Journal of Microcomputer Applications (1989) **12**, 293–317

HyperCOSTOC: a comprehensive computer-based teaching support system

F. Huber, F. Makedon* and H. Maurer

*Institutes for Information Processing Graz (IIG) of the Graz University of Technology and the Austrian Computer Society, Schießstattgasse 4a, A-8010 Graz and *Computer Science Program and the Computer Learning Research Center, The University of Texas at Dallas*

We propose the main architectural features of HyperCOSTOC, a new computer based teaching support system, a number of aspects of which are currently being implemented. We call it HyperCOSTOC because it is based on a database of lessons called COSTOC and has facilities which have a hypermedia flavor (i.e. navigation, browsing, cross-window communication, and others). However, unlike hypertext and other electronic information systems (e.g. database management systems), HyperCOSTOC is an advanced CAI system which can retrieve and process, in a non-linear fashion, structured molecules of instructional or information material networked together. HyperCOSTOC uses a mixture of presentation-type-CAI, object-oriented programming and query language techniques for university teaching and research. It provides a user interface which includes selection mechanisms, feedback mechanisms and input-output devices.

HyperCOSTOC aims at a practical and viable approach within a university setting. It comprises tools for the creation, distribution, and usage of a variety of courseware or information in a fashion which is modular, technology independent, easy to maintain, and which includes drill, self-test, exam and other modules. Hypertext largely depends on the power of the computer to traverse complex networks of information and to support advanced word processing while freely linking ideas on-line. HyperCOSTOC is supposed to be as hardware-independent as possible and geared towards easy integration with a variety of instructional environments: as a support system to traditional lecture-style instruction as well as for the development of exploratory research environments.

The COSTOC system (a simple minded precursor of HyperCOSTOC) is currently being used in about 20 educational institutions. COSTOC includes a database of hundreds of one-hour lessons, with over 250 on computer science topics. Starting from this courseware basis, HyperCOSTOC is supposed to add many new facilities, including better navigation, browsing, window cross-talk as well as aspects of student modeling. It should be emphasized, however, that the first implementation of HyperCOSTOC will put little emphasis on intelligent-CAI (ICAI) techniques (e.g. intelligent tutoring, intelligent student modeling) because we believe that further developments are necessary before ICAI will become as broadly applicable as other aspects of CAI.

1. Introduction

In one of our earlier papers we described COSTOC (see [1]), an international project which involves the production and implementation of computer science courseware within a distributed and microcomputer-based university environment. Up to now, the word COSTOC has been an acronym for *CO*mputer *S*upported *T*eaching *O*f *C*omputer-Science. Since the techniques inherent to COSTOC go beyond the domain of computer science, this acronym is now also often interpreted as: *CO*mputer *S*upported *T*eaching? *O*f *C*ourse!

0745-7138/89/040293+25 $03.00/0

Using computer systems to support teaching is an idea that is about 30 years old and into which great effort and funding has been invested . . . with small returns. The advent of inexpensive yet high-quality microcomputers and workstations has changed the situation dramatically. A flood of new efforts is now visible both in North America and in Europe. Among these efforts, significant differences are noticeable in their scope, their philosophy, their assumptions on hardware availability, new applications and the new breed of computer scientists involved.

A major recent effort in computer-based training has been in the area of presentation-type CAI (PT-CAI for short). PT-CAI is a hybrid of *ad hoc*-frame-oriented (AFO) CAI systems and information-structure-oriented (ISO) CAI Systems, as originally defined by Carbonell [2]. The main motivation for PT-CAI is that only by making the production and maintenance of courseware as simple as possible can large databases of courseware be viable in a self-supporting mode. The lack of a critical mass of courseware material and the lack of a courseware/software basis for evaluation and testing within any one given domain have been the main reasons for the failure of similar CAI projects in the past. (For a good survey of CAI and ICAI see Rickel [3]). Based on this assumption, COSTOC, the predecessor of HyperCOSTOC, was developed [1].

Given the problems outlined above, the important contribution of the ongoing COSTOC project, is a massive and growing database of high-quality lessons, mostly in computer science, created by experts around the world. Lesson material consists of text and graphic segments, combined with a variety of different types of frames (help, reference, question–answer, etc.) For reasons of portability and maintainability, the COSTOC lessons do not usually include programs or other media, although they have the capability to incorporate such. By mid 1988, some 250 such computer science lessons were available (in English and German), and this number is expected to grow to 500 computer science lessons by 1989. Several labs are currently running at various universities (in Canada, US, Germany and Austria) with the COSTOC database incorporated in the curriculum. Typical labs consist of a fileserver (a PC with a large hard disk, or a Micro-Vax), containing the database of lessons. Student stations are attached directly to the fileserver or via LAN or another network.

COSTOC labs currently run like a computer science electronic library of lessons. They are virtually maintenance free, students and faculty find them easy to use, and evaluations have been on the positive side, as Makedon & Maurer [4], Maurer & Makedon [5], and Ottmann [6], have shown. However, with the expanded involvement of faculty and students in COSTOC labs, the limitations of this type of system have become clear. Further facilities need to be added, especially those geared to research and integration with advanced workstations on campus. This is what has led to HyperCOSTOC, a new CAI architecture based on COSTOC courseware, which includes 'hypermedia' type of properties (as in [7]–[16]), while it remains modular, easy to use and cost-effective.

The main differences between COSTOC and HyperCOSTOC can be summarized as follows:

—improvements of the authoring facilities running on a wide spectrum of hardware;
—improvements of the delivery system (fileserver and database software, usage of lessons) on a variety of hardware: facilities of navigation, annotation (see [17]), cross-process communication and parallel execution of lesson segments; and

—incorporation of the ability for simulation, program inclusion within a lesson, ICAI features of student modeling and diagnosis, as well as innovative question and exam modules within a PT-CAI (presentation-type CAI) framework.

The rest of the paper is structured as follows: section 2 presents an overview of the HyperCOSTOC facilities. In section 3, HyperCOSTOC is viewed from the point of view of instructional facilities available to students, in section 4, HyperCOSTOC is viewed from the point of view of the author, in section 5 HyperCOSTOC is viewed from the point of view of the instructor, and in section 6 from the point of view of the lab operator. Finally, section 7 contains a summary.

2. HyperCOSTOC: A general view

HyperCOSTOC is a CAI system which provides a spectrum of 'hypermedia type' facilities and tools for a lesson (or information) database with a choice for online and offline use. The philosophy of HyperCOSTOC is both hardware and media independent. It is especially geared to a distributed, microcomputer-based university environment which integrates a variety of university resources in an intelligent way and which is accessible in different modes to a network of off and on campus users. While HyperCOSTOC aims towards compact storage and rapid retrieval of textual, numeric and visual data, it is also pragmatic in its approach within a university setting in that it is modular, easy to maintain and cost-effective.

2.1 *A typical HyperCOSTOC lab configuration*

A typical HyperCOSTOC lab consists of a number of student/user workstations which are connected via a LAN (or even WAN) to a database of instructional material. In the *browsing mode*, the workstation remains online permanently. In the *studying mode*, a substantial package of instructional material is 'downloaded' into the user's workstation and executed 'locally'. i.e. without requiring connection to the database of lessons during this phase. It is to be observed, however, that even in the browsing mode, the execution of lessons will often be carried out in the workstation to distribute the workload away from the database server. On the other hand, this may not be possible when workstations without sufficient local processing capabilities are used.

2.2 *Browsing versus studying mode*

The extent of browsing versus studying mode (abbreviated B- and S-mode respectively) depends on the type of configuration used:

—The *B*-mode should be favored when workstations are connected to a database server via a network with no time charges (e.g. via a LAN, or a network with leased lines). The same holds when workstations lack sufficient local processing power.
—On the other hand, the *S*-mode might be favored in the following cases: If intelligent workstations are connected to a database via a network with time charges, the *S*-mode will minimize connect time charges. Also, slow networks (below 9600 baud) tend to favor the *S*-mode and material that is 'largely sequential': waiting times for users can be virtually eliminated by 'preloading in parallel', i.e. while the user is working through other material. Another argument in favour of the *S*-mode is that a database server of a fixed size can clearly handle more workstations that way. Such is the case

with the Austria-wide network handling 9000 users described in Maurer [18] and the CONNEX lab described in Cheng *et al.* [19]. The *S*-mode is not desirable when a large database or knowledge base is continuously required.

2.3 *HyperCOSTOC hardware*

HyperCOSTOC consists of two major components: tools for creating courseware and execution environments for working through lesson material. The courseware creation system is supposed to run on high-power workstations, like Sun or Apollo, to offer the author convenient features at a moderate price.

The HyperCOSTOC execution environment is designed to handle both modes, i.e. browsing and studying. It is also planned to run on a variety of workstations ranging from dedicated-CAI micros called MUPID or home-computers like the Amiga to PC with an ega card, PS/2, Sun, Apollo and MAC2. The most widely used workstation for HyperCOSTOC labs will probably be the PC with an ega card. From the PS/2 upward, efficient process switching in a genuine multi-window environment is possible, as we describe later in the paper.

2.4 *HyperCOSTOC database: A network of hypermols*

The HyperCOSTOC database consists of a large number of (usually small) teaching units which represent some 10 seconds to a few minutes of studying time. We call these units *hyper-molecules*, *hypermols* or *mols* for short; hypermols constitute the building blocks of the HyperCOSTOC database. Hypermols have the following properties: they are the smallest modules that a user can directly access, are largely independent of each other (can be authored and tested independently), and the user can switch from one hypermol to another hypermol at any point. A hypermol encapsulates both data and a function or procedure. We propose to follow some object-oriented programming principles (as in Yankelovich *et al.* [20]) in designing the role of hypermols within HyperCOSTOC. To use CAI terminology, while there is free 'navigation' between hypermols, there is restricted navigation within hypermols (hypermols are not 'atomic' in nature).

We distinguish among three types of basic teaching units/objects or hypermols:

—*Presentation-type hypermols* (*PT-hypermols*): They present textual and graphical information which includes dynamic changes of the screen as caused by user-input and also simple animation sequences.

—*General hypermols* (*G-hypermols*): A *G*-hypermol is a type of teaching unit which can include parts of *PT*-hypermols and also have other functions defined with it. For example, a *G*-hypermol function may define a question–answer dialogue between system and user for the purpose of self-assessment or for suggesting on how to continue, or for examination purposes. Another *G*-hypermol function may define an algorithm which is specifiable within the framework of GLSS [21] used for simulation or experimentation purposes. *G*-hypermols have thus not only a structure associated with them but also a function. Thus, we can group together types of *G*-hypermols according to function or according to structure. In either case, the structure and the function (or mode of operation) are intricately connected just like in the concept of a data structure. (Imagine a *G*-hypermol which is a binary tree, its nodes parts of *PT*-mols, its function the binary search tree operation for locating or comparing a given *PT*-mol. In this case, efficient computational techniques for searching can be applied.)

—*External hypermols* (E-*hypermols*): The heart of the HyperCOSTOC database consists of *PT*- and *G*-hypermols which fits with the GLSS philosophy explained in Maurer & Stubenrauch [21]. To keep HyperCOSTOC open-ended, i.e. to be able to incorporate arbitrary software packages, we permit the incorporation of external hypermols (*E*-hypermols) subject to certain mild requirements. An *E*-hypermol is an arbitrary software package and is seen as 'black box' (i.e. no information on how the *E*-hypermol works inside is required by the HyperCOSTOC-system). Such a software package may be an application program, a driver for digitized pictures, voice, sound or video etc. This flexibility provides the 'hypermedia' quality to the HyperCOSTOC system, as we will see below.

The advantages offered to the HyperCOSTOC database by using *E*-hypermols are obvious:

(1) *E*-hypermols can be accessed with the same mechanisms as other hypermols; that is, we allow for a 'standardization' process of the available media and software to take place. Retrieval procedures for *E*-hypermols can be defined by the user in the same way as for *G*-hypermols.
(2) Parameters can be passed from a *PT*- or *G*-hypermol to an *E*-hypermol in a consistent way, possibly defined by some high-level grammar.
(3) Under user-defined assumptions, a *G*-hypermol can act as 'input filter' (selecting from a set of choices) for a set of *E*-hypermols (as explained below) and then control can be returned from an *E*-hypermol to a *PT*- or *G*-hypermol in a specified way which allows to adjust the route taken through a lesson. As will become clear later on, feature (3) is crucial in making HyperCOSTOC a powerful system for incorporation of 'hands-on' or simulation type of software, thus graduating from a CAI system to an *advanced* CAI system.

2.4.1 *Composition of hypermols and navigation.* Although hypermols can be accessed individually, they are usually tied conceptually together into a *lesson*. Groups of lessons compose a *course*. Thus, a lesson is a *network of hypermols*, with one or more hypermols defined as a starting point. We call starting hypermols 'title' hypermols. *Paths* which lead through hypermols within any given lesson, are routing suggestions, and are not part of the hypermols. They are kept separate to make a database of mols highly reusable. Such hypermol routing mechanisms are called *H-tours*.

An H-tour can be interpreted as a directed graph. Each node in this graph corresponds to a certain hypermol. At the end of a hypermol the student is shown choices of further hypermols. A choice corresponds to an arc in the directed graph and leads to a new node and the attached hypermol. There are standard types of branches, like *sequence* (next—back—dynamically last) and *index* (several choices—back). In addition, a student can use a variety of mechanisms to leave (or temporarily leave) a suggested route. One of the most important mechanisms uses highlighted keywords within hypermols. By simply clicking on a keyword the student is taken to the hypermol containing the definition of the specific word. This escape facility from any given H-tour is called an *H-tangent*.

The bulk of the information of a HyperCOSTOC database is kept in hypermols. All information necessary for navigation is kept in the nodes of H-tours. They allow to build highly individualized routes through the database without duplicating mols. Usually, each hypermol can be reached by several H-tours. Figure 1 shows part of two H-tours.

H-tour number 1 starts at hypermol *A* and continues in a straight sequence with *B*, *C*, and *D*. H-tour number 2 starts at hypermol *A*, too. At mol *B*, however, the student can continue with three different mols: a sequence starting with *K*, *L*, etc., a second one with *C*, *D*, ... and a third one emanating from *R*. To keep the figure simple, only forward links are shown. As indicated in Figure 1 mols *C*, *L*, *R*, and *S* are also contained in some other H-tours.

There are *static* and *dynamic* H-tours. Static tours take the user to a new mol regardless of the current status of execution. Dynamic H-tours depend on the outcome of question–answer dialogues. The tradeoff is clear: simple routing strategies are easier to handle and to understand while multibranching status-dependent strategies are more powerful yet sometimes tend to get the user lost in the database, a common problem in hypermedia systems (see [8], [14] and [22]). A way of improving the situation is to show the user a graphical representation of the H-tour through a lesson and the current position within the H-tour, like in Figure 1.

A typical HyperCOSTOC (Figure 2) lesson has the following network structure. The lesson starts with a title and a table of contents. The lesson (30–60 minutes of execution time) is typically split into 5–9 chapters which the author expects to be sequentially executed. In each chapter, in addition to the 'mainstream' material, there exist options to skip sections (e.g. sections including question-answer dialogues), or to include additional material provided in the database. However, beyond such suggested H-Tours, facilities are provided for users to permit at any point to continue with another hypermol, residing within the same lesson or even somewhere else in the database. In addition to being able to leave a 'bookmark' for being able to return to the point left later, the user

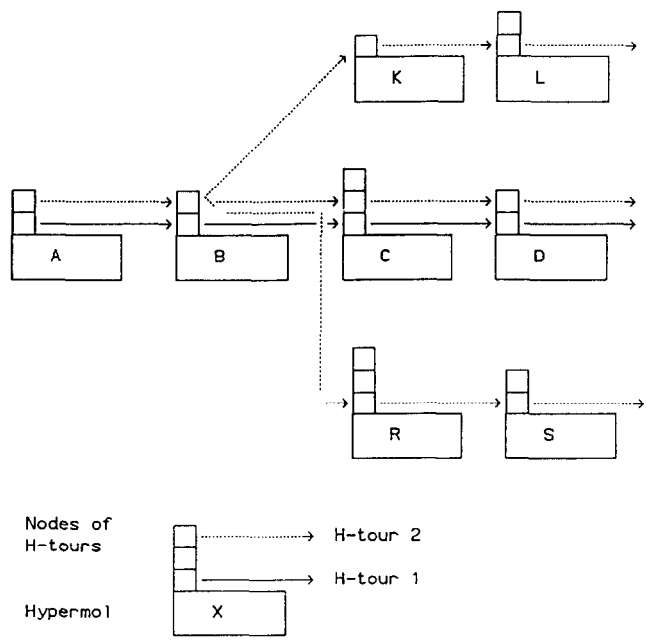

Figure 1. Some hypermols and two H-tours.

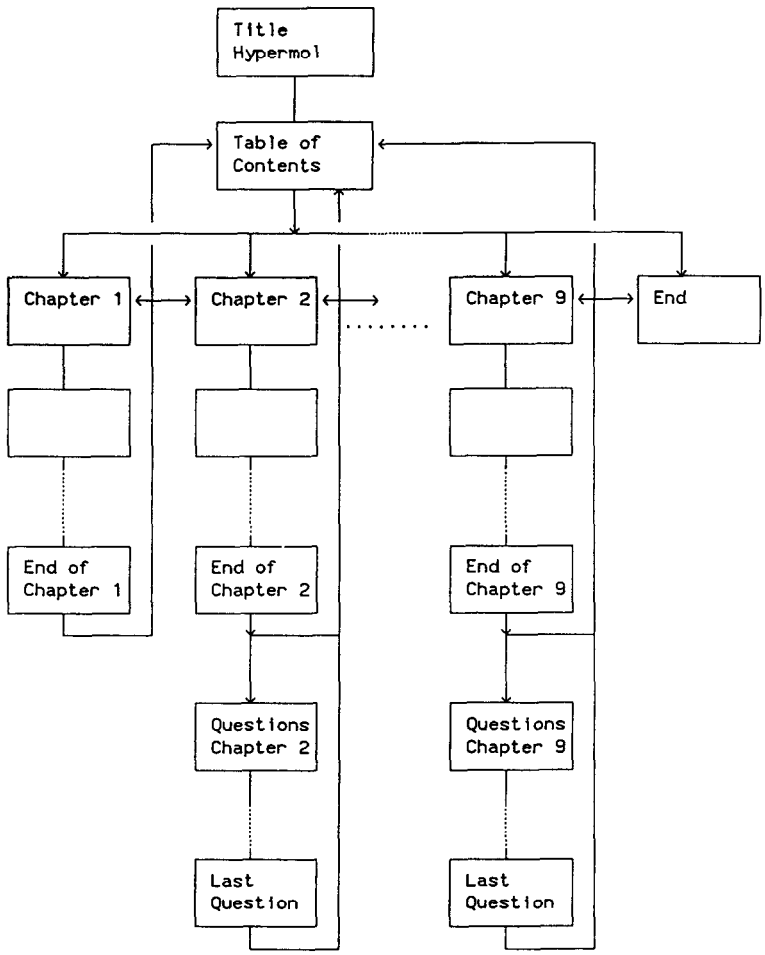

Figure 2. A graphic presentation of a lesson.

also needs other mechanisms for tailoring the lesson execution (whether in *B*-mode or *S*-mode) to specific needs. We define a facility called *active annotation* for the ability to override the lesson H-Tour as recommended by the structure of the lesson. Active annotations are described in more detail in Sections 3 and 5 and in Makedon *et al.* [17].

2.4.2 *Hypermol attributes.* Before we describe the mechanism of active annotation, we need to define the types of information associated with each hypermol. There are two basic types: *identification* information (ID) and *annotation* (AN). The significance of ID information is explained in terms of examples below; AN is explained in section 2.5.

The ID of a hypermol is used to locate the information desired and consists of a number of attributes with associated values. For example, a hypermol of a sorting lesson may have the following set of attributes (with their values indicated in parentheses): molid (sort314), course (sorting), lesson# (3), lesson-name (heapsort), molnumber (14), author (maurer), supervisor (garfield), language (English), domain (computer science),

area (data-structures, algorithms), keywords (heap, heap-definition), attribute-type (PT), etc.

Of the above attributes, only two may need explanation: attribute-type and supervisor. Attribute-type specifies what kind of hypermol we are dealing with: *T* stands for tutorial type, *PR* for program, *Q* for question, *DS* for digitized sound, etc. Attribute-type is not to be confused with the three different types of mols (*PT*, *G*, *E*) in HyperCOSTOC. The attribute supervisor specifies who is responsible for the maintenance of the hypermol; if not specified, the default is the systems operator. Messages (e.g. pointing out errors) can be sent to supervisor (an example is given in Section 4).

2.4.3 *The HyperCOSTOC query mechanism: Some query examples.* The attributes of mols can be used to search for certain mols or lessons. The search will be carried out in interactive fashion, i.e. the user identifies the desired kind of items (mols, lessons, courses . . .) and defines the key for the search. Some examples show how this feature can be used.

In the above example of a hypermol with molid(sort314), a user can find this hypermol by specifying

hypermols (*keyword* = *heap-definition*).

The result of such a query will be a list of hypermols if the database contains a number of mols involving the definition of the data-structure heap. Moreover, by using a wild card character like *, the user can abbreviate names of keywords. Specifying

hypermols ((*lesson-name* = *heapsort*) *and* (*keyword* = **heap***))

will give a list of all hypermols of the lesson heapsort where keyword contains the string heap, such as 'heap', 'definition of a heap', 'heap-definition' or 'heapify'. The names of all lessons on sorting could be determined by specifying

lessons ((*course* = *sorting*) *and* (*lesson-name* = *)).

To obtain a list of all courses on data-structures one would specify

courses (*area* = *data-structures*).

However, specifying

courses (*area* = **data***)

can yield courses on data-structures, databases and data-compression. A user working through a hypermol on the analysis of some sorting algorithm and running into the term 'binomial coefficient' without remembering its definition might mark the current hypermol, use a query

hypermols (*keyword* = *binomial**)

and will find a hypermol defining the binomial coefficient. After going through the definition the student might return to the sorting hypermol. Similarly, by initiating a query of the form

hypermols ((*lesson* = *current*) *and* (*keyword* = **heap***))

hypermols within the current lesson with key words containing 'heap' will be found. As further example, consider a user working through a lesson on syntax-analysis and

requiring the definition of finite automation as defined in a lesson by, say, Salomaa. A query

hypermols ((*author* = *Salomaa*) *and* (*keyword* = *Finite Automat**))

might well lead to the definition wanted. Observe, finally, that

hypermols ((*keyword* = *Salomaa*) *and* (*attribute-type* = *Facsimile*))

could give a facsimile-picture of Salomaa, or

hypermols ((*keyword* = **automat**) *and* (*attribute-type* = *PR*)

a list of computer programs dealing with automata (for experimentation).

A final remark concerning all query examples listed above: since the database contains all the necessary information on keywords, attribute-types etc. the query system can be written in such a way that the user has only to select from menus and to enter text only in some rare cases. Hence the possibility of making an error when constructing a query is reduced to a minimum.

If a keyword of a hypermol is also contained in the text of the hypermol, the keyword is automatically highlighted. So a user is always given hints for potentially successful queries, since most keywords will occur with several hypermols. Furthermore these keywords can be used for queries by just clicking on them.

While the above list of examples is not supposed to be an exact or exhaustive list of how hypermols can be accessed, it conveys the flavor of the HyperCOSTOC query mechanism. In a similar fashion, let us now discuss the annotation facility.

2.5 *The HyperCOSTOC annotation facility*

The annotation facility serves three main purposes:

- —It allows to add notes to each hypermol at different operational levels:
  - —as a systems operator;
  - —as an instructor; and
  - —as a user.
- —It allows to specify new private H-tours which override suggested lesson-embedded tours; hence its name 'active annotation' [17].
- —It provides a communication mechanism between various users of the instructional material (e.g. between students, authors, instructors, publishers, etc.).

In addition to annotations associated with individual hypermols, there is also a messaging facility of the usual kind. For example, students may send messages to their instructor, and the instructor can operate a bulletin board for one of the classes.

We will return to a more detailed discussion of various applications of annotations in Sections 3 and 5.

2.6 *Cross-process communication*

Another important feature of HyperCOSTOC which we call *cross-process communication* is best explained in a multi-window setting by running through a few examples.

Cross-process communication is a facility for a multi-window environment where a number of hypermols can be active on the screen at the same time. To be more precise, in

a multi-tasking environment different hypermols can correspond to different windows and be executed in truly parallel fashion; in a single-task, multi-window environment (sufficient for most CAI applications) a number of windows representing various execution stages of hypermols are visible, but only one of the windows is active. Let us consider only the simpler case here. Activation of a new hypermol (and suspension of the one active before) is either done by the hypermol itself or by user-intervention. As an arbitrary hypermol x is activated from a *PT*- or *G*-hypermol y, a command-file belonging to this activation is executed: this execution puts hypermol x into a certain state. A return from hypermol x to the initiating hypermol y after the point of activation can either be caused by the command file discussed, by the termination of hypermol x or by a command of the user.

We note that, in cross-process communication, *PT*- and *G*-hypermols are quite different from *E*-hypermols. *E*-hypermols cannot activate *PT*- or *G*-hypermols and pass command files; they can terminate or be terminated, returning control to the *PT*- or *G*-hypermole they were activated from. This restriction comes from the fact that, for HyperCOSTOC, *E*-hypermols are (in contrast to *PT*- and *G*-hypermols) 'black boxes' which cannot be manipulated.

Let us now study a number of examples.

Example 1: Parallel execution of hypermols

Suppose we have three *PT*-hypermols, *A*, *B*, *C*. Assume that each of them consists of some textual and graphical information which is broken into three pieces. For instance, *A* is of the form $A = A_1, P, A_2, P, A_3$ indicating that first part A_1 is shown, after a key press (P) part A_2 is shown, and after a further key press A_3. Assume that *B* and *C* are built up analogously. Suppose now we want to show to the user *A*, *B* and *C* in three different windows in four stages as follows: first A_1, B_1, C_1; then A_2, B_2, C_1; then A_3, B_2, C_2; then A_3, B_3, C_3; the stages separated by a key press.

In a truly parallel fashion the definition of these sequences is fairly easy: the author selects all three hypermols and executes them, showing the editor where synchronization points should be established. The editor could insert additional pauses in the nodes of the corresponding H-tour to ensure correct execution (in this way the execution system can proceed within each hypermol simultaneously). To a student the three hypermols now look as if additional pauses have been inserted:

$A_1, P_1, A_2, P_2, A_3, P_3, P_4$
$B_1, P_1, B_2, P_2, P_3, B_3, P_4$
$C_1, P_1, P_2, C_2, P_3, C_3, P_4$

In a sequential environment the mols would have to be changed in the following way:

$A' = A_1$, act(B'), A_2, react(B'), A_3, react(C')
$B' = B_1$, act(C'), B_2, P, react(A'), B_3, react(C')
$C' = C_1$, P, react(A'), C_2, P, react(B'), C_3, P

In above, act(x) means activate hypermol x, react(x) means reactivate it after an earlier suspension, i.e. in the state it has been left. Here A' has to be activated to give the same result as in a parallel environment. Note, however, that this description is only thought of as a model, since the mols themselves cannot be changed, but rather the nodes of hypermols. Changing a mol would also affect all other tours referencing this mol.

Example 2: User-control on hypermols

Different hypermols can also be shown step by step on the screen under direct user-control. Consider once more the three hypermols A, B, C above and suppose the user wants to see simultaneously first A_1, B_3, C_1 and then A_3, B_3, C_3. The user activates A in one window giving A_1, than activates B in another window and presses a key twice to obtain B_3, then activates C in a third window. Now A_1, B_3, C_1 are visible. By reactivating A and B and pressing a key twice, each, the second desired configuration is obtained.

Example 3: E-hypermol activation

Let us consider next a concrete example showing how PT- and G-hypermols can be used to activate an E-hypermol repeatedly. Suppose x is a PT-hypermol that explains how the Euclidean algorithm for determining the largest common factor (lcf) works and suppose y is a G-hypermol that reads two numbers $m, n \geqslant 0$, shows how the lcf algorithm works repeatedly and terminates if m or $n = 0$. Typically, y might display, for e.g. $(m,n) = (48,20)$ the sequence:

lcf(48,20) = lcf(8,20) since mod(48,20) = 8;
lcf(20,8) = lcf(4,8) since mod(20,8) = 4;
lcf(8,4) = lcf(0,4) since mod(8,4) = 0.
Thus, lcf(48,20) = 4.

To tie x and y together it is sufficient to add a menu such as

1 . . . Try out examples
2 . . . Continue without examples

to the corresponding node of the H-tour. This menu is then displayed in a separate window in the window of x. Choice 1 causes an act(y) to be performed, allowing the user to experiment with the lcf algorithm in a separate window as long as wanted. On termination of the lcf algorithm control returns to the hypermol x, as desired. Observe that y is considered as black box, i.e. can be programmed in whatever fashion as long as it runs under the operating system at issue (e.g. MS-DOS, VMS or Unix).

2.7 *Checking of input (filters)*

We now observe that certain modules (such as, checking the validity of inputs) keep recurring in many simulation- or experimentation-programs. To reduce the work of designing such programs, the authoring system should support the generation of G-hypermols doing the input-checking. These G-hypermols can then be used by other G-hypermols, passing the necessary parameters as command file. We will return to the authoring aspect of this matter in Section 4 but want to reconsider the lcf-example by using a suitable G-hypermol.

Assume that a G-hypermol z is available that requests the input of two integers m,n in the range, say, $0 \leqslant m,n \leqslant 30{,}000$. Input (0,0) signals termination of z; invalid inputs have to be corrected by the user; reactivation of z restarts z. When valid inputs m,n have been obtained z activates y with the command-file 'm,n', i.e. y is started and m,n are (so-to-speak) typed in automatically.

To tie, x, y, and z together is now easy: x activates z; z activates y; y need not be concerned with checking the validity of the input, nor with requesting new input; on

termination of y, z is reactivated and either terminates (reactivating x) or activates y once more with a new command-file etc.

Observe that the proper use of standard G-mols can reduce much of the development work of simulation- and experimentation software. Note further that z acts as a *filter* for y in the sense that no invalid data can ever reach y. Such filters have another important use: they can be employed to assure that experimentation takes place in an orderly fashion. This we will see in the next example.

Consider a lesson which is explaining a word-processing system *ABC*. A hypermol x has just explained how a word such as 'large' can be inserted into a sentence such as 'This is a house'. The user is now requested to actually carry out this insertion using the real system *ABC* and assuming (i) some text including 'This is a house' has already been edited before and (ii) the system, is in, say, editing mode. Hypermol x now activates *ABC* with a command-file whose execution assures (i) and (ii). For later reference, call the current situation S. The user can now freely experiment with *ABC*, reactivating hypermol x using a special command at some stage.

Observe that the problem in above approach is the 'free experimentation' of the user: it leads *ABC* to a state unknown to x. Hence a later reactivation of *ABC* by x (to e.g. practise as next step how to delete a word) is only possible by starting *ABC* all over!

To avoid such problems a filter can be used: in the situation S described above, the last entry of the command-file reactives x, initiating a 'filter program' which passes only certain character strings to *ABC* (passing a string s just means reactivating *ABC* with command file: s, react(x)). Note that the users can now be 'supervised' to whatever degree is desirable! But one has to make sure that the user knows where the messages come from—either from the filter or from the program *ABC*. This is just to prevent the student from feeling too sure about understanding *ABC* (the filter will probably supply more guidance than *ABC*!).

In passing we also want to point out that the mechanism necessary to define a valid set of answers, and the algorithms for checking whether a given answer is in the set (as they are used in the answer judging routines of the execution environment) are also useful for the construction of filters. Continuing the above example, suppose we want to pass 'large' as part of a command-file to *ABC*; this should certainly also be done if the user mistyped 'larrge'! Thus, many of the algorithms for determining spelling errors, redundant words, using synonyms etc., have applicability far beyond answer-judging: filters are just one example, general user inputs are others: when a user types

lesson-name (*course* = *Data-structures*),

and the system comes back with no lessons since the course happens to be named Datastructures (without hyphen) then the system is clearly not sufficiently user-friendly!

In the above we have roughly discussed some of the major features of HyperCOSTOC to give a first impression of the system. We will mention further details in the following sections, where we consider HyperCOSTOC from various points of view. However, before doing so we want to comment once more explicitly on the role of HyperCOSTOC, particularly in university environments. We see HyperCOSTOC as a flexible electronic reference database and library, supporting but only partially replacing ordinary class-room teaching. HyperCOSTOC will be a good tool for remedial work, for studying material from a different point of view, for making up for classes missed, for learning certain aspects easier to learn through PT-CAI work than through class-room

lecturing, for experimenting with certain software packages, for learning how to work with various application software, for deciding whether one is ready to take a certain exam or not, etc. HyperCOSTOC is, however, not intended to replace or emulate a human tutor. We feel that the effort to improve a system like HyperCOSTOC in the direction of behaving 'intelligent like a human tutor' by even a few percent is quite prohibitive as a number of attempts have shown. We are hoping that sufficient advances of ICAI will be achieved in the near future, so that more ICAI can be incorporated into the HyperCOSTOC system by means of *E*-hypermols.

3. HyperCOSTOC from a student's point of view

3.1 *Generally available execution features*

A number of general points has already been explained in the last chapter: a HyperCOSTOC session usually starts by identifying oneself and selecting a lesson from the HyperCOSTOC database via menu pages and/or a simple database query mechanism. The identification is necessary for several reasons: to save the status for later execution when temporarily leaving the system, for exams, to manage private annotations, etc.

In the studying mode, as many mols as possible are loaded into the memory of the workstation. To determine the hypermols needed, the information kept in the H-tour emanating from the title hypermol is used. Lessons come with routing suggestions leading from hypermol to hypermol, as defined in the H-tour by the original author. Whenever desired, execution can be continued with arbitrary other hypermols, (potentially in new windows and allowing to just temporarily suspend the execution of the current hypermol), hypermols possibly including question/exam modules, simulation– and experimentation programs, the use of other media, or execution of some other program-package.

3.1.1 *H-tour.* Even in low-price environments, like PC's with ega board, several windows should be available for lesson execution. The title-hypermol opens a window on the screen that can be moved and changed in its size by the student. At the end of the hypermol the next hypermol is shown in the same window. If the student decides to look at some additional information, e.g. by clicking on a highlighted word in a hypermol, a new window will be opened, showing the desired information. The student can now continue in the H-tour emanating from this mol and again adjust the current window in position and size. If the student decides to close the window, control is given back to the previous window. These links to other mols via keywords can at least partially be established automatically: whenever a new lesson is stored in the database, a cross-reference of all words in the lesson and the keywords of the old lessons can be generated automatically, leaving only the decision what explicit linkages should in fact be established to the author of the lesson or the instructor. Implicit linkages via highlighting of keywords can be done by the system automatically. Using different colors for highlighting the student can also see where the branch will end: at a program, at another *PT*-hypermol, at a definition, etc.

Whenever a branch from a hypermol x is taken to a new mol y, the student has two choices: either this branch is taken temporarily, leaving the option to return to mol x, or going to the new hypermol without the option to return. The side effect of the first choice is that the window corresponding to x will stay on the screen. By simply clicking inside

this window, control is given back to this hypermol. In the second case the window of mol x is closed as soon as the branch to y is taken. In terms of statements in a programming language the first way corresponds to a procedure-call, the second way to a goto.

3.1.2 *Marking*. In addition to the navigational features discussed so far, further features available include marking a hypermol, returning to a marked hypermol, backstepping (repeatedly if desired) through the dynamically last hypermols, stepping back or forward according to the routing provided by the author, continuing with the hypermol containing the (dynamically last) (sub)table of contents and switching on/off a fast display mode in which a hypermol is displayed in accelerated fashion.

The above features are fairly easy to implement (with the possible exception of the last) in most environments. However, there is one important navigational feature which is not so easy to handle fully, hence is often not supported, yet is of paramount importance. It is the *undo* feature, undoing the effect of the last key press in *PT*-hypermols and when following H-tours: the difficulty in implementing repeated undos in an environment of dynamically changing graphic information is that the only trivial implementation—keeping a stack of memory maps and system's status—is usually too memory intensive to be useful. More tricky implementations are possible and discussed in Maurer & Reinsperger [23].

3.1.3 *Help facility*. HyperCOSTOC comes with a help facility explaining to the user the options available. The help facilities are sufficiently extensive, so that even the most naive user can handle the system from the beginning without any training. HyperCOSTOC courses usually come with a printed documentation. In many instances it is, however, convenient that users can indicate—as they work through some hypermols—which parts they want to have printed (including various annotations, see below). This raises the whole problem of printing HyperCOSTOC material, a difficult issue to say the least: even the printing of *PT*-hypermols is difficult: how do you print in black-and-white animated color-graphics? The situation gets still more complicated when trying to print out screens obtained from simulation and experimentation programs, or still more complex software packages. How can such screens even be obtained automatically without user input? Indeed they can, if the author of the instructional material has provided at least one set of default inputs wherever user inputs are expected. Let us mention in passing that providing typical default input sets which can be activated by the user instead of typing own input is a very valuable educational tool which should be provided, anyway. In summary, printing routines for HyperCOSTOC material are available anyway for the authoring process. Hence they are also made available to the user, following the famous 'orthogonality principle' of A. van Wijngaarden. Printing routines include the options for automatic printing or printing at points determined by the user, printing of all or parts of the screen, and printing text only or text and graphics. Animation is indicated by printing initial and final positions and the path taken, erasures of substantial parts of the screen cause a new printout to be started when the automatic mode is on.

3.1.4 *Note-making*. In addition to be able to collect a mixture of parts of hypermols and own notes just for printing as explained above, the information can also be retained in an electronic version as *notebook*. Indeed usually all information is first put in a notebook (which can later be inspected and edited), the printing just being one of the options of handling the notebook. We will see later that this notebook facility is also important for both authors of instructional material and for instructors.

3.2 *Annotations*

We come now to a final but exceedingly important concept of the execution environment already mentioned in Section 2: the concept of annotations and active annotations.

3.2.1 *Public* vs. *private annotations.* As the user starts a session and accesses a first hypermol, a list of *public* annotations is shown. Any of these public annotations and further *private* annotations (for which the user must know the appropriate password, e.g. because these annotations were created by the same user in an earlier session) can be *enabled* (and later disabled, if so desired). To enable an annotation means that whenever a hypermol is activated, all enabled annotations are shown in parallel with the hypermol in a separate window. If the hypermol causes dynamic changes on the screen step by step, the annotations can also be broken into steps and synchronized with the hypermol in a way described in more detail in Makedon *et al.* [17].

Annotations mainly consist of text. They can also create special pointers outside the text window (i.e. in one of the graphic windows) for highlighting. Thus, an annotation may read 'Observe how the valve * opens and closes', with the * appearing both in the text and next to the graphic object being explained. How this is done, and how conflicts between various annotations are avoided is explained in Makedon *et al.* [17].

Public annotations are usually written by an expert commenting on or elaborating the work of the original author, or are written by a teacher as individualized information for a particular class ('. . . a further good book might be . . .', '. . . this proof won't be on the exams . . .', etc.), or are written by a user much like personal notes are scribbled in a workbook.

3.2.2 *Enabling annotations.* Note that a hypermol may have annotations from many people, and annotations can be changed independently of the rest of the hypermol. A lesson on sorting may be annotated in a particular year by two teachers X and Y. A student in Y's class is probably most interested in Y's comments and hence may not even enable the annotations of X. It may well be that Y decides to modify some annotations the next year, since new material has surfaced, new books have appeared, etc. Annotations are easy to change. Just running through the lessons once and using a word-processor for whatever changes are desired is all that has to be done.

3.2.3 *Active annotations.* Annotations are a way to allow a certain individualization and customization of instructional material. This possibility is much enhanced by one further aspect of annotations we have not mentioned deeply yet. It is this aspect which causes annotations in HyperCOSTOC to be called *active annotations*, to differentiate them from the static annotation mechanisms found in many document systems.

Annotations may contain routing suggestions of the kind 'To continue press 1' where the solicited action leads to any hypermol selected by the annotator, in particular to hypermols created by other authors. In connection with an automatic annotation activation and return facility, hypermols can be linked together in entirely new ways both by the teacher and the user.

We will return to the teacher aspect again briefly in Section 5, so let us consider here the user aspect only: as a user works through hypermols, often trying hard to find all the hypermols required to be able to understand everything, and filling in material by writing personal notes as annotations, the user can use active annotations to link all hypermols of interest together. In this fashion the user creates a personal view of the database, makes a second perusal of the material much easier and, by enabling the private annotations before starting the printing routines, a very personal documentation

can be obtained: private routings (when enabled) override all other routings in the printing process.

3.2.4 *Privacy issue.* A final word is in order concerning private annotations: to assure privacy and to conserve storage in the database, private annotations are usually kept on the student's directory or even on floppies and not in the main database itself. The user, on returning to material studied before, is then shown available personal annotations either from the private directory or from the floppy synchronously with the appropriate hypermols.

4. HyperCOSTOC from an author's point of view

It is of crucial importance for any CAI undertaking to create and maintain lesson material easily. We have discussed at length (in Maurer & Makedon [5]) that this is one of the main reasons supporting PT-CAI and, more general, CAI databases containing small pieces of instructional material (be it PT-CAI, experimentation or simulation software) which are as independent of each other as possible and hence can hopefully be designed and tested more or less as independent modules.

The hypermol concept supports this philosophy. *PT*-hypermols are usually small and quite 'context independent' modules (branching information is kept separate); the same holds true of *G*-hypermols except that certain *G*-hypermols (dealing e.g. with exam-type situations, simulations or experiments) may already become substantial in size and hence harder to create, debug and maintain. The *E*-hypermols, on the other hand, are supposed to be developed outside the HyperCOSTOC framework. Their design can be made easier, to some extent, by shifting some of the effort to standard *G*-hypermols whose generation is supported by HyperCOSTOC authoring as will be discussed below. For example, it is possible to use standard input-hypermols (which can be created using the input-hypermol generators) to remove the task of designing interfaces for requesting (and validating) student inputs from *E*-hypermols.

Basically, HyperCOSTOC authoring provides a *presentation facility editor* (for creating *PT*-hypermols or presentation segments of *G*-hypermols), a *question–answer dialogue editor*, a *routing editor*, a *standard hypermol editor* and a *program editor* (allowing to create and test program segments written in a language similar to the one specified in Maurer & Stubenrauch [21]). As an important feature, the program editor allows to use presentation-type constructs as developed by the presentation facility editor.

Both the presentation facility editor and the standard hypermol editor are extensible and hence fairly open ended. Whatever tasks the author wants to perform beyond what is possible using above features have to be shifted into *E*-hypermols. Observe in passing that the use of *E*-hypermols can influence the portability of instructional material.

4.1 *The presentation facility editor*

4.1.1 *Basic objects and operations.* Let us now turn briefly towards the various editors. The *presentation facility editor* is built around GKS features: four types of objects (markers, line objects, fill objects, text objects) are available, each of these objects with the usual attributes; in particular, each of the objects can be located at any position on the screen in any of at least 16 (of a palette of at least 256) colours; markers come in

various shapes and sizes; line objects (such as polyline and spline) are defined by a sequence of points and come in various linestyles and brush-widths; fill objects (such as circle or polygon) are defined by a number of points and come in various fillstyles; and text comes in various sizes and fonts.

The author can use a number of different character sets, can define special characters as needed and, in addition to the usual graphic text objects, character-grid oriented characters with additional attributes (flashing, underlining, inverted) are also available.

Objects can be animated, individually or in groups. Animation is performed stepwise either through a sequence of arbitrarily defined points, or through a sequence of equidistant points along a line or arc. In the process of animation, changes of size and rotations can be specified, together with some other options like a standard beep and/or delay at each position drawn (rather than erasing them each time and drawing them at a new position). Rectangular areas of the screen can be erased.

The sequence of displaying objects on the screen can be modified by the repeat function (allowing to redisplay a number of objects over and over until a key is pressed by the user) and by the pause function interrupting the display of objects either indefinitely or for a certain amount of time. Display continues in both cases as soon as a key is pressed (i.e. author defined 'timed pauses' are also interrupted by a keypress). In addition to a few further options e.g. scrolling textwindows and simple audio outputs (standard beep, at least), the above roughly describes the presentational features available in *PT*-hypermols and supported by the COSTOC authoring (i.e. editing) system.

The support offered for editing the above presentational features includes facilities usually found in modern graphics editors including pull-down menus, using a mouse or graphic tablet as pointing device, rubber-banding while drawing objects, a pick function etc. The pick function can be used to identify objects on the screen for editing or grouping them together into a *segment* which can be manipulated as a whole, e.g. can be zoomed, rotated, and stored in or retrieved from a graphic library. The HyperCOSTOC editing system works, of course, within the framework of the HyperCOSTOC database. Hence it is possible to search through the graphic library (consisting of *PT*-hypermols) in an efficient manner. The editing system should allow to view a graphic presentation at various pixel resolutions, a feature important both for detailed work and for developing instructional material for various types of graphic cards.

4.1.2 *Customization of the presentation facility editor.* The HyperCOSTOC editor we propose is also *extensible* and *customizable* (and only some of these aspects are currently under prototype implementation). Let us look at some examples to get a rough idea of what this means. The crucial aspect of extensibility is that new objects can be introduced (including the specification of all interfaces required by the author, such as menus for input prompts and calculation or other procedures where required). Once introduced, such new objects behave exactly like standard objects: After e.g. an object 'perpendicular bisector' has been introduced, whenever the author activates it, a prompt for the input of the endpoints of a line segment appears. After the endpoints have been defined by the author, the line segment and its perpendicular bisector are drawn automatically. A more complex extension would be to introduce as new objects 'and-gates' and 'or-gates' with inputs and outputs in such a fashion that when drawing such gates and their connection lines, the author is always shown automatically by the system how many inputs and outputs are still left to be dealt with.

Consider a final example and an important observation. A useful new object for many applications could be a pie-chart so that authors can just specify size of circle, location, and figures, on the basis of which the corresponding pie-chart is automatically generated. Like basic objects (such as lines, rectangles etc.) even such new objects created by extensions can be used by program-segments written as *G*-hypermols. Thus, if a pie-chart has been defined as extension (and pie-charts are part of the *standard library extensions*), an author developing a simulation *G*-hypermol using the program editor can convert the figures obtained with suitable parameters.

Customizing and extending the editor also involves actions such as renaming or deleting objects (to simplify menus or to make them more appropriate for the task at hand: since the term 'vector' is more appropriate than 'arrow' in mathematical contexts, but not in others) or pre-setting certain parameters such as color combinations, defining the layout of a table-of-contents-hypermol to be used as kind of template, etc. Consult F. Huber *et al.* [24], [25], and Huber [26], for further details and possibilities.

The main point is that before starting to create *PT*-hypermols concerning a certain subject area it may be more effective to first extend and customize the editor to fit the task at hand. This customization may even be done by someone different from the author since customization may require more computer science expertise (e.g. for writing a few procedures) than the author may care for.

4.2 *The question–answer dialogue editor*

Let us now consider the *question–answer-dialogue* editor as proposed for HyperCOSTOC. Question-answer dialogues come in a number of varieties, usually called *simple* and *composite*.

Simple dialogues are for asking single questions either of *multiple choice* or *free-text* form.

Multiple-choice questions (with up to at least 16 choices) are specified much as usual, including the possibility to suggest routings depending on the answers. Free-text questions are more interesting: the author specifies, using certain rules, a set of n model-answers, each such model-answer called a *filter*. The answer x given by the user is matched against the filters one by one by means of a so-called *answer judging algorithm*, until the filter i, which is matched best, is found ($1 \leqslant i \leqslant n$), or else $i = n + 1$. Depending on the value of i and x an action $f(i,x)$ is carried out; in the case of free-text questions, $f(i,x)$ just consists of a feed-back message to the user and possibly some routing suggestion. (The more general formulation above has been chosen since filters and the answer judging algorithm play a role also in other places, as mentioned in this paper elsewhere).

Mechanisms for the definition of filters and the corresponding answer judging algorithms are an interesting topic in itself. In Autool2 (the editor used as 'working horse' for lesson creation in the COSTOC project, see Garratt & Maurer [27]) filters are specified by simple conventions: the author specifies whether spelling errors are to be ignored (in which case a heuristic algorithm for recognizing such errors is invoked), may specify synonyms, intervals for numbers, redundant words, that word-order is irrelevant, illegal words etc. In HyperCOSTOC a slightly more general approach is used, as detailed in a separate paper [28].

4.2.1 *Dialogue-mode.* Composite question–answer dialogues fall into the categories *form-filling*, *drill*, *exercise* and *exam*. If we have form-filling in a piece of text with

various parts missing, the parts have to be typed in. The evaluation (again using filters and answer judging) is performed once at the end, with options allowing to perform corrections or to present an immediate judgement.

For each of the remaining more complex question–answer dialogues a *set of problems* and an answer judging algorithm have to be specified. Each problem consists of a question and a filter (specifying the correct answer). In the simplest case, the set of problems is specified one by one; for each problem, the question consists of some textual or graphical (!) information. In case of graphical information, it is of course created with the editing facilities mentioned earlier. The filter is specified with the standard mechanisms, and the answer-judging algorithm is the standard one.

It is also possible to define the problem-set and the answer judging algorithm entirely differently as we will see below. Let us first, however, see how the three question–answer dialogues work, assuming that the problem-set and the answer judging algorithm have already been defined.

In each case a subset of the set of problems (the size of the subset decided on by the author) is chosen at random.

4.2.2 *Drill, exercise and exam mode.* If we wish a drilling mode, a view option (which can be enabled by the author) can be used to first look at all questions and the corresponding answers before the actual drill starts. Essentially, drilling forces the learner to answer all questions repeatedly, until one can be sure that the answers have been learnt. There exist different strategies for choosing questions from the set of all possible questions, from simple picking at random to more sophisticated methods, like in Merrill & Salisbury [29].

In the exercise mode, questions from the set chosen are just presented once (in a random order) with feedback showing the correct solution if the user has not found it after a number of tries. At the end, the percentage of questions answered (and on which try) is shown to the user for self-assessment.

The exam mode is similar to the exercise mode except that a limited time, as specified by the author, is available and correct answers are shown only after the exam is finished. Also, there is an identification procedure at the beginning, and results are recorded in a file only accessible to the instructor. The instructor can request a graphical presentation of results obtained by a group of users, can define cut-off points for grades and request a printed listing of students and grades. An interface to use the data is available for applications such as automatic printing of report cards.

We want to emphasize that although the exam mode is a great tool for providing some information on the performance of students, it certainly can be used only in conjunction with other criteria to determine the overall achievement level of a student.

4.2.3 *Problem generation.* As we have pointed out, problem sets and answer-judging algorithms can either be defined by standard techniques, or else can be defined using algorithms specifically designed by the author, using the program editor of the HyperCOSTOC authoring environment: here is one of the points (like in the case of extensions to the graphic editor) that an author may be forced to develop some computer program (or get it developed by someone else).

We explain the main idea using two examples.

Suppose a teacher wants to design an exercise for high-school students to practice how to add two fractions. The problem-set and the answer-judging could be defined along the following lines:

```
  Question:
a: = ran (0, 99); b: = ran (1, 99);
c: = ran (0, 99); d: = ran (1, 99);
Sa: = string (a); Sb: = string (b);
Sc: = string (c); Sd: = string (d);
POSE_QUESTION (Sa + '/' + Sb + ' + ' Sc + '/' + Sd + ' = ?' );

  Filter:
GET_ANSWER (x, y);
r: = a * d + b * c;
s: = b * d;
red: = lcf (r,s);
r: = r/red; s: = s/red;
IF r = x and s = y
THEN BEGIN
  CORRECT: = true;
  FEEDBACK: = 'bravo'
  END

ELSE IF r * y = r * s
  THEN BEGIN
    CORRECT: = false;
    FEEDBACK: =  'Not quite correct!
                  Reduce the result'
    END
  ELSE BEGIN
    CORRECT: = false;
    nom: = string (a) + '*' + string(d) + ' + ' + string(b) + '*' + string(c);
    denom: = string(b) + '*' + string(d);
    FEEDBACK: = 'No! Compute first' + nom + '/' + denom +
                'then reduce this fraction'
END
```

In the above we assume that string(x) converts an integer value x to string, that + stands for string concatenation, and that POSE_QUESTION, GET_ANSWER, CORRECT and FEEDBACK are systems routines and variables known to HyperCOSTOC, respectively. Whenever a problem is selected (as often as specified by the author) four positive numbers a, b, c, d below 100 (b and d not equal to 0) are generated at random. Assume e.g. $a = 3$, $b = 4$, $c = 1$, $d = 6$. The string QUESTION, as it will be shown to the student, will therefore be "3/4 + 1/6 = ?". Since $r: = a * d + b*c = 22$ and $s = b*d = 24$, initially, we obtain red = 2, hence $r = 11$, $s = 12$. If the student types in 22 and 24 (or even 33 and 36) the feedback "Not quite correct . . ." is given, while for an input such as 5 and 7 some advice on how to proceed is given. (To keep the example simple further parameters needed by POSE_QUESTION and GET_ANSWER have been omitted).

4.2.4 *Exercise generation in batch-mode.* Let us consider one more, and a more interesting example. Suppose an instructor wants to provide an exercise in differentiating functions of one variable x. The instructor might define a set of problems by listing pairs of the type $(f(x), f'(x))$, where $f(x)$ contains a number of parameters which can be chosen at random. (More complex ways of generating a problem set by implementing a differentiation algorithm are of course equally possible). In the way described, a large problem set can be easily obtained without much programming.

The interesting part is the answer-judging algorithm as it is applied to the function $g(x)$ typed in by the student. First, $g(x)$ is checked whether it is a syntactically valid formula. If so, no attempt is made to prove $f'(x) = g(x)$ (since this is in general even undecidable for real-functions!) but rather some 10 sample points $x_1, x_2, \ldots, x_{10}$ are chosen, and $f'(x_i) = g(x_i)$ is checked. If equality holds in all cases the student has indeed solved the problem with very high probability! Observe that this trick reduces the complexity of the algorithms required dramatically.

4.3 *The routing editor*

The *routing-editor* allows to link hypermols together, suggesting to users one of a selection of hypermols to view next, some of the routes suggested (or even enforced) depending on the outcome of question–answer dialogues.

The database of hypermols stores for each hypermol H a list of all hypermols that can be reached from H plus an inverted list (i.e. a list containing all hypermols that point to H). If a referenced hypermol is deleted, the author is warned. If deletion is carried out, nevertheless, a note is deposited for the supervisor of the corresponding hypermol or to the systems operator, if no such supervisor is specified. Observe that often the supervisor will be an instructor who has pointed to the hypermol now deleted in an annotation. Hence, by just editing this annotation the invalid link will be corrected.

4.4 *The standard hypermol editor*

The HyperCOSTOC authoring system also comprises a *standard hypermol editor* which provides support in producing standard *PT*- or *G*-hypermols or parts of hypermols as required for many simulation and experimentation applications. We discuss this feature by just explaining one example.

In many simulation and experimentation situations the student is required to type in a number of parameters satisfying certain conditions (e.g. three positive integers, one negative integer, a sequence of at most six characters, and a cursor position—in that order). The necessary dialogue and validity checking (the latter potentially handled by the use of filters) can be generated from parameters supplied by the author by a special *input-mol generator* provided as part of the standard hypermol editor. This input-mol generator requests for each of the parameters to be handled information including: screen position where the prompt for the student has to appear; type of parameter (integer, real, string, ...) and restrictions on the parameters (specified as a conditional, such as $(x > 0)$ *and* $(x < 300)$ *and* $(x \bmod 2 = 0)$); feedback for student, if student input is wrong; and default input (and how it is activated, e.g. by the student pressing the return-key only.)

Observe also that things like a *title page generator* or a *table of contents generator* are conceivable; of course this could also be handled by the extension mechanism of the presentation facility editor mentioned earlier: it will depend on the situation which approach is more convenient. In the same way as the presentation facility editor can be extended and, indeed, a library of standard extensions can be built up, the standard hypermol editor can be extended by increasing the number of generators or by building up a library of standard generators.

4.5 *The program editor*

As last part of HyperCOSTOC authoring we come to the *program editor*. Here we find facilities beyond what is otherwise available. To use this editor, some programming knowledge (in a Pascal-like language) is necessary. The program editor allows to edit and test programs as explained in Maurer & Stubenrauch [21] (an updated version of this paper is in preparation). It comes with one important feature: in various procedures requiring the specification of parameters such as coordinates or groups of objects, these parameters can be incorporated by just pointing at drawings created earlier with the presentation facility editor.

Thus, writing programs such as moving a composite object drawn with the presentation facility editor along a path calculated by a small program will indeed be very easy.

4.6 *Other aspects of the authoring environment*

Message, notebook and calculation facilities are available to authors (exactly as for students). The facility to collect information from hypermols, edit it, add personal notes and finally print it is particularly valuable for authors as a means of producing a printed course-documentation readily. Since all COSTOC authors are required to produce a detailed course-documentation for reasons described in Makedon *et al.* [1], this facility is geared to delivering a high-quality printout.

HyperCOSTOC authoring also supports the extraction and replacement of text-pieces from hypermols for translation purposes, full-text searches and string replacements across the range of lessons and other text-related functions. It does not support searches for graphical objects or graphic replacements, an important area of graphics which is still in its infancy. Consult Stögerer [30], for first attempts.

5. HyperCOSTOC from an instructor's point of view

As first step an instructor has to find out whether any material in the HyperCOSTOC database is of interest for the specific situation. This is usually done on the basis of the printed documentation of the courses plus the so-called 'Lesson 0'—each such lesson presenting an overview over a complete course for potential instructors.

Once a course looks promising, the instructor will check through it to determine which parts to use. Often, an instructor will decide to use a few lessons as they are for replacing some class-room teaching and some further just for 'recommended reading'. Usually the instructor will go through the material once, making annotations ('. . . learn up to here for the midterm . . .', 'also look at the book . . .') for the students. The instructor may also decide to customize the material beyond annotations in two ways:

—The instructor can create some own hypermols (e.g. a new title-page including things like office hours, date of exams etc.; or inserting a hypermol to explain differences in notation as used in the textbook vs. as used in the lesson);
—The instructor can suggest new routes by so-called *active annotations* (Makedon *et al.* [17]). Thus, the instructor can side-track to some other material for review purposes, can tie in more advanced topics where desirable, skip parts of a lesson, and so forth. Once these active annotations are in place, the material looks—to the student—like a coherent lesson again.

We feel that customization of lessons through annotations both by instructors and students is a powerful tool to make HyperCOSTOC lessons more widely usable.

An important issue for instructors is often the HyperCOSTOC facility of allowing both exercises and exams to be carried out.

Finally, instructors want to know how often which hypermols have been accessed. For this purpose, HyperCOSTOC presents comprehensive statistical information on the use of database. As a matter of fact, in addition to the 'statistical' package (just presenting tables of what has been used when) a sophisticated 'monitoring' package should also be available that records all keypresses of students (in an anonymous way). Such data, together with special utilities, will provide good insight into how the database is used, which parts of questions cause problems, etc.

6. HyperCOSTOC from a system operator's point of view

Once HyperCOSTOC and the HyperCOSTOC database have been installed, maintenance work as required from the systems operator is quite minimal.

New courses will have to be installed once in a while, the files containing the passwords for the messaging system have to be updated as needed and the file of those annotations which are supposed to be public has to be maintained. Actually, HyperCOSTOC recognises a group of privileged users called instructors who can (a) decide which annotations are supposed to be public and who (b) are entitled to install and delete passwords for the messaging system. This removes a further burden from the systems operator. The only other task remaining for the systems operator is to check for system-related messages, once in a while.

7. Summary

HyperCOSTOC is a major undertaking whose aim is to provide the basis for designing the tools to set up a comprehensive CAI lab to support teaching activities at various levels. In addition, a large database of instructional material is developed in some key areas (e.g. computer science) for extensive pilot usage.

HyperCOSTOC is an incremental undertaking. Although some facilities described may never be implemented by ourselves beyond a prototype stage, COSTOC labs have been in successful pilot-operation since 1985, and in commercial operation since the beginning of 1988. A multi-national company is working on implementing some of our more advanced ideas to upgrade COSTOC more and more towards true HyperCOSTOC.

We believe that most of the features of HyperCOSTOC described will be typical of future electronic libraries. Hence we feel that our ideas should be of interest independent of what percentage will finally be implemented.

Acknowledgements

Support of part of this research by the Austrian Federal Ministry of Science and Research (Grant Zl. 604.508/2-26/86) and the Fonds zur Förderung der wissenschaftlichen Forschung (Grant P6042P) is gratefully acknowledged.

References

1. F. Makedon, H. Maurer & T. Ottomann. 1987. Presentation type CAI in computer science education at university level. *Journal of Microcomputer Applications*, **10,** 283–295.
2. J. Carbonell. 1970. AI in CAI: an artificial-intelligence approach to computer-assisted instruction. *IEEE Transactions on Man–Machine Systems*, MMS-11.
3. H. W. Rickel. 1987. *An Intelligent Tutoring Framework for Task-Oriented Domains. MS Thesis*, Computer Science, University of Texas, Dallas, Richardson, TX, 75083-0688, USA.
4. F. Makedon & H. Maurer. 1987. CLEAR: computer learning resource center. *Proceedings of the IFIP Conference on Teleteaching Budapest*, pp. 93–106. Amsterdam: North Holland Publ. Co.
5. H. Maurer & F. Makedon. 1987. COSTOC: computer supported teaching of computer science. *Proceedings of the IFIP Conference on Teleteaching Budapest*, pp. 107–119. Amsterdam: North Holland Pub. Co.
6. T. Ottmann. 1986. Can teaching by computers replace teaching by Professors? *Report 173*, Institut. für Angew. Informatik und Form. Beschreibungsverfahren, Universität Karlsruhe, Germany.
7. N. Yankelovich, N. Meyrowitz, & A. van Dam. 1985. Reading and writing the electronic book. *Computer*, **18,** October, pp 15–30.
8. L. N. Garrett, K. E. Smith, & N. Meyrowitz. 1986. Intermedia: issues, strategies and tactics in the design of a hypermedia document system. *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 163–174, Austin, TX.
9. N. Yankelovich, G. P. Landow, & D. Cody. 1986. Creating hypermedia materials for English literature students. *Technical Report*. IRIS Brown University, Providence, R.I.
10. J. Conklin. 1987. Hypertext: an introduction and survey. *IEEE Computer*, **20,** September, pp 17–41.
11. R. Trigg, L. Schuman, & F. Halasz. 1986. Supporting collboration in NoteCards. *Proceedings of the Conference on Computer Supported Cooperative Work*, Austin, TX, pp 153–162.
12. R. Trigg & M. Weiser. 1986. Texnet—a network-based approach to text-handling. *ACM Transactions on Office Information Systems*, **4,** 1–23.
13. N. Delisle, & M. Schwartz. 1986. Context—a partitioning concept for hypertext. *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 147–152, Austin, TX.
14. P. Kahn. 1987*a*. Outline for research in large database resources. *Technical Report, IRIS*, Brown University, Providence.
15. P. Kahn. 1987*b*. Isocrates project, final report. *IRIS, Technical Report* **87** (2). Brown University, Providence.
16. D. Osgood. 1987. The difference in higher education. *BYTE*, Feb., pp 165–178.
17. F. Makedon, H. Maurer & L. Reinsperger. 1988. On active annotation in CAI systems. *IIG Report No. 256*, Graz Univ. of Technology, Austria.
18. H. Maurer. 1986. Nation-wide teaching through a network of microcomputers. *IFIP-World Congress*, Dublin, pp. 429–432. Amsterdam: North Holland Publ. Co.
19. H. Cheng, P. Lipp & H. Maurer. 1985. GASC: A low-cost, no-nonsense graphic and software communication system. *Electronic Publications Review*, **5,** 141–155.
20. N. Yankelovich, B. Haan, N. K. Meyrowitz, & S. M. Drucker. 1988. Intermedia: the concept and the construction of a seamless information environment. *Computer*, **21,** January, pp 81–96.
21. H. Maurer & R. Stubenrauch. 1987. GLSS: A general lesson specification system. *IIG Report 241*, Graz Univ. of Technology, Austria.
22. G. Marchionini, in press. Information-seeking strategies of novices using a full-text electronic encyclopedia. *Journal of the American Society for Information Science*.
23. H. Maurer & L. Reinsperger, in prep. Complex execution features of CAI systems and their execution with limited resources. *IIG Report*, Graz Univ. of Technology, Austria.
24. F. Huber & H. Maurer. 1987*a*. On editors for presentation type CAI. *Applied Informatics*, **29,** 449–457.
25. F. Huber & H. Maurer. 1987*b*. Extended ideas on editors for presentation type CAI. *IIG Report 240*, Graz Univcrsity of Technology, Austria.

26. F. Huber. 1988. On customizing a PT-CAI Editor. *IIG Report 249*, Graz University of Technology, Austria.
27. J. Garratt, & H. Maurer 1987. Autool2 manual for COSTOC authors. *IIG Report 244*, Graz University of Technology, Austria.
28. H. Maurer & R. Stubenrauch, in prep. Filters for CAI. *IIG Report*, Graz Univ. of Technology, Austria.
29. P. F. Merrill & D. Salisbury. 1984. Research on drill and practice strategies. *Journal of Computer-Based Instruction*, **11**, Winter, pp 19–21.
30. J. Stögerer, in prep. Searching and replacing in picture-bases, *IIG Report*, Graz University of Technology, Austria.

Dr Fillia Makedon is a Research Assistant at Penn State University, Chicago and North Western University, Evanston where she received an M.Sc. and Ph.D. in computer science. She was Assistant Professor at the latter university from 1983–1985 and also at the University of Texas since 1985. She is Research Consultant at the Computer Technology Institute, Patras, Greece and Director of the CLEAR Centre at UTD. Her main research interests are in VLSI theory, design automation, AI techniques for VLSI design, graph layout problems, VLSI implementation of interconnection networks, parallel computing, CAI and computer science education.

Friedrich Huber studied Computer Science and Data Processing at the Graz University of Technology where he received his M.Sc. in 1984 and his Ph.D. (in the area of computer based teaching) in 1989. He is the head of the group for computer assisted instruction at the Institute for Foundations of Information Processing and Computer based New Media.

H. Maurer studied mathematics at the Universities of Vienna and Calgary. He received his Ph.D. in Mathematics from the University of Vienna 1965. He has held the following positions: Assistant and Associate Professor for Computer Science at the University of Calgary 1966–1971; Full Professor for Applied Computer Science at the University of Karlsruhe, West Germany, 1971–1977; and Full Professor of the Institute for Information Processing of the Graz University of Technology 1978–1986; Director of the Research Institute for Applied Information Processing of the Austrian Computer Society since 1983; Adjunct Professor at Denver University 1984–1988; and Chairman of the Institute for Foundations of Computer Science and Computer Supported New Media since 1988.

He is the author of four books and some 200 scientific contributions and Chairman of Working Group Videotex in Austria. He is also the Project Manager of a number of multimillion-dollar undertakings including the development of the colour-graphic micro MUPID and the distributed CAI-system HyperCOSTOC.

His main research and project areas are: languages and their applications, data structures and their efficient use, telematic services, computer networks, computer assisted instruction, new media, social implications of computers.