thinks that "the very nature of principal PILOT commands strongly encourages novice programmers to use a mediocre strategy," thus compromising the quality of their output (p. 76). As to the third, Merrill argues that programming languages are not more difficult to learn than are authoring languages, and that prospective authors should thus dispense with the bother of learning the latter and delve straight into the former, thereby allowing themselves more control over the subroutines they invoke.

This last point is particularly contentious. Showing in a chart how commands in PILOT often have BASIC and PASCAL counterparts, and noting that one who has learned the former will not have access to the higher capabilities inherent in one of the latter languages, Merrill (p. 76) argues that "if an author begins by learning PILOT and then desires greater power, he must then scrap PILOT and begin learning a new language. Why not begin with a powerful language in the first place?" On the other hand, he also points out that a language like PILOT is almost always learned in the context of CAI, whereas programming languages must usually be learned "in their total complexity . . . The PASCAL language is also embedded in a powerful but complex operating system," which the prospective CAI author must also learn to use. Thus, although any of these languages should be equally easy to learn in theory, in practice, it is easier for prospective CAI authors to learn authoring systems.

Merrill's critique of authoring systems must be taken seriously by those considering using one for CAI authoring. However, the limitations which he notes may be overcome by imaginative authors, especially ones who are aware of these limitations and who deliberately seek out ways to circumvent them. Smith (1982) has written a version of PILOT which allows any legal BASIC command to be integrated into a PILOT program, thus effecting higher level capabilities for PILOT. Also, it is possible to find books explaining BASIC

programming purely in the context of CAI (for example, Huntington, 1979). Meanwhile, Merrill's contention that CAI authoring languages prevent lesson authors from exploring the medium is turned against BASIC in Huntington's (1980:77) assessment of that language as being limited by the ad nauseum aspects of logical sequencing. Huntington finds BASIC, in its unextended version, to be "grossly inadequate for significant CAI development."

Time for learning a programming language was a factor in the present project. Since at the beginning of the project, the author neither knew PILOT nor had a good working knowledge of BASIC, it was felt that for the amount of time spent studying a programming code, one could with PILOT be well assured of having, in a short time, the capability of mastering the language for the purposes of writing the needed CAI lessons. Had the author spent the same amount of time learning and programming in BASIC, the courseware produced could not possibly have been as sophisticated as that eventually produced with PILOT. Therefore the decision to use PILOT as the most expedient means of programming the CAI lessons conceived for this project appeared to be the right one.

Of the two lessons created for this thesis project, the one designated PDL allowed students to use game paddles to select from among options in the lesson. This lesson was designed to avoid the pitfalls mentioned by Merrill.

However, as it was found that PILOT did indeed allow "page turner" programming, the second lesson, designated REG, was deliberately created following a linearly structured drill and practice paradigm. The latter lesson was to be compared with the PDL lesson in the thesis experiment.

2.2 Considering Options for Possible Instructional Paradigms Appropriate to CAI

A software developer must avoid educational paradigms which are unsuitable to the nature of computing and select from among those (mentioned in the previous chapter) which are most effective with CAI. This was done for both CAI lessons created for this project, although not to the same extent.

2.2.1 Adapting Elements of Clarifying Educational Environments

The lessons created for this project were designed so as to fulfill the four criteria for Moore and Anderson's clarifying educational environments. These four criteria are that the environment must (1) allow the learner to vary perspective, and be (2) autotelic, (3) productive, and (4) personalized. We will now examine how the lessons created for the project met these criteria.

(1) Both lessons allowed the learners to vary perspective on their learning. The PDL lesson fulfilled this criterion to a greater degree than did the REG lesson, especially in that it allowed learners enhanced agent

capabilities in selecting sentences to be worked, and in that learners could move from the inductive to deductive portions of their lesson according to their individual whims and learning styles. In problem mode, both lessons presented equivalent linguistic data and then asked the students to act on that data, thus allowing shifts from patient to agent; however, this was a mechanical shift. Insofar as the computer's reactions to input were not always predictable to the students, both lessons allowed reciprocal perspective as well.

(2) The CAI environment was designed to be autotelic; that is, it was meant to be an activity free of consequences and enjoyable for its own sake. This might have held to some extent for those subjects who felt that they were benefiting by using the computer, but in actual practice, it is doubtful that the students in the thesis experiment enjoyed a purely autotelic experience. They were told to work the lessons, and then told how long they could take and when to stop. Also, they were tested directly before and afterwards, giving them the impression that they were being judged on their performance. Although this rigidity was in conflict with the intent of the lesson designer, it was necessary for the integrity of the thesis experiment. However, the lessons were meant as a resource to be used voluntarily and non-threateningly by the students; hence they were by design autotelic.

(3) The CAI environment was productive; that is, what was learned in that environment was applicable to analogous situations outside the learning environment.

(4) The CAI environment was personalized and reflexive. A personalized environment is responsive to the learner's activities. In order for the environment to be responsive, according to Moore and Anderson (p. 65), it has to (a) permit the learner to explore the environment freely, (b) give the learner immediate feedback, (c) be self-pacing, (d) permit the learner "to make full use of his capacity for discovering relations of various kinds", and (e) be "so structured that the learner is likely to make a series of interconnected discoveries about the physical, cultural, or social world. Both experimental lessons easily incorporated all of these criteria, except that only the PDL lesson permitted free exploration of the learning environment. Also, the PDL lesson possibly allowed fuller use of the learners' "capacities for discovering relations" and hence perhaps increased the chance of the learners' making "interconnected discoveries". Whether this was true was in fact the focus of the thesis experiment.

Finally, the environment was to some extent reflexive, i.e. facilitated the learner's seeing himself in a social perspective. The setting of both lessons was such that the learner had to identify with a rake named Max, whose foibles were intended to cause the learner to consider himself in

the (anti) social situations in which Max found himself. Thus the environment was reflexive to the degree the learners placed themselves in its setting. Furthermore, learning languages is inherently a reflexive activity, since proficiency in that language facilitates interaction with the environment in which that language is used. Hence, it is safe to claim that each CALL lesson, and particularly the PDL lesson, created a clarifying educational environment, since each met all the above criteria.

2.2.2 Adapting Elements of Games

The CALL lessons created for the present project, in particular the PDL lesson, had some characteristics of games, although neither lesson was a game per se. They had to some degree the elements of games enumerated by Stevick; that is they had rules and goals, and the "players" (particularly the PDL players) all had control over options and something in common.

The goal of the "game" was to discover the grammar governing the use of either gerund or infinitive complement with the matrix verbs 'stop', 'remember', 'regret', and 'forget'. The elements common to all "players" were that they all wanted (presumably) to discover the above-mentioned rules, and that they could all relate to the context of the lesson, wherein Max makes a fool of himself at a party and

spends the next day trying to piece together exactly what he did the previous evening, before the lights went out.

The "game" had rules insofar as a computer program, by definition, works according to "rules" or parameters set by the programmer. Figuring out these rules is always one of the objects of the "game" that computer users play with computers, and hence one of the reasons that use of computers in learning can be valuable in and of itself (especially if these rules are not transparent).

2.2.3 Adapting Elements of the Berry Metaphor

The most important element in the PDL lesson created for this thesis project was that students using the experimental lesson were given control over their choice of options in elucidating the rules governing both the computer program and governing the grammar of English itself. They made these choices by picking sentences off a chart in the manner characterized by the berry-bush metaphor suggested by Scollon and Scollon (1982).

This last point was the one about which the experimental question revolved: to what degree is choice and control crucial in effective CAI programming? Herriott (1982) counsels against letting the computer take too much control, as this can have a lulling effect that might work to the detriment of other benefits inherent in the medium. Instead, he advises that students be given more control over their

direction in a course, even to the point of deciding when to leave it. This capability was in fact one of the options built into the experimental program designed for this project. Marty (1981) stresses that computers should be programmed in such a way that students have ultimate control over how they adapt them to their individual learning styles. As he puts it (p. 33), "The student should view the computer system as an ally ready to help him learn as efficiently as he can, not as a slave driver." Of the programs designed for this project, the PDL one was an "ally", and the REG lesson a "slave driver".

Thus we see that how one views the nature of computing influences the creation of CAI materials. Now we shall turn to considerations revolving about how one views both the nature of learning, and the nature of the material to be learned.

2.3 Working from a Definite Theory of Learning, and Having Clear Goals in Mind

One frequently mentioned criticism of CAI is that courseware is often produced which is not based on a viable model of learning. Only by working from such a model can one successfully attain whatever learning objectives have been set. This section shall discuss the learning strategies employed in the lessons developed for this project in light of such considerations; for example, Wager's (1981-2) adaptation of Gagne's learning algorithm to CAI. I shall

then briefly mention other important programming considerations. In the chapter following this one, I shall discuss how the lessons were based on a viable model of the specific grammar point taught in the lessons.

2.3.1 The Need for Theoretically-Based Software

The fact that existing educational software often lacks a theoretical base is frequently mentioned in the literature. Thé (1982:52), for example, says that "software must be designed with clear learning objectives in mind," a point which is echoed by Bork (1981:4). Steffin (1982a:25) suggests that without such objectives, CAI might as well not be used: "The ease with which we can design effective instructional strategies that involve the use of the computer and other media is directly related to the degree to which we know what we want the learner to be able to do after instruction . . . . Generally, in situations where learner objectives are clear and agreed upon the computer is in an excellent position to serve as an instructional tool. But to the degree that objectives are vague or ambiguous, or where there is wide discrepancy in the views of various teachers about intended outcomes, lecture/recitation, classroom discussion, written essays, or the viewing of well prepared film or video demonstrations are likely to be more useful tools than the computer."

Burnett and Miller (1981-2) developed some innovative lessons giving reading instructors insights into the reading process. In the authors' view, "the main feature of this project lies in the adaptation of a viable model of the reading process to a computer assisted instructional format." (p. 218) For example, their model of learning dictated a thoroughly inductive approach. Thus the authors feel that the resulting exercises were "unique in that no factual information is presented to the student; he discovers the answers as more and more written language is presented." In addition, metalanguage was totally dispensed with in the lessons.

Feeling that an inductive approach best taught the principles involved, Burnett and Miller created programs that required prospective teachers to make predictions and to draw conclusions about the reading process based solely on the data presented in the programs. Accordingly, the programs designed for the present project used a similar strategy, except that the inductive phase was followed by a deductively structured rule and recapitulation section. That is to say that students initially had to rely solely on a presentation of linguistic items and make sense of these through appropriate setting and paraphrase.

2.3.2 Goal Directed Backward Chaining

Burnett and Miller (1981-2) used what is sometimes called goal directed backward chaining in creating their programs in three distinct stages. First, exercises were conceptualized without consideration for programming limitations. Next, the exercises were modified for the computer (the major constraint at this stage being the size of the text window). Finally, the program code was altered to improve machine speed. Working in this way, they avoided letting the medium dictate mode of delivery, a strategy which can easily lead to shallow gimmickry.

The present CALL lessons followed exactly this line of development. The author first "imagined" the exercises, conceptualizing how the subject matter could best be taught within the desired theoretical framework, and without considering constraints in programming. The latter were dealt with only later. Thus the lessons produced had roots first in the paradigms for education appropriate to the technological age espoused in the first chapter, and second in the author's assumptions about learning discussed here. They were not built, for example, around a desire to use the game paddles; rather the game paddles were employed as a vehicle for carrying out a preconceived instructional strategy.

2.3.3 Applying a Suitable Algorithm for Learning

Another trap to avoid in working in the medium of computers is total reliance on the "relentless, linear logic" which we have seen distracts novice courseware authors using authoring systems such as PILOT. Wager (1981-2) is concerned that users of such systems in particular show "lack of concern with the application of principles for the design of the instructional materials derived from a consistent and valid theory base." (p. 269) He points out that following the suggestions in manuals and guides accompanying these authoring programs is unlikely to be of much help in producing viable materials. "Without a sound theoretical position it is difficult to come up with a consistent set of rules for CAI lesson design." Such a set of rules would "consider the types of learning, learner characteristics, and the situation in which the CAI will be used." (p. 268) In developing the lesson, the author would first classify the objectives of the lesson, then properly sequence these objectives, and finally produce program elements based on an information processing model such as one developed by Gagné.

There are several algorithms adaptable to CAI; Wager mentions tutorial, drill and practice, simulation, and games. The first two can be distinguished by the fact that the former (the tutorial) is used in acquiring a skill, while the latter (drill and practice) is used in maintaining and improving a skill (Vinsonhaler and Bass, 1972:30). This is

not however a universally held distinction; Kulik et al. (1980:529), for example, considered any program that "presented instruction directly to students" a tutorial, and did not list drill and practice as one of their "four major types of applications of the computer to instruction".

Wager feels that the tutorial, as opposed to drill and practice, simulation, or games, is the most efficient and widely used presentation for CAI. Also, he feels that the tutorial algorithm in particular seems amenable to adaptation to a learning algorithm proposed by Gagné, which would result in a CAI lesson having the following components:

(1) First, a motivating display would provide a set for learning. (2) Next, objectives of the lesson would be made clear. (3) The learner would then be informed of what skills were necessary to do the lesson, or would be given an entry quiz. (4) A stimulus would be presented in the form of new information, a definition, a rule, or a representative problem. (5) Some form of learning guidance, (for example, framing the new information in a meaningful context) would then be presented. (6) Performance would be elicited. (7) One of four types of feedback would be given. (For more on feedback in CAI, see Swenson & Anderson, 1982.) (8) Performance would then be assessed in a way that the student would have some idea of his degree of mastery. (9) Finally, steps would be taken to enhance retention and transfer.

With the exception of numbers (3) and (9) above, these steps were all present in the lessons developed for this project. (No entry quiz was deemed necessary, and no steps were taken to enhance retention since no CAI work was envisioned beyond the experimental work.) The lessons began with a set for learning (a title page), and this was followed by a statement of objectives. After some further "setting", a stimulus was presented by way of a chart from which students would select sentence components. On selecting each sentence, learning guidance was provided by way of paraphrase. For each item, the students were then asked a question, their responses elicited, and feedback given. At the end of the problem mode, performance was assessed in a five-problem quiz.

In being aware of and applying current ideas in learning, specialists in education can be expected to approach CAI lesson design differently from (and possibly without the assistance of) professional programmers. In their approach, education specialists would likely opt to use a CAI authoring system in lieu of a programming language, remaining aware of possible paradigms for education applicable to the programming task, possibly circumventing the paradigm "suggested" by the creators of the authoring system. Furthermore, specialists in education could be expected to create successful courseware by utilizing an algorithm for learning based on a viable model of the learning process. Thus we

have compelling reasons for enlisting educators in the creation of CAI programming.

Educators have a further advantage over programmers in that they alone know the subject matter to be taught. Focusing a thorough knowledge of this specific subject matter into goals for terminal student performance is still another component in the successful creation of viable CAI materials. In the following chapter, we shall discuss in greater detail the importance of a knowledge of English grammar to the creation of the present pedagogically-based CALL lessons. At present, we continue our discussion of what a lesson author needs to consider in order to produce courseware that will attain the goals the author has in mind.

2.4 Considering the Advice and Experience of Others in Settling on the Particulars of Programming

A final consideration in CAI lesson creation is to acknowledge guidelines for programming based on the experience of others where appropriate. For example, the following are suggestions noted by Thé (1982), each related in turn to the lessons created for this project:

(1) The software should be friendly to the extent that getting from one place in the program to another is no more difficult than thumbing through a book. This capability was to a degree programmed into the PDL lesson in that

students were able to at least choose and control what section of the lesson they wanted to work in. However, it was deliberately omitted from the REG lesson, thus enabling the researcher to test the effectiveness of this variable.

(2) Use of the program should be self-explanatory. Both of the experimental programs were in fact designed to be self-explanatory. In practice, students usually needed no assistance with the REG program, and the PDL program contained an explanation. But as this explanation was difficult for non-native speakers to understand, use of the lesson was explained personally to each student. Incidentally, much effort was invested in trying to make the PDL lesson truly self-explanatory, but each successive attempt was somehow flawed; in the end, a few words from the researcher proved to be the most expedient and effective means of explanation.

(3) Software should present information clearly and in consistent format and include all the cues to comprehension (i.e. upper and lower case) found in printed matter. Pains were taken in the experimental lessons to see that format was held constant. Also, PILOT has the advantage over BASIC (on the standardly equipped Apple II+) that it allows upper and lower case.

(4) Educational software should use color, graphics, and sound when appropriate, and allow automatic remedial branching. In short, "it should exploit the unique

capabilities of the computer . . . . There is no reason to buy software that isn't superior to a book." (1982:114) Except for the fact that color was not used, the experimental lessons incorporated all of these elements unique to computers. Use of color is easily invoked with PILOT, but color monitors were not available at any of the experimental locations, and this is why these lessons made no use of PILOT's color capabilities. (For more on graphics in CAI, see Alessandrini, 1982).

(5) Appropriate reinforcement should be provided, and should help the student "not only catch mistakes but analyze them for patterns, which helps the [student] understand how he made the mistake, and not just that he made it." (Thé, 1982:110) This feedback should be kept simple, according to Marty (1981), who also notes that errors students correct on their own are less likely to recur than are other errors. Feedback in the present lessons was so designed that students would be guided into correct responses on second attempts at answering questions correctly.

(6) The program should accept a wide range of correct answers so as to avoid telling the student he is wrong when he isn't. This is one of the most difficult tenets to follow in programming CAI. The present lessons circumvented this problem by restricting the range of possible answers. If the student deviated from this range, he was given

feedback which guided him back into range. Thus, there was never an instance in the experimental programs where students received inappropriate responses to their input to the computer.

(7) Finally, The echoes Moore and Anderson in saying that the more flexibility allowed in the ways the lesson material can be presented and sequenced, or in the levels of difficulty available, the better. These experimental programs did not take into consideration levels of difficulty, but the PDL lesson allowed variation in presentation and sequencing. In these lessons, students were allowed to choose between inductive and deductive presentations, and to control the sequence of these presentations and of items in the problem mode.

From the preceding two chapters, it can be seen that the lessons created for this thesis project (1) followed paradigms for education consistent with the nature of computing, (2) adhered to instructional strategies having some theoretical basis, and (3) utilized the experience of previous programmers in incorporating certain principles of successful CAI programming. We know also that viable CAI must be based on a thorough knowledge of the subject matter to be taught; thus it shall be shown in the next chapter how the lessons created for this project utilized pedagogically sound lesson materials.