DOCUMENT RESUME

ED 078 646 EM 011 175

AUTHOR	Kaplow, Roy; And Others
TITLE	Computer Assistance for Writing Interactive Programs: TICS.
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	Apr 73
NOTE	14p.
JOURNAL CIT	ACM SIGCUE Bulletin, Computer Uses in Education; April 1973
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Authors; *Computer Assisted Instruction; *Computer Programs; Computers; Computer Science; Instructional Media; *Instructional Programs; Interaction; *On Line Systems; Program Descriptions; Program Development; *Programing; Programs
IDENTIFIERS	*Teacher Interactive Computer System; TICS

ABSTRACT

Investigators developed an on-line, interactive programing system--the Teacher-Interactive Computer System (TICS)--to provide assistance to those who were not programers, but nevertheless wished to write interactive instructional programs. TICS had two components: an author system and a delivery system. Underlying assumptions were that instructional programs required complex logical structures and could not be written in one linear sequence, that they must be modifiable, that authors should be able to use on- and off-line modes, that programs required communication with and some control by students, and that programs should allow different modes of interaction. TIC's structural framework consisted of interconnected nodes, each containing a linear sequence of actions. A dynamic data base was maintained and the author language served as the interface between the author and the system's routines: for creating, mapping, testing, and modifying programs, for viewing their structures, and for developing auxiliary information stores. Automatic maintenance features included space allocation for the growing data base, the assignment of identifiers for new items, and the notation of errors. (PB)

ACM SIGCUE Bulletin - Computer Uses in Education - April 1973

Computer Assistance for Writing Interactive Programs: TICS

by

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

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

The need to make it easier to create the "software" associated with the general use of "new" technology for education is being increasingly recognized. During the past few years, we have concentrated on a part of the computer aspect of the technology, to implement a partial solution of the software problem within that narrower framework. In this paper we describe an on-line and interactive programming system (TICS,[1] for Teacher-Interactive-Computer-System), which is aimed at facilitating the authoring of interactive, instructional computer programs by persons who are experts on the subject matter being addressed, but not necessarily programmers. To that purpose, the system provides a greater degree of computer-assistance for the authoring process itself than has been afforded in earlier languages and programming systems of similar orientation.[2-5] TICS is implemented within the M.I.T. Multics time-sharing system[6] in two components: an author system and a delivery system. The former provides the tools for writing, investigating, editing, and trying out programs. The latter provides a special environment for student use of the programs.

---

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

† Graduate Student, Department of Electrical Engineering.

The system reflects a number of basic premises about the nature of instructional computer programs and about the needs of the authors. The following list, although not inclusive of all the assumptions, indicates the considerations on which the overall design and specific features of the system are based.

Premises

1. Instructional programs will generally require complex internal logical structures; the internal structure may be sufficiently complex that the author himself will not be able to keep track of it without assistance.
2. It is not sensible to think of an author writing such a program as one linear sequence of statements.
3. During the authoring of such programs, it should be easy to include a process of trial-and-modification, using feedback from students.
4. It should be feasible for persons other than the original authors to modify and augment programs, possibly months or years later.
5. Authors may prefer to use an off-line, as well as an on-line mode, in arbitrarily mixed combination, while developing one program.
6. Such programs will depend strongly on two-way communication with the student; for example, we need to have a concern about student input and authour output which goes far beyond "read-from-terminal" and "write-to-terminal" instructions.
7. In using such programs, the student should not be "boxed-in" but should have some <u>explicit</u> control over the interaction flow.
8. Authors may want to use various combinations of different interaction modes, such as tutorial, socratic, enquiry, questionnaire, gaming, problem-solving, simulation, and drill.[7] This implies the need for a number of features, including an information data base, a data base for computational parameters and results, and access to general purpose subroutines.

9. For such programs, there are certain repetitive execution-time features that can be automated, relieving the author of concern over them, unless he chooses to override them.

There are always a variety of viewpoints from which a complicated computer system can be described. We think it is useful to consider TICS, in the light of the preceding premises, and in terms of four particular aspects: 1) the structural framework which it provides for the program being authored; 2) the data base in which the current description of the desired program is stored; 3) the author language for specifying and studying the program; 4) the "automatic" maintenance and execution-time facilities.

Structural Framework

The provision of a structural framework for the author's description of what should happen during the student's interaction with the program is based on a number of the given premises, especially the first, second, fourth, and ninth. The mere existence of a structural framework can facilitate the author's design process. A structural framework can be particularly useful if 1) its subunits can correspond to conceptual units in the author's design, 2) it facilitates a multidemensional addressing scheme for referring to individual items within the program (which in turn makes it easier to find one's way around the program), and 3) it facilitates implementation of an efficient delivery (execution-time) system for the students. Generally speaking, the more tightly defined the structural framework, the more the authoring process can be made easier through computerized assistance and automated maintenance facilities. At the same time it is necessary to minimize restrictions on what the author can do and certainly to avoid a set format for the student interaction.

In the TICS system we ask the author to imagine that his program consists of a collection of nodes, interconnected by arbitrary numbers of branches. The descriptions for the events which can occur during the student's execution of the program are contained within the nodes, and the flow of the interaction is determined by the branches taken during an actual execution.

Looking inside of a node, one sees the next level of the structural framework. This is described most simply as a linear list of action sequences to be conditionally executed, with the possibility of implicit, system-automated actions superimposed. Each action sequence is specified in the form:

if <condition> is true, then <action> and <action> and ... <action>.

The specific actions can include: outputting something to the terminal and getting a student response (e.g., an "ask" or a "hint"), outputting to the terminal and going directly on (e.g., a "print"), doing mathematical or character operations on variables, writing entries in a report file, calling subroutines, and branching to another node. The <condition>'s can be null, in which case the action sequence is always executed if the execution progresses to that point in that node. If a condition is specified, it can depend on: the match between anticipated responses and the student's response in the current node, on responses given elsewhere in the program, on the values of variables, and on elapsed-time. Among the implicit operations are the following: if conditionals are phrased in terms of one or more anticipated responses, then a student response is sought automatically after the initial "ask" and subsequent "hints"; the full list of conditional action sequences is repeated each time another response is sought in the same node, except for the initial output and sequences containing hints (this avoids the possibility of "looping" within a node); if a student response does not lead to a hint or branch, the system creates a multiple choice offering out of the responses anticipated. These conventions imply that a "dead-end" cannot occur during execution if at least one anticipated response (or a null condition) necessarily leads to a branch; that condition is monitored by the system automatically, for every node.

There are actually two types of nodes as regards the internode structure; we might call these ordinary and return type. For the former, all potential branches-out must be given explicitly (e.g., "go to node such-and-such"). For the return type, potential branches are either explicitly to other nodes of the same type, or returns to the internal point in the ordinary node which initiated the "call" to a return-type node. This allows what might be thought of as an internal sub-process, consisting of a cluster of return-type nodes. These are useful for such purposes as 1) a single interaction which needs to be called from different points in the program, 2) interactive "hints", i.e., a many-node interaction effectively contained in a single action sequence, and 3) to define a "sub-process" execution-time duration for data allocation.

Two formal structural limitations are imposed, mainly to make it feasible for the system to monitor the internode structure of the program for display and error-checking purposes. The first is that "calls" to return-type nodes can only be made from ordinary nodes. The second is that there are no computed branches; that is, no branches of the form "go to variable", where the value of variable, to be computed at execution time, might be any node in the program.

The Data Base

The system dynamically maintains an on-line data base description of the program being authored. The primary directory for the data base is a table of node descriptor blocks; each node block contains pointers to rings of entries for the specific execution-related items which nodes contain, such as anticipated responses, branches in and out, conditional action sequences, and each of the individual types of actions. The data base also includes, in both the table and as additional rings, other information which is useful for the authoring process--although not needed for an execution-time description. The author can attach to each node, for example, a name, documentation comments, personal reminders, and keyword phrases. The system automatically maintains flags which indicate the presence of active intra-node conditions (such as incompleteness, errors, or attached messages) and rings of inter-node data such as item cross-references and errors (or possible errors) caused by edits on cross-referenced items.

One important aspect of the dynamic data base is its accessibility to the author through an off-line (e.g., punched card) job, as readily as when he is at a terminal. The off-line facility of the author system allows the entire author command language to be used, except for the portions which absolutely require the author's immediate presence. The mode may be used whenever desired, and mixed with on-line work. Although it utilizes punched cards, the off-line mode differs from usual "batch-processing", in that the "deck" never represents more than the new work, the instructions to add new things or to modify old items in the internally stored dynamic data base.

The Author Language

The TICS author language is the interface between the author and the operators (or routines) which the system provides: for creating a program; for providing a map of the program--in the form of keywords and other documentation; for viewing its structure; for trying out the program--as a student would see it; for making changes in its content; and for creating an auxilliary information store for the student, like a dictionary/thesaurus.

The chosen structural framework, and its realization in the dynamic data base, leads to a multidimensional addressing scheme for the items in the program as a natural consequence. Using his own identifiers and those assigned by the system, the author is able to work in terms of specific items, which is especially useful when changes or additions are required. Thus, for example, a particular item in the data base--say an arithmetic assignment action--might be referred to as the "third action in the second conditional action sequence of the node named wave_set". In another context, the identical item might be referred to as "the fourth arithmetic assignment in the twenty-fifth node".* The same addressing schemes are also used by the system, in responding to author queries, for example, such as "Where is the variable 'var' used?"

Creating a Program

An author can tackle the program all over at once if he chooses, since commands of all types can be mixed in any order, and the items being addressed can be anywhere in the data base. However, the system does assume that he will tend to work on a node-by-node basis, and uses the concept of an author-selected working node. That is, the author selects a node and subsequent commands are assumed to refer to that node until he changes it. The commands for creating the program itself are probably the simplest part of the language; the following sequence is intended simply to give its flavor and is shown without the shorthand conventions that comprise the true language.

set working node "name_1"

ask "What is the date of Washington's Birthday?"

if response = "february 22" then print "Yes, that's right!" and go to node "name_2".

if response = "february 19" & response (in node 17) = "old enough" then do flag = 1 + flag and print "Yes, in America they will even change a President's birthday to make a long weekend." and goto node "name_3".

if response = "february 14" & variable > 2.3 then call subroutine_name (flag, variable) and hint "That's Valentine's Day, ¬name. Please try again".

In an actual on-line session, the system responds to each statement with a notice of the identifiers assigned to each new item created; it gives warnings (and may seek verification) of possibly unexpected creations (e.g., of a new anticipated

* In both instances, of course, a short-hand code would actually be used.

response in another node); and it checks for consistency with the existing data base as well as for language syntax.

Although anticipated responses can be specified as simply as illustrated above, a variety of alternatives have been provided for greater sophistication. Special allowance is made for numeric and algebraic responses, and texts can be finely detailed in terms of exactness required, parts which should or should not be included, synonomous forms, and other respects. Subroutines may also be used to operate on the student's responses. In addition, nodes may be designated to give multiple-choice presentations (among the anticipated responses); to seek a free-form response; to re-interpret a previous response with respect to a different set of anticipated responses; or to analyze a response in terms of its being a list of responses.

For the text output side, the system tries to provide simple, format-free options, like that illustrated above, with a simple code for inclusion of variables (the ¬symbol, illustrated by ¬name in the above example). At the same time, format control is available when desired. In accordance with the general philosophy of the system, it trys to minimize the difficulties associated with setting up and making alterations on output texts, whether or not the author gives his own format specifications. In sum, this requires many of the features of a full text processing system, including system defaults for execution-time formatting for variables and automatic margin line adjustment. An analogous sophistication is needed for graphical and "line-drawing" output.

Providing a Map of the Program

As indicated earlier, the system provides means for attaching documentation to individual nodes in the form of comments (for long term reference) and reminders (which are printed out whenever the node is entered, either as the working node or during a simulation). Unique names may also be attached, and any number of keyword phrases. The language includes commands for attaching keywords to nodes with individual "hierarchy" levels and for using the keyword list as a multi-level node directory, if desired. A subset of the keyword-node assignments can be specified by the author to be a map for the student of the points in the program to which he can jump arbitrarily. The selected list serves both to tell the student what the specific parts of the program are about and also to control the student's mobility.

Viewing the Structure of a Program

The system provides a number of commands for examining the existing data base and thereby for studying what will happen during a student's execution of the program. The content of the program may be displayed in tabularized formats on the author's console or via a remote high speed printer. Graphical and printed displays of the internode (tree) branching structure, and of block diagrams of intranode structures can be obtained. The author may request a "trace", that is, a playing through of the outputs, student responses and conditional action sequences that would be involved in a path through a given set of nodes. In addition, the system includes a mode of operation in which the author can play the role of a student, while the system simulates the execution of the program, starting at any point. This can be done while the program is structurally incomplete and even erroneous; the system detects and reports such conditions during the simulation. The author has available a variety of commands for controlling the simulation, for examining and setting variable values, and for setting "stop-points" within nodes for halting the simulation to allow examination of the instantaneous state of affairs. He may also interrupt a simulation to use any of the standard TICS requests to create, display, examine or alter any part of the program. The simulator can also be run in a mode in which the auxilliary information output and the user control capabilities are inhibited; this is useful for providing real student feedback during the entire authoring process.

Editing a Program

The problems with editing a complex program are not completely solved by providing the needed operators, text editor and author language alone. The difficult part of the job for the author (and especially for subsequent modifiers of a program) is to keep track of the interrelationships among different parts of the program. Often, a change (especially a deletion) made in one place will have ramifications elsewhere. Obviously, it is not possible to keep track of all relationships automatically, although the author can do a lot in that regard with the means provided for documentation and mapping, and for studying the program structure. On the other hand the system can and does keep track of all of the explicit cross-references among items, both intra- and inter-node. These tables (rings in the data base) are available for author examination and--more importantly--are referred to automatically by the system whenever changes or deletions are made. When alterations cause certain or possible errors, the affected items are flagged (with back pointers to the altered or deleted item). When deletions are the cause of errors, the deleted item is actually saved (as a ghost) for subsequent reinstatement. Even if the author does not take the initial warnings seriously, error and warning messages remain attached to the affected nodes, and those which represent structural incompleteness will demand subsequent correction.

Creating the Dictionary Thesaurus

The author can build up to five separate dictionary/thesaurus information stores. These are subsequently made available to the student, along with appropriate look-up requests, as an integral attachment to each program. The author language allows the author to specify, modify and edit entries, consisting of words and phrases linked to one another in the sense of a thesaurus, and to attach descriptive encyclopedia-like texts to each such list. Any one word or phrase can be included in any number of entries, each with its own descriptive text. These data are primarily intended for use by the student in an on-line request mode, while he is using the program. It is recognized, however, that the data might also be conveniently used in hardcopy form, so the system also provides a well-formatted print-out option for making duplication masters or high-speed printer copies. In a classical application, these might be analogous to a glossary in a textbook, but they can also serve as indices, catalogues and even as more general data bases. One interesting application has occurred in a language program, for which three dictionaries were used to list the meanings, pronunciation, and etymology, respectively, of German words used in the program. It seems, in fact, that such data bases, with the look-up requests that the system supplies, can themselves comprise an important type of learning interaction, with very little program structure superimposed by the author.

Automatic Facilities

Many of the automated facilities have been referred to already, particularly in the discussion of the author language. The most important author-time dynamic system maintenance features are: to make the allocations of space required for the dynamic data base as it grows; to assign identifiers for items as they are created; to note and record all interconnections and cross-references as they are created; to monitor the structural completeness of each node; and to note, warn about and flag errors or potential errors caused during editing.

Other maintenance operations are essentially automatic as far as the author is concerned, although he must request them. These include ordering the data base (i.e., to put everything in systematic order, to increase operational efficiency and decrease storage requirements); checking the data base for integrity (i.e., to guard against computer errors); and compressing the data base (i.e., to produce a highly coded copy of only the portions required for the execution of the program).

After the compression step, the entire delivery system is essentially automatic from the author's point of view, but much of the control is nonetheless at his discretion, on a program-by-program basis. The system provides, for students, a limited-access environment with [illegible]it..s for using TICS-authored programs and any other Multics facilities that the author of a program so stipulates. The system maintains an awareness of all available TICS programs; it provides a directory and choice of programs for students; it maintains the individual sets of data for each student's execution of each program; it creates report files, optional history-of-execution records, and message-to-teacher files for each student and each program.

During execution the system carries out a variety of implicit operations within a program itself (such as creating a multiple choice offering when a student's given response leads nowhere). It also gives the student a number of "interruptive" requests to use. These are for 1) looking up information in the dictionaries, 2) searching through the keyword map, 3) jumping to a point specified by a keyword phrase, 4) backing up to a previous response of his in the interaction, 5) sending a message to whomever is in charge of the use of the program, 6) calling any subroutine or Multics subsystem that the author has stipulated he should be able to use, and 7) stopping the session, with the option of continuing later.

Concluding Remarks

We believe that TICS incorporates a number of important features which make it easier to write interactive programs, particularly programs intended for instructional purposes. Many of these features have evolved during the development of the system itself. Just as we would advocate that an instructional program should be developed with a lot of feedback from students; we have taken advantage of the experiences of authors who have tolerated working with a changing system. In part, this has been possible because most of the changes could be handled without disrupting the authors' work, even to the point of automatically changing existing data bases for partially completed programs.

We know that major refinements and additions will continue to be made to the system. On the other hand, these are only details in comparison with the fundamental issue of focusing on the problem of providing as much computer assistance as possible to the authoring process.

Acknowledgements

This work has been supported by a grant from the National Science Foundation, Office of Computing Activities. We would also like to thank a number of persons who have contributed to the design of the system: Dr. John Brackett, Dr. Alan Campagna, John P. Linderman, David Pettijohn, Seth Cohen, Lee Scheffler, Paul Leach, Richard Goldhor, Geoffrey Bunza, Gary Stahl, Samuel Desch, and Dr. Melvin Rodman.

References

1. Roy Kaplow, D. S. Schneider, F. C. Smith, Jr., and W. R. Stensrud, TICS: The Author Language and Instruction Manual, Massachusetts Institute of Technology (1971); TICS, A System for the Authoring and Delivery of Interactive Instructional Programs, Seventh Annual Princeton Conference (1973).
2. 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).
3. 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.

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

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

6. 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).

7. Roy Kaplow, S. H. Desch, D. O. Pettijohn, M. H. Rodman, and F. C. Smith, Jr., Illustrations of Conversational, Inquiry, Problem-Solving and Questionnaire Type Interactions within the TICS System, Seventh Annual Princeton Conference Proceedings (1973).