

BUILDING THE INTERFACE IN SHOOP, AN OOP BASED SHELL

E. Fischetti and A. Gisolfi

Dipartimento di Informatica ed Applicazioni

University of Salerno, Italy

Tfax 39.(0)89/822272 - Tel.39.(0)89/822212

E.mail gisolfi @ udsab.dia.unisa.it

ABSTRACT

SHOOP (Shell in Object Oriented Programming) is an authoring facility for Intelligent Tutoring Systems whose construction is currently under way. Nowadays there is a tendency towards standard architecture for ITS, in particular "shells" which can operate across a set of domains as opposed to handcrafted individual systems. Moreover, OOP is one of the most popular among the emerging software technologies since it allows modeling systems in terms that match the human thinking and language. SHOOP is implemented in Smalltalk/V and this article presents the considerations that were involved in the planning and construction of the system. Then the attention is focused on the interface module (whose duty is to process the information flow from the user to the system and viceversa) whose architecture is described in some details and finally the features of a prototype devoted to elementary geometry learning are described.

INTRODUCTION

Originated during the seventies, Intelligent Tutoring Systems (ITS) result from superimposing AI techniques to classical teaching methods: ITS should be provided with reasoning and problem solving capabilities on the application domain; moreover ITS should maintain knowledge on the student profile so that the system can be more sensitive to the student's behavior. Finally, a friendly user interface with natural language dialogue capabilities should be developed. The ideal goal of a system capable of entirely autonomous tutoring is still prospective and whether it can be reached at all, or to what extent, or how soon are still matters of investigation [3]. The basic difference between CAI (Computed Aided Instruction) systems and ITS is the shift from the programming of *decisions* to the programming of *knowledge*: this is a *methodological* shift and thus we need a deep understanding of tutoring processes in order to build suitable *models*. Moreover, an educational situation involves a *teacher* and a *student*, the object of teaching is *knowledge* in some domain and this should be *presented* to the student in the most suitable way. The basic structure of an ITS can be depicted as follows (Fig. 1):

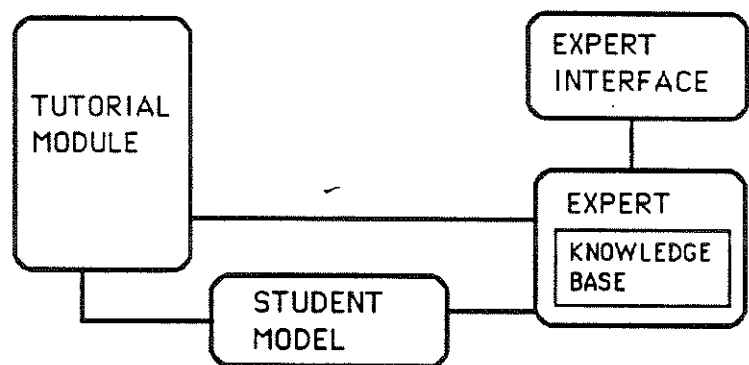


Fig. 1 - Architecture of an ITS

We have four distinct modules:

- 1) an *expert module*, that has knowledge about the topic to be taught
- 2) a *student model*, that suits the teaching strategies to the need of the student
- 3) a *teaching expert*, that explicates appropriate tutoring strategies
- 4) an *interface*, that deals with the presentation of the knowledge to the student.

In order to test the system, the other modules of SHOOP have been just sketched and their structure is, for the moment, oversimplified. Moreover, we have chosen some topics of elementary Geometry as knowledge domain. Realizations of ITS in Geometry are not numerous (see, for example, [2], [7]). The most well known ITS in Geometry is Anderson's tutor [1]: as ours, this system allows a flexible proof and coaches the student with on-line help. However, our planning is concerned with a less directive tutor and moreover we use Smalltalk/V instead of LISP.

THE INTERFACE TO SHOOP

The role played by the interface to an ITS is very important: in fact the Interface can make the presentation of a topic more or less understandable and this can deeply affect the student's acceptance of the system. Moreover, advancements in media technology furnish sophisticated systems that can even drive the design of the system itself. For example, it is worth noting that computer graphics, although the design of graphic interfaces presents relevant problems of its own, increasingly supplements conventional text processing. Furthermore, as an ITS shell aims at achieving true domain independence, its interface should permit people, expert in the domain to be taught but lacking programming expertise, to easily construct new knowledge domains. The peculiar features of Smalltalk/V (e.g. polymorphism, data abstraction and inheritance) were fully exploited: in fact, we have developed a hierarchy of classes, where the top level class implements the inference engine, i.e. the set of methods that allow to select and activate the rules in the appropriate sequence. The other classes refer to collections of rules sharing some characteristics (e.g. some facts and conclusions). Moreover, we have the class "metaknowledge" which interfaces the lower level classes that represent the objects. This methodological approach permits to avoid sequential inspection of the rules: in fact, the rules which can be chosen are only those present in the class activated by the inference engine. Furthermore, the rules can be addresses according to their contents and not to their names.

THE ARCHITECTURE OF SHOOP

First of all, let us consider the two basic objects which our interface is built with. The Smalltalk/V environment is characterized by *windows* and *menus*. Thus we decided to develop the interface by associating suitable screen windows with menus. In such way the use of the keyboard is greatly reduced since it is required only for furnishing the various steps a geometrical proof consists of.

We note that most of the prototype ITS do a lot less than the topic tackled suggests and are only experimental. Moreover, the educational approach is often very simplistic and only few systems have been thoroughly tested. Last, but not least, cost-effectiveness is a crucial problem and we should develop adequate methodologies and tools that can be applied in order that these systems be truly operational and maintainable. In particular, we have seen a number of developments over the last few years (e.g. a tendency towards authoring facilities and standard architectures (shells) as opposed to handcrafted individual ITS), new software paradigms emerged (e.g. Object Oriented Programming (OOP)), and computing power has become sufficiently cheap and powerful so that many systems can run on machines (PC's) that can be afforded by the schools.

An ITS *shell* is a tutoring system which can operate across a set of different domains. The two basic uses of shells are *building*, where the expert, possibly the students themselves, constructs a new knowledge base, and *consultation*, i.e. having students consult a pre-built knowledge base. A truly operational shell should allow getting a new ITS simply by changing the knowledge base, yet despite the push for shells to be used in tutoring activities, so far there has been little take-up of these systems, due to widespread dissatisfaction with existing shells. In fact, many shells require an expert AI programmer to create the knowledge representation for each domain. Albeit the many difficulties with generalizing across knowledge domains, there is a strong need to get cost-effective systems and thus the trend towards "shellification" is impressive [4].

In turn, OOP is nowadays one of the most popular among the emerging software technologies. The language Smalltalk/V (running on IBM PC's and compatibles) uses the OOP paradigm so that systems can be modeled in terms that match human thinking and language, i.e. in terms of objects and actions on objects. The main reason for employing the OOP in ITS research is its adequacy for diagnosing where the issues manifest the conceptual approach. In a computer-based educational context each concept or misconception of the student is implemented as an object, objects are in turn organized in a hierarchy of asses that enables the inheritance of the attributes needed for the diagnosis. Moreover, the programming environment of Smalltalk/V provides easy visual access to each object [5].

In this paper we present SHOOP (Shell in Object Oriented Programming), a shell whose implementation is currently under way and employes the OOP paradigm, using Smalltalk/V. In particular our attention is focused on the first available module, the Interface, whose duty is to process the information flow from the user to the system and viceversa.

Defining a window in Smalltalk/V implies to consider an instance of the class TopPane and then add some "panes" to it. A *pane* is defined as an instance of the class Pane (usually we have used text panes). On the other hand to define menus we have to consider an instance of the class Menu: practically one has a sequence of "label" strings which will be shown to the user and moreover a sequence of selectors which will be executed when the corresponding string is selected. Furthermore, we note that opening a Smalltalk/V window obliges to define its size: of course it is not possible to carry out this technical operation in a tutorial environment and thus the interface always provides windows spanning the whole screen. When several menus are associated with the same text pane and the window is activated, the Dispatcher that manages the text pane will show the menu which is currently the value of the variable of instance menu. We mention that having prototypes of the objects "menu" and "window" offers valuable advantages in comparison with the traditional programming.

Now let us consider the internal architecture of the interface. In order to depict it we use a graph whose vertices represent the classes forming the interface and whose edges indicate the hierarchical relations between the classes (Fig. 2).

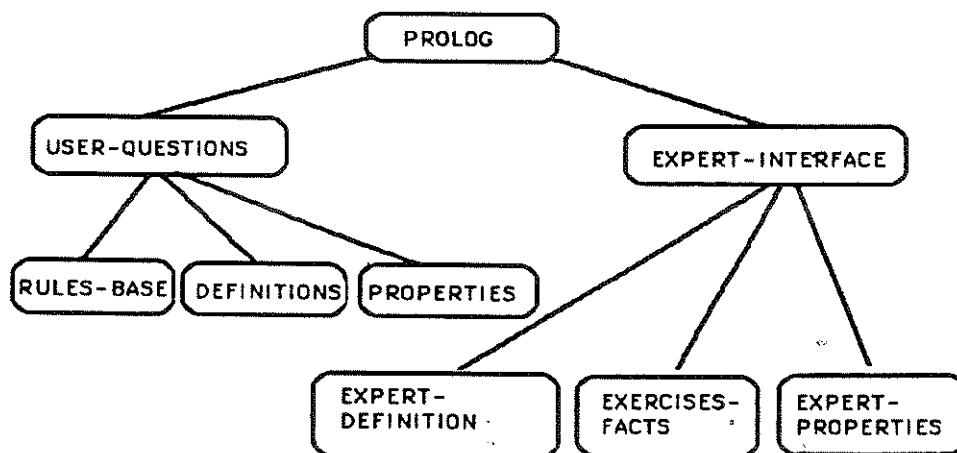


Fig. 2 - The classes of the module interface

In Fig. 3 the four modules the interface consists of and which do not necessarily obey subclass ties are shown:

- 1) user interface
- 2) graphic interface
- 3) pseudo-natural language interface
- 4) module devoted to build the knowledge base

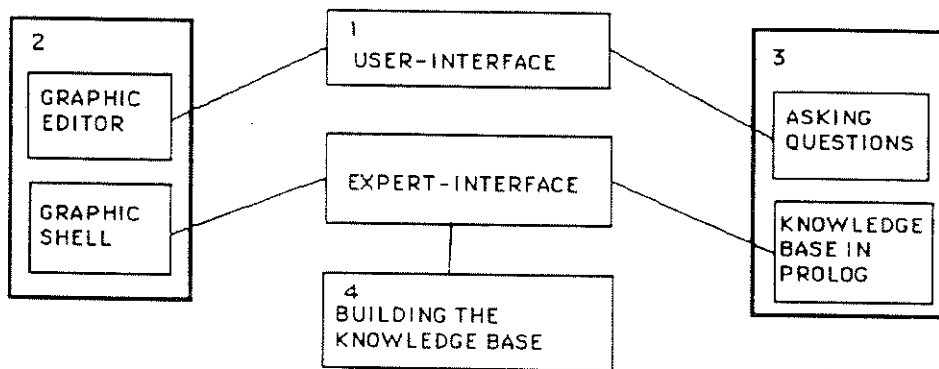


Fig. 3 - Structure of the module interface

The modules 1 and 3 contain two submodules devoted to the student and the expert, respectively. The user interface includes the screen pictures available to the user, each of them is associated with a graphic editor that allows geometrical drawing and moreover permits to ask questions about the proposed exercises by using an appropriate pseudo-natural language. The user interface is organized in classes as shown in Fig. 4 .

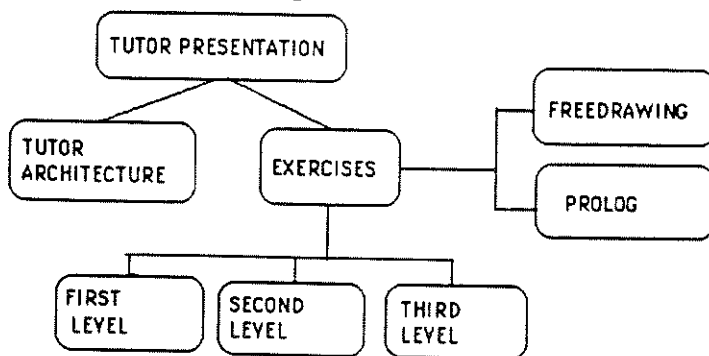


Fig. 4 - The classes forming the user interface

Each node represents a class and the levels of the tree characterize the hierarchical structure of this module. The class TutorPresentation allows accessing the modules TutorArchitecture or Exercises and it has the ultimate supervision of the various activities. The module TutorArchitecture permits to examine the system structure. This class includes the methods to carry out several operations such as opening a window by a text pane or a menu, drawing a graph, showing a graph on the window currently activated. We note that a graph has been implemented by means of an instance of the class NetworkNode. Then the display selector set to an instance of the class permits to show the graph on the text pane.

A major problem to be tackled was the organization of screen pictures and menus so that some feel is given to the user about the architecture of the system.

The module Exercises has been constructed in such way that three levels of difficulty can be managed according to the student's learning level. When an exercise is selected by the tutor the student can implicitly access both the graphic module (to carry out a drawing) and the Prolog module (whose duty is to help the student by supplying him definitions, information and related theorems and corollaries). An instance of the class Exercises definitely is not to be considered as an usual exercise, but it is a module managing the exercise via two different kinds of knowledge: "Text" and "Prolog". The "Text" knowledge is a file containing the text of the exercise and some hints. The class "Exercises" contains those methods which allow showing a file on a text pane, shifting to another window, carrying out the exercises (Fig. 5). The system includes all the methods that are required to interface the tutorial module. Moreover, there are methods in the class Exercises which permit to manage the interaction between the student and the knowledge base (e.g. it is possible to insert proof lines related to the current exercise). The interaction takes place by means of an appropriate pseudo-natural language.

In order to cope with a correct management of the interfacing process the Smalltalk/V structure "Prompter" has been utilized.

It is an instance of the class Prompter and is a window + pane which allows the system sending a message to the user and, in turn, receiving an input message. The message is the label of the prompter and is used by the student to ask a question and to insert proof lines. It is worth noting that a student question is assigned to an instance of the class String and is associated with a variable of the class Prompter. But, as the Prompter is present in the class Exercises, some difficulties might originate when the question has to be interpreted by other modules: to cope with this problem the student's question is inserted into a global variable, called Line that can be accessed by all the modules of the system. It is duty of the interpreter to translate the question into Prolog and then the knowledge base is consulted. Thus when one of the subclasses of the class Exercises is selected, the related window and text pane will be activated. We note that in the class Exercises is present, beyond the variable of instance Menu, also a variable text pane containing the description of another text pane. In such way we were able to define a screen picture different from the usual ones. In fact, the window is divided into two parts: the description of the upper part is in the variable ThePane inherited by the class TutorPresentation and this half-window is used by the student to

furnish the solution of the exercise, the description of the lower part is in text pane and the pane is used for the dialogue.

In order to permit the student to draw the module GraphicInterface can be activated by means of the method GraphicEditor (Fig. 6). This module is chained to the tutorial module whose underlying philosophy are the well known principles of George Polya [6].

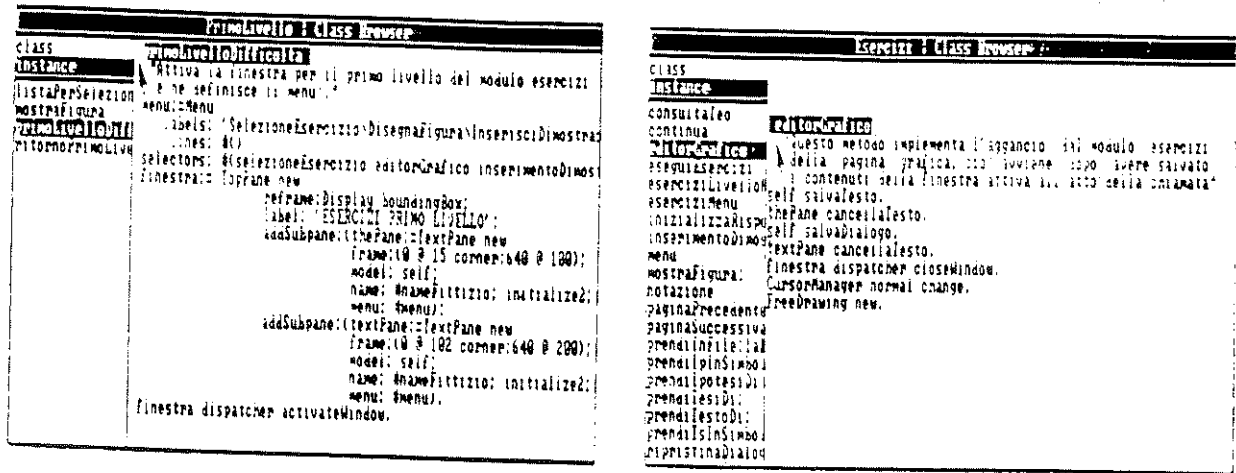


Fig. 5,6 - Two sample methods

The student can draw, cancel, move geometric pictures across the screen. Moreover, the expert can define, for each exercise, a string of several graphical hints: in such way the student can receive a specific help related to the particular difficulty encountered. The module has been implemented by means of two subclasses, Free-Drawing and Free-Drawing-Shell. The first allows the student drawing, the second, via the methods contained therein, permits to get a graphic shell.

All the pictures associated with an exercise form a global dictionary which is an instance of the class Dictionary, i.e. a collection of couples key-value where the key is the name of the exercise and the value is still a dictionary containing graphical objects (circles, ellipses, lines, etc.). The tie between the modules GraphicInterface and Exercises is managed by the message FreeDrawingNew which provides to activate the hard disk for saving information. This trick is necessary since Smalltalk/V allows an efficient interaction only for hierarchical classes. We might consider as global the contents of the text panes but this choice would have produced very slow compilations.

The management of the sequence of exercises is devoted to a suitable structure based on circular queues. When the difficulty level is selected, the corresponding queue will be activated and the next exercise is singled out. Now let us consider the consultation module: the student can ask questions about the geometrical properties of the current exercise and the structure of the module is shown in Fig. 7.

The class Prolog contains a method which implements the predicate Consult and in such way the classes Prolog can communicate each other. The message Consult (definition(x,y),PrologNew) allows consulting the base of definitions whichever is

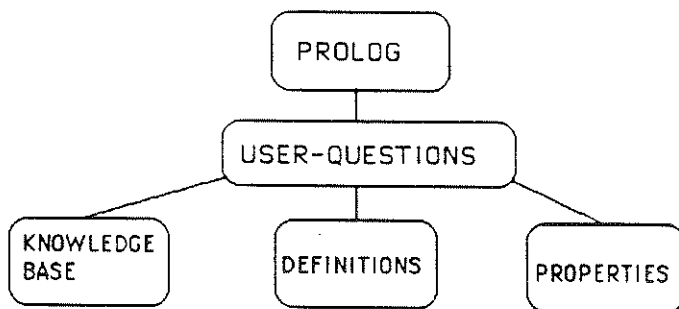


Fig. 7 - The structure of the consultation module

the class the message belongs to. The knowledge base has a similar organization and it can be easily built by the expert.

Finally, we mention that the proof model is represented by an AND-OR graph and the algorithm used is shown in Fig. 8. An example of graph is reported in Fig. 9.

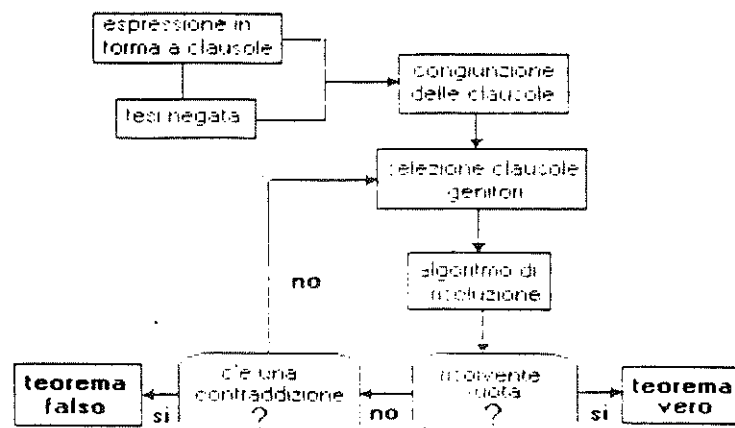


Fig. 8 - The solution algorithm

Ip : $\langle dba = \langle dbc \rangle, ad \perp ba, cd \perp bc \rangle$ Id : $\alpha_2 = a, \alpha_3 = b, \beta_2 = c, \beta_3 = d$.
 A = Ipotesi + Identazioni. Base di conoscenza: P, Q, R, S, T, U.

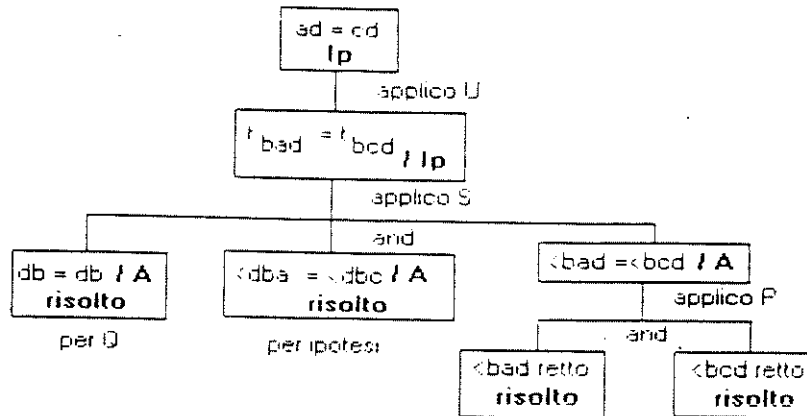


Fig. 9 - An example of AND-OR graph

In our opinion the internal architecture of the interface to SHOOP deserved most of the attention. Now we can only skim some of the screen pictures to give some feel of what the system looks like for the user.

The student has the following opportunities:

- to examine the system architecture
- to consult theoretical lessons before practicing
- to select an exercise.

Practicing is the most interesting activity: in fact, the screen pictures are divided into two parts: the upper one for the proof lines, the second one for the dialogue student-tutor (Fig. 10).

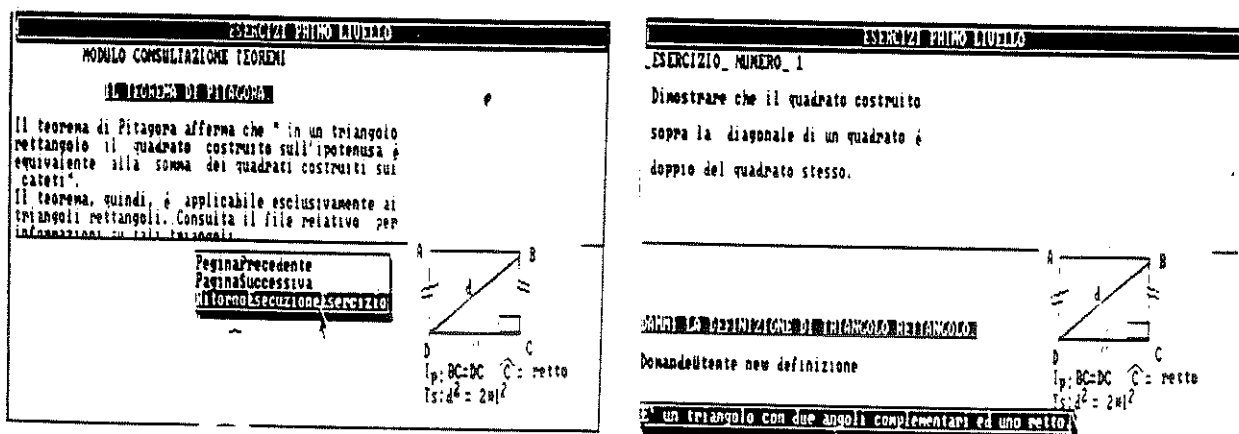


Fig. 10 - A geometric exercise

Moreover, the student can draw by using the mouse and the module GraphicHelp is at his disposal if any drawing problem occurs. Also the expert can carry out drawings which yet associated with the exercises. He can use a dictionary of drawings for each exercise and of course he can furnish all appropriate textual information. We emphasize that the expert is not required to be a programmer: in fact he can consult several menus and he has yet to answer the questions the system asks.

CONCLUDING REMARKS

The SHOOP project is currently in progress and has to be pursued towards a fully operational shell. In fact, we mentioned before that the other modules (student model, teaching and expert modules) of the shell still have an elementary structure. Yet the interface has a general validity and offers quite a sophisticated simplicity. The geometric tutor is currently used for test with a variety of users: although it has yet to be fully evaluated it is hoped that the shell described can be regarded as a way in which the potential of ITS shells as useful classroom tools can begin to be realized. Moreover, we think that the OOP paradigm will be considered more and more a strategic tool for the construction of ITS and authoring facilities.

REFERENCES

- [1] J.R. Anderson, The geometry tutor, Proceedings IJCAI, p.1, 1985
- [2] E.Chouraqui & C.Inghilterra: Conception d'un systeme expert d'EAO de la geometrie, Proceedings Cognitiva 87, Paris, 1987
- [3] E. Fischetti & A.Gisolfi: From Computer Aided Instruction to Intelligent Tutoring Systems, in press in "Educational Technology", 1990.
- [4] E. Fischetti & A.Gisolfi: Architecture of an authoring system in Smalltalk/V, Proceedings 7th ICTE, Brussels, 1990
- [5] A.Gisolfi & G.Moccaldi: An ITS for the factorization on algebraic expressions in Smalltalk/V, In press in "Educ.Train.Tech.Int.", 1990.
- [6] G.Polya: Come risolvere i problem di matematica, Feltrinelli, 1982.
- [7] D.Py: Mentoniez: an ITS in geometry, Proceedings 4th Int. Conf. AI & ED, Amsterdam, p.202, 1989.