DOCUMENT RESUME

ED 077 246 EM 011 174

AUTHOR	Kaplow, Roy; And Others
TITLE	TICS: A System For The Authoring and Delivery Of Interactive Instructional Programs.
INSTITUTION	Massachusetts Inst. of Tech., Cambridge. Dept. of Metallurgy and Materials Science.
SPONS AGENCY	National Science Foundation, Washington, D.C. Office of Computing Activities.
PUB DATE	Mar 73
NOTE	5p.; Proceedings, Seventh Annual Princeton Conference on Information Sciences and Systems, March 22-23, 1973
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Authors; *Computer Assisted Instruction; *Instructional Programs; *Interaction; Man Machine Systems; On Line Systems; Program Descriptions; *Program Development; Programing Languages
IDENTIFIERS	*Teacher Interactive Computer System; TICS

ABSTRACT

The Teacher-Interactive Computer System (TICS) is an on-line and interactive programing system for authoring interactive programs, particularly instructional programs. The system provides a fairly natural language, in which the author's statements for creating items in a program, for examining the structure and flow, for simulating its use by students, for modifying the existing description, and for making entries in a thesaurus/encyclopedia can be intermixed homogeneously. During the authoring process, the current specification of the program is stored dynamically as a structured data base, which includes automatically generated information relating to the interdependencies among items in the program and other diagnostically useful data. Implemented in a large, general-purpose time-sharing system (Multics), the TICS authoring system is complemented by a delivery system for student use of the program. It is also intended to provide automatic conversion of completed programs to alternate formats for implementation on other computers. (Author/PB)

Proceedings, Seventh Annual Princeton Conference on Information Sciences and Systems, March 22-23, 1973

TICS: A SYSTEM FOR THE AUTHORING AND DELIVERY OF INTERACTIVE INSTRUCTIONAL PROGRAMS

by

Roy Kaplow*, David S. Schneider†, Franklin C. Smith, Jr.* and William R. Stensrud*

Department of Metallurgy and Materials Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Summary

TICS (Teacher-Interactive Computer System) is an on-line and interactive programming system for authoring interactive programs, particularly instructional programs of various types. The system provides a fairly natural language, in which the author's statements for creating items in a program, for examining the structure and flow, for simulating its use by students, for modifying the existing description, and for making entries in a "thesaurus/encyclopedia" can be intermixed homogeneously. During the authoring process, the current specification of a program is stored dynamically as a structured data base, which includes automatically-generated information relating to the interdependencies among items in the program and other diagnostically useful data. Implemented in a large, general purpose time-sharing system (Multics), the TICS authoring system is complemented by a delivery system for student use of the programs. It is also intended to provide automatic conversion of completed programs to alternate formats for implementation on other computers.

Introduction

In this paper, we describe an interactive programming system: TICS (for Teacher-Interactive Computer System) in terms of its application to the authoring and use of computerized instructional programs, which we call Tutorials. This system, more than other programming languages and systems which have been applied to similar purposes,[1-4] treats the authoring of a Tutorial as a dynamic process which itself requires significant computerized assistance.

We regard Tutorial programs as defining an interaction which is primarily controlled by the computer (that is, by the author) but with the student being able to direct the flow either implicitly by his responses, or explicitly by direct requests. During that interaction an attempt may be made to stimulate the student by offering information, asking questions, and soliciting responses; the computer should also be able to respond to the student's questions, even if within a limited framework. A particular student will see a sequence of such actions; with the detail of the content varying in a manner which depends on his immediately preceding response, previous responses, and other parameters.

In writing a Tutorial, the author is concerned with a multi-dimensional object, for which he needs to consider both the overall design and specific details at different points--often at essentially the same moment. The notion that he might write the program by presenting a one-dimensional sequential list of "instructions" starting at one end of the object and going to the other, is not accurate, even though that view is so fundamental an aspect of most programming languages that non-professionals often identify it as the process of programming a computer. TICS, on the other hand, assumes that the author will move around within the ongoing description of his Tutorial. It also recognizes that the acts of describing items in the Tutorial, of examining the Tutorial, of trying it to see how it works, of changing it or adding to it on the basis of looking at it (in the general sense), and even of inserting data into an information data base for the student are all intimately related aspects of the one dynamic process of writing a Tutorial. Therefore, the TICS commands which relate to all of those different kinds of activity are available to the author at any time, regardless of the then current state of his program, and the various types of actions may be mixed in a natural way, according to his own predilection. Many of the available instructions are analogous to programming language statements, in that they specify actions which are to occur and the logical decisions which should be made when the student uses the program. However, unlike a programming language, in which statements are considered to be simply a sequence of program steps, the TICS system transforms the author's ongoing input into a data base of fairly general structure, which is called the "dynamic data base". The predefined structural form of the data base is a direct reflection of the system's conceptual model for a Tutorial, which is described in some detail in a later section.

In addition to providing a reasonably natural language to the author for the creation of a Tutorial and for reviewing its content, logic and structure, the system assumes the burden of managerial tasks, such as numbering items, recording interdependencies among items, and keeping track of structural incompleteness. The system also includes mechanisms to convert the author's

* Professor and staff members, respectively, Department of Metallurgy and Materials Science.

† Graduate student, Department of Electrical Engineering.

description of a Tutorial into a concise form for student use on one or more "target" machines, not necessarily the same type of computer on which the TICS author-system itself resides.

The Overall System

TICS is implemented at M.I.T. as a subsystem in the Multics[5] time-sharing system. Figure 1 diagrams some aspects of its organization and input-output modes. Within the author system, each Tutorial-in-process is embodied in one "dynamic-data-base", resident in Multics storage. The author can work on his description of a tutorial through an on-line terminal or via off-line media (e.g., cards, printer). He can allow a "try out" access to others who then can use, but not alter, the program, while it is still being developed and even if it contains structural errors! Any number of authors may work simultaneously, each on his or her own tutorial. An author may work on as many different tutorials as he chooses and, conversely, it can be arranged that any number of authors can work on one tutorial.

When a Tutorial is thought to be finished, the author initiates a transformation process (analogous to compilation), which we call compression. That process comprises the various steps of ordering the data base, searching for structural errors, extracting only those elements which are needed for the execution-time description of the Tutorial, and formatting them into a highly coded, compressed form. This is used in the delivery system, also implemented on Multics, in conjunction with a driver program. (Driver programs are also being written in IBM/PL1 and for a PDP 11/40 system.)

The Structure of a TICS Tutorial

We regard the structure of a Tutorial as being representable by a collection of nodes, interconnected by arbitrary numbers of brancnes. Each node can contain a small interaction, ideally a tiny conceptual unit in the author's plan. One node in a program has the initial attribute and acts as the starting point. Any number of nodes may be points at which the Tutorial ends. Generally, branches are specified explicitly, although conditionally (e.g.: if "such and such" is true, then go to node "such and such"). In addition, for certain types of nodes, or clusters of nodes, a branch out need not be explicitly identified but can be a return to the point where the branch into the cluster originated.

Assuming that the allowable contents of a node are adequate, these simple structural concepts allow a Tutorial to be very general from a purely structural point of view. Moreover, there are three distinct and important advantages in the node/branch concepts and the associated multi-level address space defined within each Tutorial: 1) It provides a convenient structural form for the dynamic data base, which in turn allows the provision of author commands which are both convenient and efficient in maintaining, manipulating, and investigating that data base. 2) For the author, the nodes might comprise intellectually meaningful units. Also, since they are readily distinguishable and clearly delineated locations in the overall structure, the nodes constitute a reference basis by which tne author (or a later modifier of the program) can move around within the already existing description. 3) Implementation of the target (delivery) system for student use is conceptually simpler with the nodal concept, and the inherent demarcation lines in the final program description provide an opportunity for efficient utilization of limited core storage in a special purpose, multi-user, time-sharing student system.

It is of equal importance to note that the nodal structure is not selected because of any presumed advantage it holds as a format for presentation to the student. Indeed, although an obviously "page-by-page" presentation can be written, such great variations in the contents of nodes are possible that students will not generally discern the node boundaries even if they are aware of the underlying nodal structure of the programs.

For Tutorial programs, we may envision nodes as having a specific internal structural pattern (which need not be filled) as indicated schematically in Figure 2. Within any one node there may be 1) execution of called subroutines, 2) outputs to the student, 3) acceptance of a response from the student (if one was anticipated), 4) analysis (mapping) of the response with respect to a list of anticipated responses, and 5) testing of "condition statements" which, if true, cause associated sequences of actions to be carried out. The conditions to be tested can relate to the student's response in the current node, to responses in other nodes, and to values of variables. The action sequences can include calling subroutines, printing statements, writing entries in a report file, doing mathematical operations on arithmetic variables, executing "return" type nodes or node clusters, printing hints (and getting other responses), and (sooner or later) branching to another node.

In addition to the description for those contents, which specify what the program should do when the student is using it, each node may have a number of additional items associated with it, while the author is working on the program. These include a name, a number, a documentation comment, a self-reminder author message, a system-maintained set of warning and error messages relating to changes made in interdependent items, a system-maintained list of all items in the node on which other items depend, optional attribute specifications which relate to the interpretation of the student's response, and one or more keyword phrases. Using a subset of the node-keyword pairs (which is then included in the student's version of the program), the teacher can identify those points in a Tutorial to which a student may arbitrarily skip, and at the same time provide the "map" for the student to appreciate what those points are about.

OVERALL STRUCTURE OF THE TICS SYSTEM

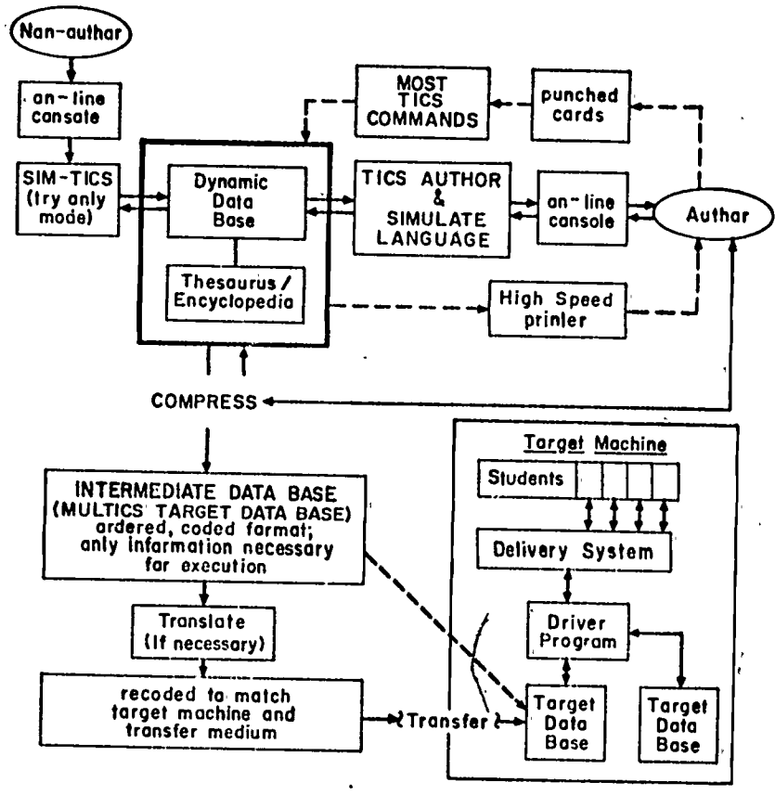
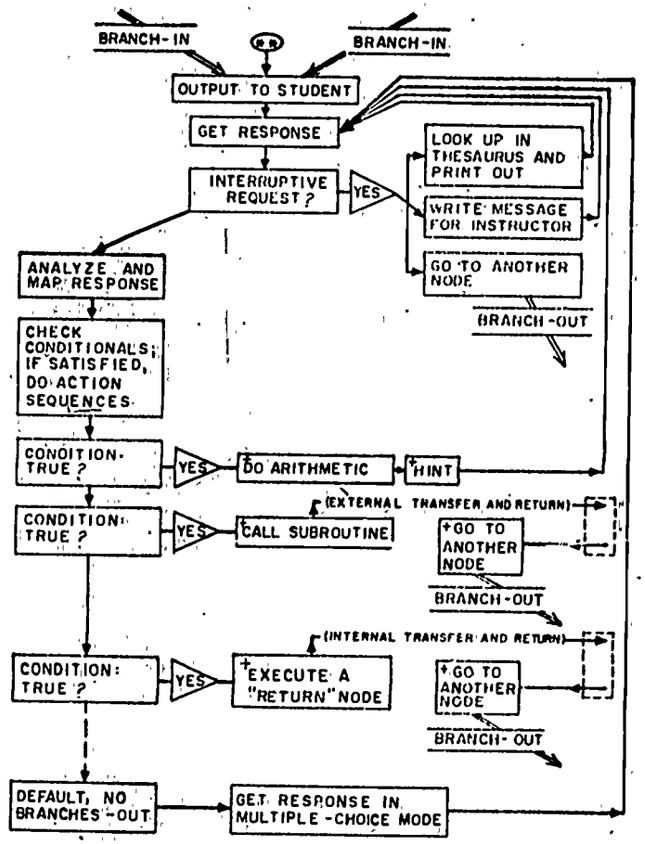


Figure 1



(+) These are examples of actions; sequences can contain any number of actions.

(**) Subroutine calls may be inserted here as well as in action sequences

SCHEMATIC DIAGRAM OF THE INTERNAL STRUCTURE OF A NODE.

Figure 2

There are, additionally, implicit items in the description of every node because of certain operational conventions which are followed when the Tutorial is being used: 1) If a response does not result in a hint or a branch-out, the system defaults to a multiple-choice mode, in effect asking the student to select among the responses anticipated by the author. 2) If a hint is given, the associated statement is checked off, so that the same hint is not given again inadvertently. It may be noted that these conventions eliminate the possibility of "looping" or of a dead end occurring within a node, providing that at least one anticipated response (or an always true conditional) necessarily leads to a branch-out. This condition is readily checked and is monitored by the system. Another implicit item in the Tutorial is the optional recording of the history of the student's path, including the full text of his responses. These data may be useful to the author or to the instructor of the related course, if the use of the Tutorial is formalized.

There are also three types of global items associated with each Tutorial which apply throughout, rather than to a specific node: 1) A data base for author-specified numerical scalar, numerical array, and character variables, and for a number of system-maintained variables, which may be pre-initialized or derived from student input, included in text output, manipulated by arithmetic (and character) operations and subroutines, and used in making decisions. 2) An organized file of textual information, called a thesaurus/encyclopedia. Words and phrases can be linked to one another in the sense of a thesaurus and associated with descriptive text in the sense of a dictionary or encyclopedia. In use, the student can access the information from any point in the tutorial through the use of "interruptive" requests. 3) There are seven types of interruptive requests with which the student can change or temporarily interrupt the flow. Three of these require no action by the author: the student can back up to a point where he gave a response previously; send a message to whomever is in charge of the use of the tutorial; or stop the session, with the option of continuing at a later time. Three more requests are always implicitly available, but are useful only if the teacher supplies appropriate data. These allow the student to search through and look up items in the thesaurus/encyclopedias; search the keyword phrase list; and jump to points identified in the keyword list. In addition to these, the teacher may specify subroutines to be made available and their names may then be used as interruptive requests, with or without student-supplied parameters.

Authoring a Tutorial

The author is always considered to be at a certain node in the Tutorial, the working node. The working node may be relocated to any node by the use of a simple command, and the consequent ease of moving the site of operation among the nodes helps to make the description of a non-linear Tutorial a natural process. With other commands, the author can create specific items within the Tutorial, generally at his working node. The elements of a node may be created in the order in which they will be executed or in any other order that the author finds natural. Since any one statement may call; either explicitly or implicitly, for the creation of a number of new items in the program, the system informs the author of the specific entries made in the dynamic data base and of the unique identifier assigned to each item, which may be used thereafter to refer to the item without repeating its complete specifications. The system gives warnings and may seek verification of possibly unexpected creations (e.g., the implied creation of a new variable or of an anticipated response in another node) and input statements are checked for consistency with the existing data base as well as for language syntax. When an error occurs, a descriptive message is printed, and the author may switch to a general purpose editor to fix the statement.

The same editor is called at the author's request to alter any previously specified textual entry in the program. Other modifications to existing entries may involve deletions, rearranging the order of things, changes in the logical structure, or substitution of one item for another and appropriate commands are therefore provided.

"Shorthand" features are provided to give the language added convenience. An author may define an input shorthand for commands or for text which he uses often, and previously specified text may be used, by reference, as input in creating new entries. The author can also control the verbosity of the computer feedback and, if he chooses, switch to a block input mode to vary his pattern of interaction with the computer. Moreover, all facilities in the system except those which obviously require the author's live presence (such as simulation), can be used in an off-line, card-input mode.

To help the author keep track of the interrelationships among items in the tutorial, a cross reference table is maintained for each node. These tables are available to the author and are also monitored automatically to determine what effect modifications have on items elsewhere in the Tutorial; when an item is altered, an appropriate warning entry is made in any affected node(s). If an item is deleted from the program, such that some other item is put in error, a non-deletable error entry is made in any node(s) containing the erroneous item(s), and the deleted entry is actually saved in "ghost-like" form, so that it may be reinstated, if desired. Such error entries can be removed only by correcting the erroneous conditions.

The ease with which the original author and subsequent contributors can refine a Tutorial depends greatly on their being able to inspect and review its contents. The TICS system therefore provides a variety of tools for viewing the structure and the content of a Tutorial, in coarse or fine grain, on the author's console or via a remote high-speed printer, in print or graphical form. Another set of commands is provided for focusing on the

logical structure of the Tutorial; that is, for looking at possible paths through the node/branch structure. One such command is the tree command which allows the author to view the branching structure starting or ending at a given node. A trace command "plays through" the outputs, student responses, and conditional choices involved in a path through a set of nodes. Another command yields a block diagram of the internal logic of a single node.

An important part of the authoring process is for the author to "see how it runs"; that is, to execute the Tutorial as a student. For this purpose, the system includes a mode of operation intended to simulate the execution of a program as it will appear to the student, but which works on the author's dynamic data base, which may be structurally incomplete and erroneous. (Such conditions are detected during simulation and brought to the author's attention.) There are, moreover, two modes of simulation: a) Student (or "demo") mode, for which the target system interaction is emulated as precisely as possible and which can provide actual user feed-back during the entire process of writing the tutorial. b) Teacher (or "non-demo") mode, in which the system not only presents the described interaction, but also identifies each node as it is entered through the branching sequences, prints the error, warning and reminder messages which happen to be attached, and makes additional comments about incomplete states. In addition, during the course of the simulation, the author then has available a number of commands for the purpose of examining and setting values for variables, and for controlling the simulation. He can set detailed "stop-points" within nodes, which will cause a halt in the simulation and allow examination of the state of affairs at the instant. Throughout, the author may use any of the standard TICS requests to create, display, examine or alter any part of the Tutorial description.

Anticipated Responses and Response Analysis

For the interaction format described earlier which has the flavor of a "conversation" between the computer and the student, and especially if the student responses are the prime determinant of the program flow, specification of anticipated student responses are a central aspect of the authoring process. If, in addition, free format student responses are desired, rather than (say) multiple-choice selection, response analysis (or interpretation) with respect to the anticipated responses is a critical function of the delivery system.

The generalized response analysis problem is made more tractable by the structure of the Tutorial which allows each individual analysis to be made in a very local context, and reduces the problem from "what does it mean?" to "does it mean the same as one of the anticipated responses". It is worth noting, as a related point, that there is not much advantage gained by a response analyzer which "understands" responses for which the rest of the Tutorial is not prepared. In a real sense, therefore, an author's success is more dependent on the degree to which he becomes not only aware of but also responsive to the meaningful range of student responses in each node, during the dynamic process of design and trial of a Tutorial.

While trying not to involve the system or the author deeply in questions of language structure or analysis, requiring certain simple specifications to indicate alternatives for the response comparison process has been useful in allowing reasonably natural student responses. Thus, a node may give the student a multiple-choice presentation directly or look for a free-format response; or, it may seek a response which is actually a list of responses. It may seek no response, or re-interpret a previous one with respect to a different set of anticipated responses. Each anticipated response carries its own instructions for the response comparison routines; these may indicate, for example, that the expected response is a number between specified limits, or an algebraic expression or equation, both of which require a mapping very different than for text. Text responses may be finely detailed in terms of pieces which should or should not be included, in terms of listing synonomous or alternative forms, in terms of the exactness of match required, and in other respects. Additional alternatives regarding response analysis are implicit in the author's option of using subroutines to act directly on the input text.

Acknowledgments

We would like to thank a number of persons who have contributed to the design and implementation of the system. Dr. John Brackett, Dr. Alan Campagna, John P. Linderman, David Pettijohn, Seth Cohen, Lee Scheffler, Paul Leach, Richard Goldhor, Geoffrey Bunza, and Gary Stahl. This work was sponsored by the National Science Foundation, Office of Computing Activities.

References

1. Swets, J. and Feurzeig, W., "Computer-Aided Instruction", Science 150 (1965); also see Feurzeig, W., Computer Systems for Teaching Complex Concepts, Report No. 1742, Bolt, Beranek and Newman, Cambridge, Mass. (1969).

2. Feingold, S. L.: "PLANIT - A Flexible Language Designed for Computer-Human Interaction", Proc. AFIPS 1967 Fall Joint Computer Conf. 31, pp.545-552, Thompson Book Co., Washington.

3. IBM Corp. Coursewriter III for System/360, Version 2, Application Description Manual. No. GH20-0587-1 (3rd ed., August 1969).

4. Computer-Based Education Research Laboratory. Tutor User's Manual. University of Illinois, Urbana, July 1971.

5. F. J. Corbato, J. H. Saltzer, C. T. Klingen, Multics--the First Seven Years, AFIPS Proceedings, 40, p. 571, Spring Joint Computer Conference (1972); E. I. Organick, the Multics System--an Examination of its Structure, M.I.T. Press (1972).