MEDINFO 80, Lindberg/Kaihara, Editors
© IFIP, North-Holland Publishing Company (1980)

A CUSTOMIZED PROGRAMMING ENVIRONMENT FOR PATIENT MANAGEMENT SIMULATIONS

Johan ELFSTRÖM (1), Jan GILLQUIST (1), Hans HOLMGREN (2)
Sture HÄGGLUND (2), Olle ROSIN (3) and Ove WIGERTZ (3)

1) Department of Surgery, Linköping University, S-581 85 Linköping, Sweden
2) Informatics Laboratory, Linköping University, S-581 83 Linköping, Sweden
3) Dept. of Medical Informatics, Linköping University, S-581 85 Linköping, Sweden

Patient management simulations have emerged as a technique to train medical students in clinical decision making. We present a prototype system, MEDICS, which has been implemented as a customized programming environment allowing the teaching physician to interactively develop such simulation programs. Simulated cases are stored as structured data in a database and development of a new simulation problem is viewed as editing operations upon the description of the simulated patient. Execution of a simulation is monitored by extendible interpreters and procedural code may also be attached to the data structures for exceptional cases. The system supports abstract domain specific concepts and notations in addition to general programming facilities of a conventional type. Thereby the system does not necessarily demand computer proficiency from its users, while still retaining powerfulness and flexibility.

1. INTRODUCTION.

One important aspect of medical work is decision making, e.g. regarding which investigations are relevant and justified, considering their cost and possible harm, when encountering a new patient. This kind of decision making is hard to train in the education of medical students, where it could only to a limited extent be accomplished by confrontation with real clinical cases.

Simulation of patient management was pioneered by Rimoldi <9> as a method to train medical students in clinical decision making through simulations. This technique with patient management problems (PMPs) was later developed by e.g. McGuire et al. <7> using text booklets containing a series of sections dealing with different aspects of the management process. Within each such section a number of options are available for selection, by using a special pen to develop the latent images of the associated responses.

In Sweden, experiences of the McGuire method of paper-and-pencil simulations have been gained in Linköping since 1976. A research project, investigating the prospects of computerized support for PMP development and execution, is presently performed as a joint effort between the departments of Surgery, Medical Informatics and the Informatics Laboratory at Linköping University. The most salient feature of this endeavour is the pursuit of a habitable interactive environment for the teaching physician, where PMPs are easily constructed, tested, refined, documented and subsequently managed.

Computerized versions of PMPs have been used e.g. at University of Illinois <8>, University of Wisconsin <2> and University of Alberta <11>. Another related approach is the computerized simulation of the clinical encounter, as developed by Harless et.al <4>. To develop simple programs of this kind is a fairly well recognized task from the programming point of view. However it is a common experience from this and other similar applications of computer aided instruction (CAI), that the amount of routine work required for each single program, is often too large to make computer programs a viable educational tool. Thus it seems important that a main investment is made in the development of software supporting authors of educational programs. We feel that previous efforts in this direction have primarily emphasized language design and not enough observed the benefits of interactive programming environments, i.e. systems with interactive tools for program development.

Our approach to the task of computerizing PMPs is to design a highly specialized programming environment supporting a suitably chosen PMP model framework, thus customizing the system to the needs of the physician who is creating the PMPs. Simulation problems are then represented by problem oriented data structures, which are interpreted at run-time. These data structures implements a model of the patient management task, as defined by the general structure of PMPs. Evolution of the simulation model is reflected by modifications and extensions to the structure definitions. Creating a new problem means declaring what kind of management is to be simulated, how control is to be executed and then filling in the data structure with appropriate instantiated data.

This approach of a PMP editor rather than a programming language for the author of a problem is

essential, since it means that the need for education of the author himself is substantially reduced. If the effect of a declaration or modification is immediately displayed or at least inspectable, it is not so important to know the details of the semantics of a statement in the PMP definition language, especially if restoring back-ups are supported as in our system. We have also devoted a considerable effort to design facilities adaptable to various categories of users, ranging from support of different terminals (e.g typewriters or CRTs), mixed initiative dialogue (controlled by the user, not the program) and functional redundancy, i.e. the availability of many means to the same end.

In fact the task of PMP construction was chosen as a test case to accumulate experiences of methods and program tools for implementation of customized programming environments for specific classes of applications <6>. We believe that our approach represents a significant improvement, when compared with the mainly program guided scripts for entering simulated patients, that have been implemented as an aid for authors in some related systems, see e.g. <2, 5>.

2. SIMULATION OF PATIENT MANAGEMENT.

Patient management, as being trained by our computer simulations, consists of a number of distinct activities to be performed in appropriate order and in a way depending on the clinical condition of the patient. Such activities involves history taking, physical examination, laboratory tests, medical treatments etc. The actions of the student are assessed primarily depending on their relevance against the background of accumulated information. Suggestion of the proper diagnosis and treatment prescriptions are but components in this assessment.

The physician who is designing these patient management simulation problems has an explicit model for the management task. This model consists of a section for each type of activity that may be performed. Within each section a specific type of information about the patient may by gathered or some treatment measures prescribed.

Each section thus typically defines a context where a number of alternative actions are allowed, for instance one or more X-ray examinations may be ordered. In the paper-and-pencil version these options are explicitly listed, and the corresponding responses are developed (from latent images) when an alternative is selected.

Some options defined in the context of a management section are not pure information gathering or simple treatment ordinations, but rather decisions to proceed to another section for continued work. These items are usually listed separately to be considered when the activity within the section is completed. Such branching to a new section may also be forced by explicit instructions in the response associated with an item. Figure 1 shows a simplified example of such a section in a (paper-and-pencil) PMP.

Section D. Physical examination.

Choose as many items as you think are indicated.

1. Abdomen.
2. Blood pressure.
3. General appearance.
4. . . .
   . . .

After collecting this information, select from the list below what further action you wish to take.

Choose only one!

1. Take detailed history.
2. Order chemical investigations.
3. Order X-ray examination.
4. Give emergency treatment.
5. Admit the patient to the ward.

Figure 1. Example of a PMP section.

2.1 Computerizing simulations.

Given this kind of simulation task, what would we expect a good computerized support system to look like? It is obvious that the paper-and-pencil version of a PMP is programmable in a rather straight-forward way, especially if a CAI language such as e.g. TUTOR <3> is used. However writing a conventional program does not only, usually, mean tedious repetitions of similar code sequences, but also demands an intermediary programmer or at least a physician with a certain degree of computer proficiency.

It would be better if we could use a specification language, where details of run-time control are left out as far as possible. Still such a language is a formal "programming" language that has to be learned and correctly mastered. Making such a language small enough to be easily comprehended means either that flexibility or power of expression is lost.

Our solution to this problem is to implement an interactive programming environment specially designed for development of PMP simulation programs. In order to support a physician, who is prepared to use an on-line terminal, it is essential that the concepts and structures used in the system be as close as possible to the mental model of the PMP designer. Another important aspect is the dialogue facility used for interactions between the physician and the system. Basically there is a choice between some kind of command language and a guided dialogue, where the user is prompted for input. In the former case the user controls the actions of the system by issuing various commands. In the latter case a strict script for the dialogue is programmed where the user only has to supply missing information when prompted by the system. In practice a certain mix of these extremes is preferable, depending on the characteristics of the application situation.

To create computerized patient management simulation problems we thus need:

1. a general model framework for the patient management task.

2. a formal specification language, implemented e.g. as an interactive command language and a set of supporting utility procedures, adapted to this model framework.

3. a data base facility where the description of a simulated patient and the corresponding management task is stored.

4. a script for the dialogue between the student and the program, complete with screen layout descriptions etc.

MEDICS (Medical EDucation with Interactive Computerized Simulations) is a prototype software system, specialized for the PMP application. In the next section we will describe the main features of PMP representation and construction tools, as realized in the MEDICS system.

3. PROGRAM ORGANIZATION.

3.1 The structure of a PMP "program".

A rewarding strategy to gain insight into the nature of the program organization problem is to use design iteration. In our case this meant that simulation problems were first implemented using conventional programming techniques. The resulting programs were then analyzed and experiences from their usage were collected. Building upon this material a generalized design was formulated as a foundation for a second generation implementation.

In the first generation design, each management section had its own procedural code. It turned out however that this code could be classified according to basically two independent aspects. We prefer to name these aspects the execution type and the management type of the section respectively.

These concepts are cornerstones in our current approach. The management type is used as a declaration of what kind of activity is modelled in the section. Thereby application dependency is introduced as an indication of which default structures and data should be chosen if not explicitly stated by the user. For instance, if the management type is declared as "laboratory tests", a section with standard screen layouts and laboratory test data for a normal patient is generated. Of course, the contents of the section may be completely changed subsequently using editing commands.

While the management type labels the semantics of data stored within a PMP section, it still remains to declare how the actual execution of the simulation, using this data, is to be performed. Thus the execution type defines how control is executed at run-time. Presently, execution type is primarily used to define whether management options should be selected from explicitly displayed menus or suggested in a free text format, whether a section is to be executed as a tree structure of subsections, e.g. depending on space limits of the screen when menus are used etc.

Ideally the dialogue mode may be changed simply by assigning a new execution type to a given section, although in practice some extra data or structures are often needed. The concept of execution type facilitates introduction of more elaborate simulation models with a minimal demand for reprogramming of existing cases. It is also very useful for experimental studies, e.g. in order to estimate the influence of technical matters upon the performance of the students, by letting different student groups run the same PMP with different versions of the dialogue.

We have tried as far as possible to store the specification of PMPs as structured data in a PMP database. Procedural definitions of how control is to be executed are then reduced to small program modules, typically a few pages of code, acting as standard interpreters of the data structures. Extending the simulation model may be done by a) rewriting an interpreter, b) writing a new interpreter and assigning a new execution type, or c) attaching executable program code instead of constant text responses to items available within the section. We expect alternatives b and c to be the standard methods in continued use of the system, while alternative a has been used extensively in the iterations during development of the MEDICS prototype.

This program organization method provides a basis both for evolution of the simulation model and for successive implementation of various utilities, e.g. for editing, testing and documenting the PMPs. Before we proceed to a short description of this programming environment, we will conclude with a few examples elucidating the extensibility gained by separating control and the simulation model.

At a rather late stage the idea was introduced that it would be interesting to interrogate the students regarding their current diagnostic hypotheses during the simulation. An interpreter submodule, implementing the desired interaction with the student, was easily written and connected to the existing standard interpreters. Then any PMP previously implemented, could be modified to perform this dialogue variant by a simple update of an attribute for each section where an hypothesis should be articulated. The same strategy was used, when the menu selection technique originally used was to be replaced by prompting without clues, e.g. in the laboratory test section.

Another example concerns a feedback system, that was implemented in order to help the students to evaluate their performance. One component of this system is the option to reenter management sections afterwards. Then special variants of the section interpreters are used, which display the options and their associated score, with an asterisk marking those items previously chosen by the student.

3.2 The programming environment.

From the PMP designer's point of view, MEDICS appears as an interactive environment, where simulation problems are developed, tested, refined and executed. The different services of the system are invoked from the top level on a command issuing basis, but the user may at any time choose to be guided by the system instead, which means that e.g. valid options are displayed and explained.

The following classes of facilities are available at the top level of the MEDICS system:

1. Invocation of subsystems for creation of new simulation problems, data entry of elementary medical items, rearranging of problem structure, test execution of problems etc.

2. Editing of contents and layouts for sections in a simulation problem. This is usually done directly during test execution and the effects of editing commands are immediately visible to the problem designer.

3. Utility programs, e.g. for production of formatted listings of the database, consistency control, scoring support, print-out of booklet versions of computerized PMPs etc.

4. General declarations adapting the system to the needs and skills of the present user, such as terminal characteristics, degree of dialogue verbosity etc.

5. Dialogue control commands for reviewing previous interactions, back-up with updates nullified, resuming after back-up, screen refreshing etc.

We feel that the "undo" facility supplied by the possibility to back-up in the dialogue, having the PMP database restored to its previous appearance, is of utmost importance for a casual user of the system, since a trial-and-error approach is encouraged with limited risk of disastrous mistakes. Unexperienced users are further supported by the availability of guided prompting as a supplement to parameterized commands.

3.3 Entering new PMPs.

There are basically two modes of entering new simulation problems into the MEDICS system. One way is to start with a sequential input of the medical items relevant for the patient at hand. Then follows screen layout decisions and procedural additions for specific parts of a problem if they can't be accommodated within the standard conceptual framework. Final adjustments and refinements are done during test execution of the problem. Notice that these subtasks corresponds to demand for typing skills, computer programming expertise and medical insights respectively. Thus a division of labour may be made, although our experience is that the responsible physician very well can perform all three steps himself.

As an alternative, we have implemented a more elaborate system, designed to support the physician who is prepared to use the system directly as an aid for the PMP construction process. Here parametric specifications for the simulation problem are entered initially, followed by editing of a generated standard problem. In fact we feel that this top-down specification method is a very advantageous way to create new PMPs. Notice that the quality of a simulation as described here is increased the more management alternatives and overall information are available. Since however most of the information is irrelevant and seldom asked for, standard or randomly generated values may be used for many items with little loss in overall quality.

The initial specification of problem characteristics may be more or less extensive. Presently a relatively detailed, basically system guided, dialogue is used where overall structure, management and execution types for sections, standard or specific medical items etc. are chosen. It would also be possible to use the principle of "editing the healthy patient", which means that starting from a simulation problem with perfectly normal clinical conditions, personal characteristics and abnormal conditions are successively introduced until the desired case is reached.

Whichever mode is chosen for construction of simulation problems, it is essential that after typically a few hours work at the terminal, a new problem may be run and tested. Notice also that each section is executable as soon as it has been declared and some items defined. We believe that it is most important for a good final result that the first version of the resulting program is runnable as early as possible and that subsequent additions and refinements are done with powerful computer support during test run of the system.

3.4 Implementation details.

The MEDICS system has been implemented in Lisp 1.6 on DEC System-10, using a set of program packages supporting dialogue management, file management, structure editing etc. <6>. Lisp is an interpretative language, with some features which makes it an excellent vehicle for experimental implementation of special-purpose language environments. Program-data equivalence facilitates a program organization with procedural code and parametric data structures intermixed as dictated by the convenience of the user. The availability of list structuring primitives and uniquely named property list carriers, atoms, makes implementation of dynamically extendible data structures easy. The price to be paid for this flexibility mainly concerns space consumption rather than execution time, since I/O activities typically dominate time performance in the kind of highly interactive applications treated here. For a general introduction to Lisp principles, see <10>.

The Lisp system contains a core-resident database for management of structured data during execution. Unfortunately there is no shared external database facility available for Lisp and we use the ordinary file management system for permanent saving of PMP sections in files. These files are loaded into the address space of Lisp when referenced.

In the prototype system no real effort has been made to optimize the finished PMPs. However we use a special environment for students´ executions. Interpreters and other programs are transferred to this environment from the MEDICS system, if needed. One reason for this division is to reduce the core space needed, which is the critical resource in our system. Another reason is the desire to have a pilot study of the possibilities to translate or rewrite the simulation monitors to another run-time languge environment, having a suitable representation of data for a specific PMP generated from MEDICS.

4. SUMMARY AND EXPERIENCES.

A central concept in our approach is the representation of a PMP simulation problem as a data structure rather than as a program text. This means that programming is in some sense reduced to a database editing problem. By supporting domain specific concepts, such as e.g. medical item, decision item, treatment section etc., and giving immediate feedback of editing operations as far as possible, the MEDICS system can be used with only a minor introduction to computer specific aspects of PMP construction.

In the choice between a concentration upon author support or implementation of elaborate simulation models, we have assigned a high priority to the former task. However a well-defined strategy for extension of the simulation capabilities is part of our approach, due to the separation of the description of the patient and the management task on one hand and the procedural interpreters executing a simulation on the other.

About twenty PMPs have been constructed and made available to the students during the successive development of the MEDICS system. This has resulted in a prolific interaction between practical experiences and system design decisions, although a considerable cost in terms of resources for backward compatibility has to be paid when this approach to system development is used. Some preliminary experiences of using the system are reported in <1>.

As an indication of the achieved degree of acceptance of MEDICS as a construction aid for PMP development, it may be mentioned that the system has also been regularly used to produce the text booklets used in surgery examination at our university. Then PMPs are first entered by the responsible physician as ordinary computerized simulations and edited if necessary. Finally a special utility program is used to print the stored information in the format used for paper-and-pencil simulations.

ACKNOWLEDGEMENT.

This work was supported by the Swedish Board for Technical Development under contracts 77-7535, 78-5269, 79-4958.

REFERENCES:

1. **Elfström, J., Gillquist, J., Holmgren, H. and Hägglund, S.,** Experience with a System for Training Medical Students in Patient Management. *Proc. of the 3rd int. conference of EARDHE,* Klagenfurt (1979).

2. **Friedman, R.B., Korst, D.R., Schultz, J.V., Beatty, E. and Entine, S.,** Experience with the Simulated Patient-Physician Encounter. *J. Med. Educ., 53,* pp 825 - 830, (1978).

3. **Ghesquiere, J., Davis, C. and Thompson, C.,** *Introduction to TUTOR.* Computer-based Education Research Laboratory, University of Illinois (1975).

4. **Harless, W.G., Drennon, G.G., Marxer, J.J., Root, J.A., Wilson, L.L. and Miller, G.E.,** CASE - A Natural Language Computer Model. *Computers in Biology and Medicine, 3,* pp 227 - 246 (1973).

5. **Harless, W.G., Drennon, G.G., Marxer, J.J., Root, J.A., Wilson, L.L. and Miller, G.E.,** GENESYS - A Generating System for the CASE Natural Language model. *Computers in Biology and Medicine, 3,* pp 247 - 268 (1973).

6. **Hägglund, S.,** An Application of Lisp as an Implementation Language for the Domain Expert's Programming Environment. *Report LiTH-MAT-R-79-39,* Informatics Laboratory, Linköping University, Sweden (1979).

7. **McGuire, Ch., Solomon, L.M. and Bashook, P.G.,** *Handbook of written Simulations.* Center for Educational Development, University of Chicago, Illinois (1972).

8. **Nelson, C.D., Sajid, A.W. and Solomon, L.W.,** Diagnose: A Medical Computer Game Utilizing Deductive Reasoning. *Med. Educ., 10,* pp 55 - 56, (1979).

9. **Rimoldi, H.J.A.,** The test of Diagnostic Skills. *J. Med. Educ., 36,* pp 73 - 79, (1961).

10. **Sandewall, E.,** Programming in an Interactive Environment: the Lisp Experience. *ACM Comp. Surveys, vol. 10,* no 1, pp 35 -71 (1978).

11. **Taylor, W.C., Grace, M., Taylor, T.R., Fincham, S.M. and Skakun, E.N.** The Use of Computerized Patient Management Problems in a Certifying Examination. *J. Med. Educ, 51,* pp 179 - 182, (1976)