ED 111 398 IR 002 454

AUTHOR	Schuyler, James A.
TITLE	Hypertext + Tutor = Hypertutor.
PUB DATE	Aug 75
NOTE	17p.; Paper presented at the Association for the Development of Computer-Based Instructional Systems Summer Meeting (Portland, Maine, August 4, 5, 1975)
EDRS PRICE	MF-$0.76 HC-$1.58 Plus Postage
DESCRIPTORS	*Computer Assisted Instruction; Computer Programs; *Cost Effectiveness; *Programing Languages; *Systems Concepts; *Teaching Methods
IDENTIFIERS	Hypertext; HYPERTUTOR; Northwestern University; PLATO IV; Program Sharing; TUTOR IV

ABSTRACT

The HYPERTUTOR incorporates the ideas of a "hypertext" and the TUTOR-IV programing language used on the PLATO-IV system. The HYPERTUTOR is a part of Northwestern University's MULTI-TUTOR system and runs on a non-PLATO, non-dedicated CDC 6400 computer. It allows the transfer of courseware from PLATO to non-PLATO systems. It has successfully been transferred to other CDC 6000-series and Cyber-70 computers. This paper outlines the rationale for the creation of such a system, and gives the background of MULTI-TUTOR, its systems structure, and its compatability problems with PLATO's Tutor. Current MULTI-TUTOR sites are listed along with an outline of the clearinghouses for lessons now being established at Northwestern. An analysis of current cost factors of the MULTI-TUTOR system is included. (Author)

HYPERTEXT + TUTOR = HYPERTUTOR

James A Schuyler; Northwestern University

Abstract: The HYPERTUTOR incorporates the ideas of a "Hypertext" (a word coined by Theodor H. Nelson) and the TUTOR-IV programming language used on the PLATO-IV system. The HYPERTUTOR is a part of Northwestern University's MULTI-TUTOR system and runs on a non-PLATO, non-dedicated CDC 6400 computer. It has successfully been transferred to other CDC 6000-series and CYBER-70 computers. This paper outlines the rationale for creation of such a system, gives the background of MULTI-TUTOR, system structure, compatibility problems with Plato's Tutor, lists current MULTI-TUTOR sites, and outlines the clearinghouse for lessons now being established at Northwestern. It concludes with an analysis of current operating costs on the MULTI-TUTOR system.

Background

The MULTI-TUTOR, including the HYPER-TUTOR, really exists for a single reason -- to make the transfer of Tutor-language courseware from Plato to non-Plato systems possible (and vice versa). The reasons for this desire to transfer courseware, and the many problems complicating transfer, are the subject of this paper.*

There is really a single basic reason for transfer of C.A.I. courseware -- it takes so long to produce C.A.I. instructional units that the cost must be shared among many institutions and students to make it worthwhile. The sharing can be done by creating gigantic C.A.I. systems such as Plato[1] where 1000 students may be connected to the computer at once, or it can be done by physically transporting courseware to many different locations to be run (the TICCIT[2] approach).

The MULTI-TUTOR (previously called HyperTutor) was created in response to three specific needs:

a) We needed to share the cost of lesson development, as outlined above. The potential to transfer our lessons to Plato systems and receive royalties, as well as the potential to import Plato lessons is still the overwhelming factor.

b) People who cannot afford the cost of extra equipment, terminals and software to run an actual Plato system still would like to use the Tutor language, so that their programs will be fairly compatible with Plato systems in the future. Many installations have hundreds of CRT terminals, and cannot afford to trade them on the comparatively costly Plato terminals as this would reduce the number of terminals available to students. In addition, even though the cost of the

** The term "courseware" includes computer programs ("software") and associated documentation for students.*

Plato terminal is expected to decrease, it shows no signs of doing so in the immediate future. MULTI-TUTOR provides something to do while we're waiting.

c) Many programs written in the Tutor language do not require the full capabilities of the Plato system or its terminals. In the future, as well as the present, these programs should be run on inexpensive CRT terminals, rather than utilizing the costly Plato terminals.

There are some reasons for caution in discussion of the transport of Tutor programs:

*a) Some programs require the communications provided by Plato in its 1000-terminal environment. Such programs should not be transported unless the target system can support these communications.**

b) Some programs really make effective use of the plasma-panel graphics as part of the teaching process (as opposed to the "bells-and-whistles" use made of graphics in many Plato programs). These may be run on MULTI-TUTOR, but still require a Plato terminal (which MULTI-TUTOR supports).

c) Some programs require the instant response Plato can give. Brackett[3] discusses this point, and claims that if a Plato program does not "feel" the same on another system, it will lose its educational effectiveness. MULTI-TUTOR can run in this way, but the operating system must give CAI top priority.

The MULTI-TUTOR/HyperTutor approach

MULTI-TUTOR provides most of the commands of the Tutor language, plus an "editing" and lesson-creation tool called the HyperTutor, to users of Control Data 6000 and Cyber-70 computers, running under standard Kronos or Scope operating systems.

Two assembly-language programs serve as the base of the system:

a) A syntax scanner and translator (READIN) which converts Tutor programs into Micro-Tutor, which can be interpreted during interactions with students. (This is roughly the function of Plato's condensor program.)

b) An interpreter for the Micro-Tutor language.

On top of this base stand Tutor lessons written by courseware authors, and a number of "system programs", all written in the Tutor language:

a) The Hyper-Editor (Hyper-level),

b) A more conventional text-editor,

** MULTI-TUTOR does, in fact, support many communications tools. But, in practice, local MULTI-TUTOR users may find that the operating system or physical facilities may make them difficult to use.*

c) A sign-on program which determines the type of terminal a student is using and sets operational parameters for MULTI-TUTOR,

d) All communications programs, and

e) An execution-error reporter and processor which gives feedback to courseware authors about student problems.

The MULTI-TUTOR was designed to provide a lesson-organizing structure much like Theodor H. Nelson's proposed Hypertext[4]. This structure would make it possible for students or teachers to browse through existing "public" lessons on each computer system. Because the Hypertext concept was combined with the Tutor language, the system was originally called the HyperTutor. It now supports the operation of Hazeltine, A.D.D.S, Teletype-replacement terminals, Plato and Tektronix 4023 (CRT) terminals. It does not require dedication of any part of the computer, running as a regular time-sharing job. Lesson materials can be swapped directly from disk or from high-speed extended core storage (ECS) as on Plato. Because of the flexibility of this system, it is now called the MULTI-TUTOR.

The MULTI-TUTOR is composed of three operational levels:

a) The HyperTutor, an interactive "editor" where the authors of courseware manipulate paragraphs of text, answers, actions to be taken contingent upon students' responses, objectives and flow-charts for lessons. The terminal used for interaction at this level is normally a Plato terminal with touch-panel, because HyperTutor depends heavily on graphics and quick interaction. The HyperTutor produces Tutor commands to be used at the next lower level.

b) Tutor, the high-level language, may be used directly by programmers, or may simply be used to process code produced at the Hyper level. Some authors prefer to create courseware initially in the HyperTutor, then slip down the ladder to edit that material further in the Tutor level. Experiments done here at Northwestern have shown a reduction in lesson preparation time of approximately 25% when this approach is used.

c) The Micro-Tutor. Here the actual functions required to service the student are represented in a Tutor-like (but more detailed) form. This level of the system is the one we anticipate using to support other "hard core" C.A.I. languages*in the future. All that is required to add a new language to the system is the creation of a syntax scanner and translator. Nobody programs at the micro-level; it is simply the representation chosen for the concepts critical to all C.A.I. languages, and only serves to drive the interpreter.

* "Hard core" C.A.I. is the term used to reference systems in which a special-purpose language or equipment are used for C.A.I.

History

The design and programming of MULTI-TUTOR began in late 1972, and although the system is perfectly operational, it is constantly updated to provide compatibility with the ever-changing Tutor language. During 1974, MULTI-TUTOR (under the name HyperTutor) was transferred to other universities for preliminary tests. By July, 1975, it had been shipped to seventeen sites, and was known to be operational at seven of these.

One major pitfall in the growth of C.A.I. systems has been the distribution of systems without courseware to accompany them. In September, 1975, when MULTI-TUTOR is officially released, it will be accompanied by a courseware exchange service. In the initial exchange will be lessons developed and tested at Northwestern, in C.A.I., Pediatrics and Linguistics. From this small base of about fifty lessons, the exchange will be expanded to include lessons created on Plato as well as other MULTI-TUTOR systems.

Current MULTI-TUTOR development is concentrated on the development of the HyperTutor program described in a later section of this paper.

Compatibility

MULTI-TUTOR is not completely compatible with Tutor in the sense that it accepts the full Tutor language. There are several levels of compatibility which MULTI-TUTOR supports:

a) Any MULTI-TUTOR lesson should run, unchanged except for character set translations and a few simple replacements done by a Snobol program, on Plato. If it does not, the condition is not intended.

b) Plato lessons incorporating no line-drawing or author-designed characters, will run on MULTI-TUTOR's character-oriented CRT terminals. In addition, many lessons which use graphics do not use them in a substantial, educational, way. These also can be transferred at this level. These lessons may also include common-storage and datasets.

c) Lessons incorporating graphics, including character sets, can be run on Plato terminals connected to the MULTI-TUTOR.

d) On the Kronos operating system only, single keypresses can be processed, allowing a much more flexible interaction with the student.. The speed of this processing depends to a large degree upon local operating system variables.

e) MULTI-TUTOR will not support the echoing of special character sets to the student typist, and therefore is contraindicated for language lessons requiring strange character sets. Special local systems with front-end computers may be able to support such echoing, but Plato is a far better choice.

Courseware transfer, in both directions, depends upon simple character-set related conversions. These are now performed by a Snobol program at Northwestern.

Potential for Courseware Exchange

MULTI-TUTOR is now operating at at least seven sites:
Northwestern University, Evanston, IL
The University of Texas, Austin, TX
The University of Colorado, Boulder, CO
Mass. State College Computer Network, Boston, MA
McMaster University, Hamilton, Ontario, Canada
COWO - University of Amsterdam, Amsterdam, Netherlands
Oregon State University, Corvallis, OR

A number of other institutions with substantial interest in C.A.I. currently have MULTI-TUTOR tapes and are in some stage of implementation. Implementation at the sites known to be running has taken at most a few days, with most making successful implementations in less than a day. Several sites other than Northwestern are known to be developing courseware -- the extent of these committments is not known. We, therefore, do not expect a great influx of courseware from MULTI-TUTOR sites for another year or two.

We are currently seeking courseware from Plato authors for the exchange. Implementation of this courseware will procede cautiously until we can ensure proper compatibility with Plato's Tutor, and then we will release the courseware for other sites with Plato terminals.

The MULTI-TUTOR courseware clearinghouse will perform several functions, and is patterned after the CONDUIT[5] model:

a) Courseware to be transferred between Plato and MULTI-TUTOR will be unblocked and some character conversions performed so that it can be easily read on MULTI-TUTOR systems. MULTI-TUTOR courseware for Plato use will also be blocked and converted.

b) Technical validation will be performed at Northwestern, using terminals of a type suitable for the courseware in question. All programs must run without technical flaws to pass this stage.

c) The clearinghouse will then suggest appropriate people or organizations for review of the content of the lesson. Certification in the CONDUIT organization is an appropriate model -- discipline committees in recognized academic areas review the courseware's content and strategy.

d) Courseware which has been passed will be catalogued and released to MULTI-TUTOR users.

Experience has shown that it may take a year or more for courseware to reach the catalog stage.

Royalties or single-sale payments will be made to the authors or owners of courseware used on MULTI-TUTOR systems. MULTI-TUTOR reports all lesson usage scrupulously.

Documentation

C.A.I. courseware includes not only the computer programs, but associated documentation aimed at teacher, student or programmer. Using the preliminary documentation guidelines prepared by CONDUIT,[6] and input from C.A.I. experts participating in a C.A.I. task-force for CONDUIT, we have prepared preliminary check-lists for C.A.I. documentation. These are divided into three parts: teacher, student and programmer (or technical).

Teacher documentation:

This documentation is expected to give the prospective adoptor an overview of the material, followed by all details necessary to run (but not necessarily change or fix "bugs" in) it. It is suggested that this documentation include:

Overall information: A statement of the availability of the materials (time required for shipment, cost, procedures, location of the source, copyrights or other restrictions, and limitations for liability of error). A list of other centers using the material would also be useful, and a statement of what local support is included.

An abstract: The abstract includes the descriptive title of the package, authors' names, original source, names and locations of subsequent modifiers, a summary of the substantive content, statement of educational objectives, specification of the computer's unique instructional role, background requirements for instructors and students, level of student, references used in the construction of the program, a statement of supplementary materials needed (text, film, etc), instructional strategy and logic (e.g. drill, tutorial, linear, inquiry, etc.), and citations of published reviews.

Educational documentation: It is desirable to include the theoretical foundations, formulae and techniques of a substantive nature, relation between program and course materials, recommendations and examples for use in a standard curriculum, descriptions of how computer activities have been integrated into the classroom using this package, a statement of expected time allocation for students, faculty and computer equipment, suggested methods for evaluating student performance, reports on actual use of the materials in class, and sources of information on formal evaluation (e.g. published results).

Student documentation:

Documentation for the student is expected to give an overview of the material, plus specific instructions on how to use it.

Abstract: It is recommended that the student be provided with the full abstract as outlined above. In any event, s/he should be provided with: Descriptive title, summary of substantive content, statement of educational objectives, background requirements for students, a statement of supplementary materials needed (text, film, etc).

Educational documentation: It is recommended that the student be provided with at least the theoretical foundations, formulae and techniques of a substantive nature, and relation between program and course materials.

Technical documentation: If the student is expected to make any modifications to the program, s/he should be provided with full technical documentation as described below for the programmer. Otherwise, it should include: a rough explanation of the program's logical organization, descriptions of sample input and output (if not adequately explained interactively, or if the student must prepare input before the session), test data and expected results, a full list of terms and available options and how to exercise them, and an explanation of any program-generated messages and how to correct associated faults.

Local documentation: Instructions on how to use the local computer, examples showing use of the local computer, a setup example of how to access the program under consideration, notations on any significant differences between the local version and the universal model of the program, and information on typical run times and costs. It is also recommended that this information contain names and phone numbers for 1) the teacher, 2) personnel at the computer site who are familiar with the program, and 3) consultants who can help with computer problems.

Programmer documentation:

The programmer is the person who initially gets the package of lessons going on the local computer, and is responsible for local technical verification before the programs are used by students. Thus, the full packet of information for teacher, student and programmer should be available to him. However, specific technical information is also needed as follows:

Technical documentation: An explanation of the program's logical organization, an explanation of the functions of discrete modules within the program, a well-commented program listing (usually on tape), sample input for all options, sample interactions, listings of output generated by sample data, full description of output, suggested test problems, full list of terms and options, an explanation of program-generated messages and the conditions which cause them, a glossary of variable names and their uses, list of counters and their uses, auxilliary storage areas, supporting programs, files and common storage areas. If files or datasets are used, include a description of file organization, indices, security procedures and passwords. A good system flowchart should always be included.

The programmer will often be the person to write the local documentation for the students, and for this reason should be familiar with the operating system and potential student problems.

Cost of MULTI-TUTOR

We have conducted some short tests to determine how much it is costing us to create lessons using the MULTI-TUTOR, and the factors we have included are listed below:

	cost component	time component
C.R.T. terminal ($2,000) lasts 5 years, used 8 hours per day, 5 days per week, 50 weeks per year. (10,000 hours)	$ .20 per stud. hr.	
Repair bills per terminal Est. $300 per year (2,000 hrs/year)	$ .15 per stud. hr.	
Modems: last 5 years, as for term'l. Initial cost $300	$ .03 "	
Phone: dial-up local line Est. $15.00 per month	$ .09 "	
Cost of C.A.I. operating system $1,000 per year to maintain, distributed over 10,000 student-contact-hours at Northwestern)	$ .10 "	
Computer time: actual average at Northwestern	$1.75 "	
Researching, planning, consulting, discussing with others @$10/hour	$200 per 1-hr lesson	20 hours per 1-hour lesson
Coding, editing, entering into the computer @$5.00/hour	$150 "	30 hours per 1-hour lesson
Preliminary testing, review and recoding @$10/hour	$100 "	10 hours per 1-hour lesson
(Costs of updating the lesson from time to time are not included.)	totals	
Assuming that an average 1-hour lesson is used by 500 students, over a single year period	$ .90 per stud. hr.	
Additional possible costs: Supervision of C.A.I. center	$?	
Connect-charge: varies a lot, is $1.00 per hour at Northwestern	$1.00	
TOTAL COST PER STUDENT HOUR	$4.22	60 hours

These costs are typical of those encountered at Northwestern with the MULTI-TUTOR. Certain items will vary widely in the cost equation, primarily the cost of terminals, phone, cost of the operating system, number of student-hours a lesson produces and the connect charge.

For example, a change in connect-time charges can be directly added to the cost equation. Or a purchase of a Plato terminal rather than the inexpensive CRT we use could be calculated to raise the cost to a total of $5.02 per hour.

Conclusion

The MULTI-TUTOR provides a moderately-priced C.A.I. system for people who have some Cyber 70 computer power to spare. It does not require additional equipment, either computer or terminals. It provides most of the facilities of the Tutor language and a great degree of compatibility with Plato's Tutor. It should, therefore, allow relatively easy transfer of lessons to and from Plato systems.

Notes

(1) Numerous reports available from Computer-based Education Research Laboratory, University of Illinois, Urbana, IL 61801
Best reference on the Tutor language is:
Sherwood, Bruce Arne, "The Tutor Language", 1974

(2) Bunderson, C. V. "The Ticcit Project: Design Strategy for Educational Innovation", Institute for Computer Uses in Education, Technical Report # 4, Brigham Young University, Provo, Utah 1973

(3) Brackett, John (Principal Investigator) "A TUTOR Minicomputer System", SofTech, 460 Totten Pond Road, Waltham, Mass. 1975

(4) Nelson, Theodor H., "No More Teachers' Dirty Looks", in Computer Decisions, September, 1970 (pp.16-23)

(5) See numerous papers in "Computing and the Decision Makers", EDUCOM, Interuniversity Communications Council, Princeton, NJ 1974

(6) "Preliminary Documentation Guidelines", working-paper used within CONDUIT (A consortium of regional networks at Oregon State University, North Carolina Educ. Computing Service, Dartmouth College and the Universities of Iowa and Texas (Austin)).

A HYPER-Level System for the Creation of Computer-Aided-Instruction Courseware

Background and Problem Statement:

Plato's Tutor language [1] will be widely known in a few years, and much courseware will be finished, having gone through debugging, validation with classes and certification by peer-review boards such as those created by CONDUIT [2]. Tutor certainly appears destined to become the "Fortran" of Computer-Aided-Instruction.

Tutor is among those languages for C.A.I. used on a system where lessons (or programs) are actually programmed, as opposed to systems where the computer generates questions from a database, or Ticcit-like systems where authors of courseware enter information on coding forms or in an interactive fashion different from the programming format of Plato. Though Tutor programmers may write code which allows the student wide lattitude in lesson choice, and may even write code which mimics the behavior of other C.A.I. systems, it is difficult to do so because of the necessity of learning more and more Tutor commands to do it. Most courseware authors take the straight and narrow path, which may tend to steer them in the direction of "programmed instruction". The approach outlined in this paper would cut authoring time for complex strategies by 25 to 50% for both experiences and inexperienced authors.

Authoring time for C.A.I. lessons:

Experiments carried out at Northwestern have shown that lesson-writing for inexperienced authors can take from 30 to 120 hours of author time for each student-hour of material produced as a final product. Average times for a group of six inexperienced authors are shown below:

Authoring time to produce one student hour

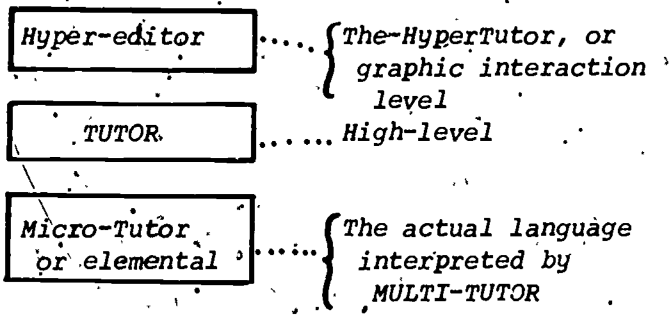
	hours	*percent*
Planning, consulting, looking at other lessons	*15*	*30%*
Coding, typing and editing	*25*	*50%*
Testing, revision	*10*	*20%*
	50 hrs=100%	

Notice that even though the editing system used is quite sophisticated, and allows immediate interactive testing of a lesson, the major amount of time is spent just entering code and getting it to work.

A Hyper-level Language for C.A.I.:

Current C.A.I. authors generally work with high-level languages, such as Plato's Tutor. These allow rapid progra-ming of interactive units when an author is an experienced programmer. Tutor also allows inexperienced authors to program relatively simple interactive strategies, but its full repertoire of commands is beyond the reach of anyone without six months or more of programming experience. Tutor contains over 200 different "commands", but a subset of perhaps 15 is enough for the beginner.

In the MULTI-TUTOR system, the Hyper-editor (or HyperTutor) is now being added above the level of the Tutor (high-level) language:



Hyper-editor ..... *The HyperTutor, or graphic interaction level*

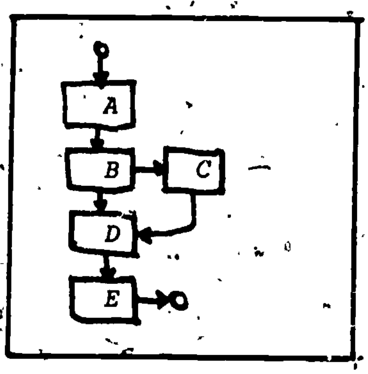
TUTOR ..... *High-level*

Micro-Tutor or elemental ..... *The actual language interpreted by MULTI-TUTOR*

At the Hyper-level, a courseware author interacts with the computer using graphics and text. The Hyper-editor accepts or elicits courseware objectives and structure, and creates a map of the lesson to be written during the interaction. Though the Hyper-editor will not be completed until the end of 1975, we have provided a short scenario describing the intended interaction.

A Scenario

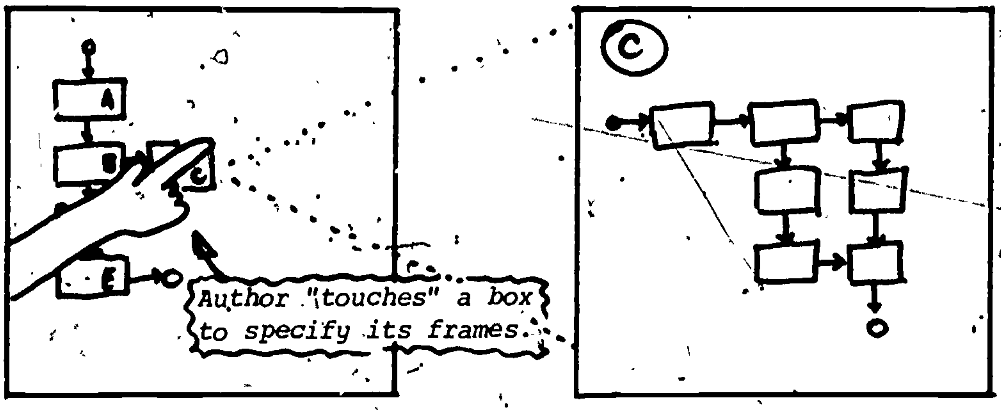
The Hyper-editor, using a Plato terminal equipped with a touch-panel, lets the courseware author specify a flow-chart for the general flow of the new lesson.



This map, along with the objectives specified for each box, is then available to the students who are browsing through MULTI-TUTOR's list of public lessons, letting them "flip the pages" of the lesson before deciding whether to explore further.

2

The lesson-creation process then continues with the author proceeding to more and more detail in the flowchart. S/he "enters" a box on the flowchart by touching it -- HyperTutor then requests individual frames ("Units" in the terminology of Tutor) of the lesson.

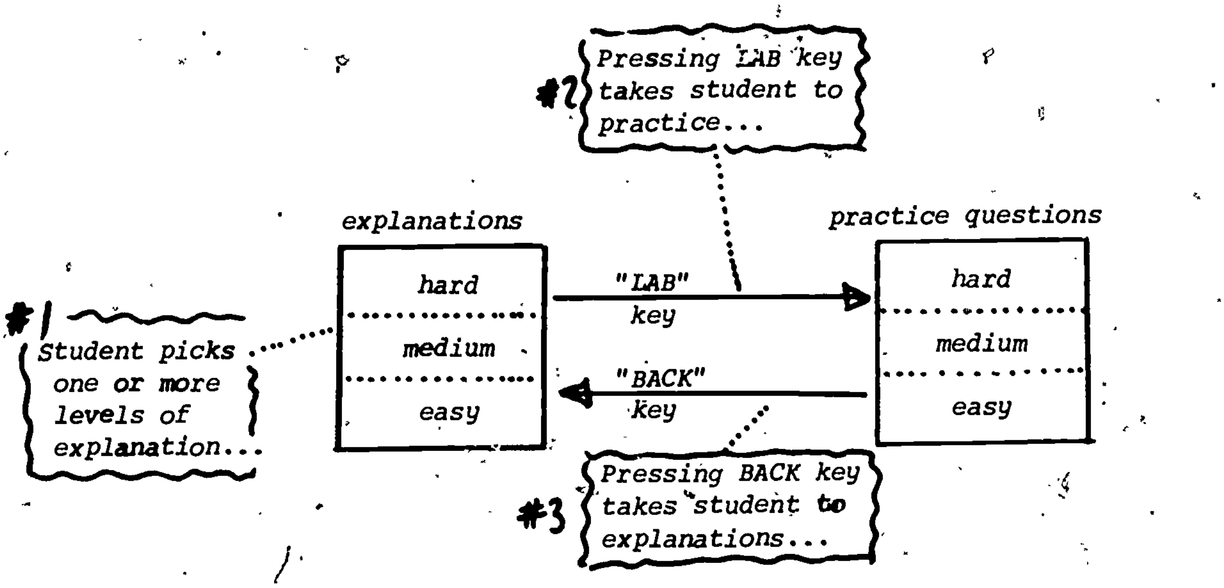


The specification of each frame may include a teaching-strategy, as well as a short description of the content of the frame. It is anticipated that a few (perhaps half a dozen) strategies for individual units would be provided with the first Hyper-editor, to determine just how much faster courseware can be created this way. In addition, authors may design their own frames for use as prototypes.

The Hyper-editor finally takes these specifications and creates actual code in the Tutor language. There is no barrier to creating code in other "hard core" C.A.I. languages as well, if this seems reasonable. The code thus created can then be immediately tested on the system the author is using, and once debugged, can be shipped to other systems for use by other teachers and students. It is essential that the system used by authors of courseware contain both the Hyper-editor and the ability to execute code for students, but it is not necessary that the eventual target (student) system contain the Hyper-editor.

Preliminary tests at Northwestern:

In an actual test case, this strategy for lesson creation has been partially tested by the author of this paper. A teaching strategy much like that to be used on Ticcit [3] was developed, and is described by the diagram at the top of the next page. The student may progress from one such frame to another under program control, or the student may pick frames (or short sequences of frames) from an index. The method of connecting the frames is determined by the courseware author during the interaction with the Hyper-editor. During the testing of this strategy, the regular system editor was used to copy a "paradigm" frame as many times as was necessary to create the finished lesson. Specific information for each frame was filled in as each frame was completed. The lesson thus created contained 18 sequences of 5 frames each.


#2
Pressing LAB key takes student to practice...
explanations
practice questions
hard
medium
easy
"LAB" key
"BACK" key
hard
medium
easy
#1
Student picks one or more levels of explanation...
#3
Pressing BACK key takes student to explanations...

The creation of these 90 frames, each with three levels of explanation and practice questions, took 35 hours when coded completely by hand, but only 15 hours when the editor was used to copy and modify the frames. This was a reduction of over 50 % in coding and editing time. Testing and debuggin time was not strictly measured, but in the case of the hand coded lesson, there were many programming errors, some not detected until students started using the lesson during the academic term, and in the case of the Hyper-coded lesson, the only errors were typing mistakes in the text to be displayed in each frame... there were no errors in the logical flow within frames.

There are no estimates currently made on how much time the setting of objectives and creation of flow-charts via the hyper-editor would add or subtract from lesson-creation time.

A Full-scale Test of the Hyper-system:

A Hyper-editor will soon be completed, capable of creating lessons using combinations of a few test strategies. Lessons will be produced in the Tutor language, and the Hyper-system will be written entirely in Tutor, so that it may be run as a major part of the MULTI-TUTOR.

Lesson-creation time using this experimental Hyper-editor will be measured by having both experienced and inexperienced author create lesson utilizing these strategies. Each author will create the lesson twice, once "by hand" and once by Hyper-editor. For inexperienced authors, the time necessary to learn how to program and how to operate the Hyper-editor will be measured separately, followed by repeated trials as the authors gain experience.

Advantages of this scheme:

There are numerous advantages to the creation of courseware using the Hyper-editor. First, automatic documentation of lessons is achieved as the Hyper-system records objectives and flow-charts while the courseware is being created. Second, the system can suggest high-interaction of proven strategies to an author, and will in fact keep track of an author's preferences. Third, the strategies are already debugged and the author of the courseware need only debug the lesson's content and overall strategy. And finally, data to be used for evaluation will be "automatically" collected by the code the Hyper-editor creates, and may be used by the author to review or restructure the lesson at a later date.

Summary:

This three-layered C.A.I. system will be a revolutionary means of helping authors create quality courseware in the future. The editing and coding process alone accounts for 50 % of lesson-creation time in C.A.I., and preliminary tests have shown at 50 % reduction in that time alone using the methods explained herein. Full implementation of this system will show us what decrease in lesson-creation time may be expected overall, and perhaps what type of author (experienced or inexperienced) will see the greater improvement.

Notes:

(1) Sherwood, Bruce Arne, "The Tutor Language", Computer-based Education Research Laboratory, University of Illinois, Urbana, IL; 1974

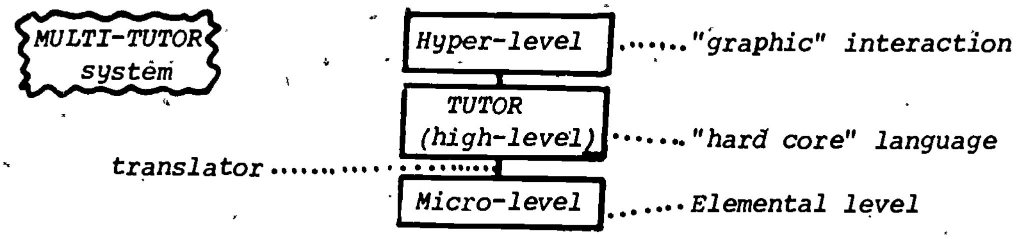
(2) Numerous papers in "Computing and the Decision Makers", EDUCOM, Interuniversity Communications Council, Princeton, NJ , 1974

(3) Bunderson, C. V. "The Ticcit Project: Design Strategy for Educational Innovation", Technical Report # 4, Institute for Computer Uses In Education, Brigham Young University, Provo, Utah , 1973

A Micro-Tutor to make C.A.I. Courseware Transfer Feasible

The companion papers to this pieve describe a three-layered Computer-Aided Instruction (C.A.I.) system called the MULTI-TUTOR. The "hyper" level, highest of the three, allows painless creation of C.A.I. courseware by inexperienced programmers. Below that level, the high-level language of the computer system is used (Tutor is used by MULTI-TUTOR currently). The hyper-level system interacts with an author, producing printed documentation and Tutor code as a result. This paper describes the operation of the lower, elemental level of this system, which is charged with the execution of the various high-level languages which will be acceptable to the MULTI-TUTOR.

In the best of all possible worlds, a C.A.I. author would wish to be able to transfer courseware to any C.A.I. system, regardless of the language in which it is coded. This is clearly impossible today, given the state of the C.A.I. art -- every new C.A.I. system designer invents a new C.A.I. language which is visualized as somehow "better" than all previous languages. What is really needed is a C.A.I. system which will adapt to the syntax of many languages, thus giving the ability to run practically any C.A.I. lesson on a single computer system. The following diagram is a complte sketch of the MULTI-TUTOR:



In the Plato design, the bridge between high-level and what's actually used when students are on-line is a program called the "condensor" -- it takes Tutor code, scans it for syntax errors, creates various tables which will be needed during execution and condenses most Tutor commands into concise, easily interpretable codes. When a lesson is needed for a student, it is the "condensed" version which is actually processed.

In the MULTI-TUTOR, the bridge between high-level and elemental-level is another program -- this time a translator which changes Tutor or another language into an elemental, very basic but totally comprehensive micro-C.A.I. language, which can then be interpreted in order to service students. Because the translator is driven by the syntax of the high-level language, it will be able to accept any C.A.I. language for which a translation table can be developed. (Part of this project will involve generating the rules for Tutor. Planit, Coursewriter and other languages in which substantial courseware now exists.)

Some specific points:

It is not anticipatedthat authors will actually be writing lessons using the micro-tutor language -- instead, they will continue to operate in their own favorite high or hyper-level languages, and the elemental language will make this possible.

Lesson transfer is not conducted in the micro-level language itself, though one might use it as a descriptive language in which other C.A.I. programs could be described. The power of the system is that it can accept lessons written in other language.

One is tempted to think of using Tutor itself as an elemental language, since it has all of the facilities the other languages have, but this will probably not be wise, since there are so many actions Tutor does not make explicit (such as repeating the question when the student gives a wrong answer). However, there is no barrier to starting with the functions Tutor embodies, since it is the most sophisticated of the languages which will be translated by the MULTI-TUTOR.

Properly implemented, the MULTI-TUTOR would even allow authors to create and write courseware in their own private languages, simply by providing rules which could be used to translate into the micro-tutor langvage.

Some preliminary work has shown that the functions needed at the micro level can be divided into about ten groups:

1) *Calculations,*
2) *Answer-checking,*
3) *Branching,*
4) *Special keys and functions,*
5) *Datasets,*
6) *Paging (units, frames)*
7) *Labelling of units or commands,*
8) *Displays,*
9) *Waiting for input,*
10) *Processing (manipulating) responses as characters.*

These groups of functions will be explored further in a subsequent publication.

Summary:

The Micro-Tutor is looked to as a courseware-transfer-facilitator. It will allow MULTI-TUTOR to accept several "hard core" C.A.I. languages at some future time, mapping each of them into an elemental representation which is actually interpreted by the MULTI-TUTOR. In this way, several high-level languages might be used on a single computer system, or authors might develop their own languages for personal use. Work done to date has revealed some broad categories into which commands in C.A.I. languages fall, and actual micro-tutor representations of these will be brought to light in a future publication.

2