*Int. J. Engng Ed.* Vol. 12, No. 6, p. 423–427, 1996
Printed in Great Britain.

# Embedding Authoring Support in an ITS for the Learning of Object-Oriented Programming

A. ZEKL
*IBM Germany, CBT projects, Am Fichtenberg 1, D-71083 Herrenberg, Germany*

I. MORSCHEL
*Institute for Computer Science, University of Stuttgart, Pfaffenwaldring 47, D-70550 Stuttgart, Germany*

*Intelligent Tutoring Systems (ITS) are of increasing importance for education in many areas. Existing authoring systems are no great support for authors of learning material. Authors should possess knowledge about the subject matter and pedagogical knowledge, and they need to acquire programmer skills, because most authoring languages are similar to normal programming languages. We propose a hypertext-based ITS with tools for structuring the subject matter without programming. Pedagogical knowledge is inherently realized in the system using AI-planning techniques. The authoring components are embedded in an ITS for the learning of object-oriented programming. It comprehends tools to support the visualization, animation, critique and testing of object-oriented programs written in Smalltalk.*

## INTRODUCTION

INTELLIGENT Tutoring Systems (ITS) are of increasing importance for education in many areas. However, the development of ITS is very time consuming and therefore expensive. ITS will gain widespread use only if they are easy to develop and maintain. Normally authors have to be experts of the subject matter, pedagogical experts and software engineers because they need to program explicitly all parts of the ITS. We propose an ITS with components for authoring support so that authors only have to be experts in the subject matter. In the following, we introduce a learning environment called SmallTutor, which is being used to master object-oriented programming.

## THE SMALLTUTOR SYSTEM

The SmallTutor system consists of three separate components: the authoring support tools; the ITS itself; and tools which help students to develop their own Smalltalk applications.

Our ITS is hypertext-based, like many of the existing systems. The great disadvantage of hypertext-based ITS is that they do not adapt the hypertext structure to the student's knowledge. We overcome such problems by using planning techniques which makes our system flexible. The hypertext network and therefore the structure of the learning material is dynamically adapted to the student's knowledge state. The exact functions of the different components will be described later.

The authoring tools can be used to develop learning material on different topics. The ITS can then be used to teach these topics. The SmallTutor tools are developed especially for the learning of object-oriented programming (e.g. Smalltalk-80).

Figure 1 shows the whole system.

## THE ITS

The heart of our system is a hypertext system. The system uses planning techniques to adapt the presentation of the learning material to the student's needs. The domain knowledge exists in the form of hypertext documents. We use the expression hypertext document for one page of the hypertext network—in contrast to the literature, where it often means the whole hypertext network. Its granularity is too fine for constructing a curriculum. Based on the article of Peachey and McCalla [1], the learning material is structured in concepts. 'Each concept corresponds to some unit of subject matter which the CAI system might potentially teach to the student...' [1]. For the learning of a concept, they defined prerequisite knowledge, which states the causal relationships between different concepts. This leads to several possible plans to achieve the learning goal (plan-generator in Fig. 1). A plan is executed (plan-executor in Fig. 1) by presenting the corresponding learning material to the student. If a plan goes wrong, which means that the student

* Accepted 25 July 1996.

cannot reach a learning goal, the next possible plan is taken. Learning success is modelled in a simple student model, which contains a list of learned concepts or misconceptions. There are severe restrictions for planning systems which are based on formal methods. 'However, while sound and complete planning algorithms have been published, none of them are heuristically adequate (assuming enough expressiveness to be interesting). Even for small problems, they are simply not usable in practice' [2]. They are mainly not usable in practice because of performance problems. Such a planning algorithm is only applicable in small knowledge domains. The acceptance of the learning system depends on its performance. Therefore, we made some extensions to this planning technique to overcome these problems. Our basic plan is the so-called guided tour, which is defined by the author and leads the student through the whole hypertext network. Therefore, replanning is only necessary if a student cannot learn a concept. We test the learning success with multiple choice questions. There are two other situations when replanning takes place. First, if a student does not follow the guided tour, then the system will search for the student's learning goal and supply a new appropriate order of the hypertext documents. Second, one great advantage of our system is that students can mark concepts as known before the beginning of the learning session. The remaining hypertext documents are then dynamically reordered, so that the students do not have to go through all the subjects every time they search for some information. This will increase the acceptance of the system. Beginners take the guided tour through the whole material, while for experts the system provides exactly the required information.

As we have seen, replanning only starts under certain situations, but the performance problem still remains; therefore we use heuristics based, for example, on different learner types. Students can choose their preferred learning strategy (theory or examples preferred, explorative learning, etc.) before beginning the learning session. Some information is already in the hypertext network, such as typed documents and semantical links, which are also used as constraints for finding the appropriate plan.

In the following we will look at the tasks of authors to develop learning material for this system.

## AUTHORING SUPPORT

### *Introduction*

Like simple Computer-based training programs, ITS normally need to be programmed. Therefore, the developers of ITS have to be experts in the subject matter, experts in pedagogical issues and software engineers. However, we normally only have experts in the subject matter (the so-called authors) with little knowledge about software

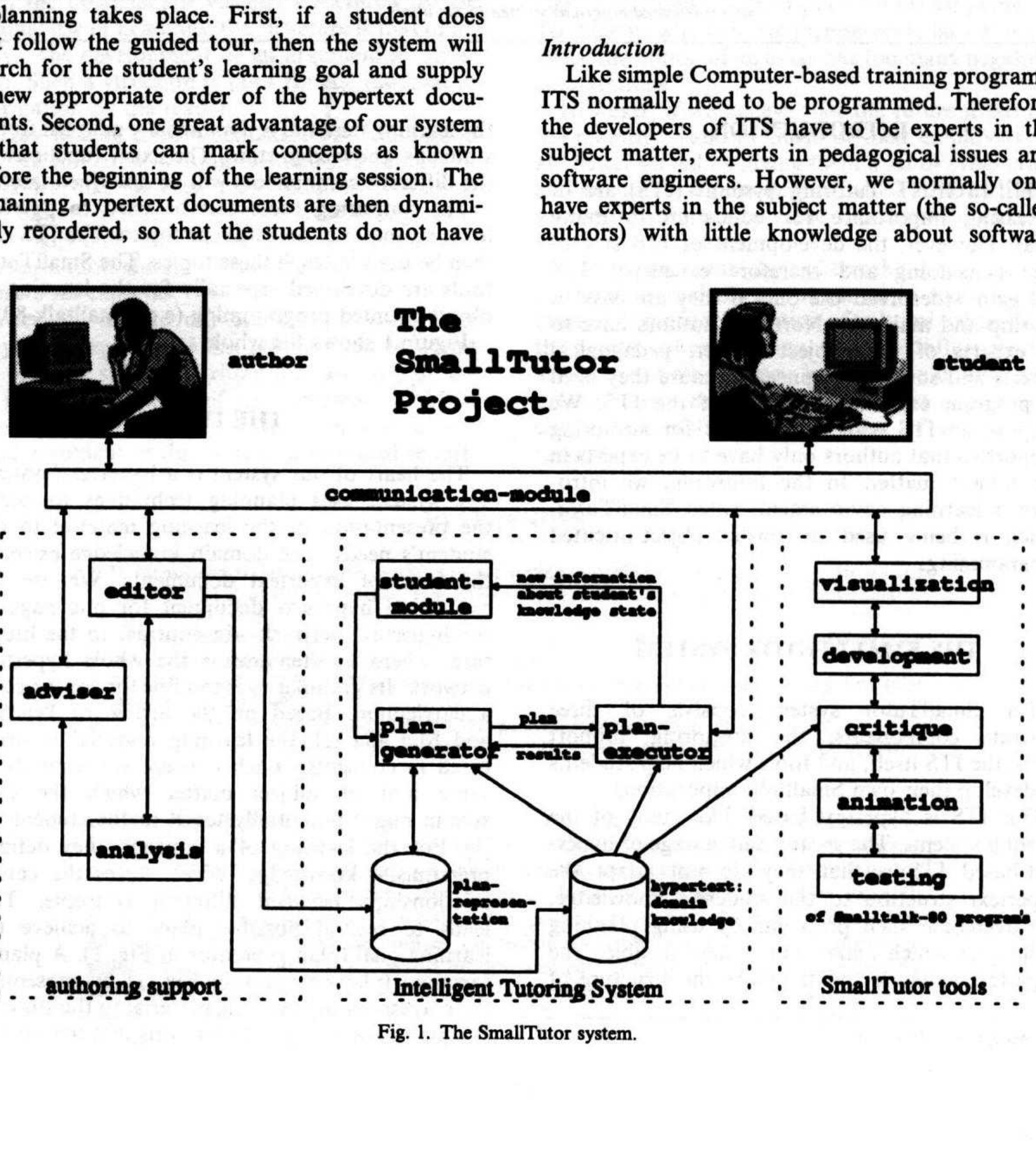

Fig. 1. The SmallTutor system.

engineering. A lot of authoring languages and authoring systems for the developers of learning material exist. But authoring languages are not really different from other programming languages. Authoring systems give too restricted possibilities in developing a learning system. Significant for both is the lack of support for design decisions concerning the structuring of the learning material and pedagogical questions. Our aim is to support authors in al these questions. Besides this, authors should not be burdened with programming and pedagogical questions. It is evident that the methods and tools for authoring support therefore depend on the architecture of the underlying ITS.

*The different levels of authoring*

Because our system is hypertext-based, authors are concerned with the inherent problems of hypertext, including disorientation and cognitive overhead [3]. There is a limited capacity of the human mind to grasp information. It is not possible to process more than about seven pieces of information at the same time [4]. Hierarchical structuring seems to be a way to overcome these problems. 'The human mind can accommodate any amount of complexity as long as it is presented in easy-to-grasp chunks that are structured together to make the whole... No matter how these principles are addressed, they always end up with hierarchic decomposition as being the heart of good storytelling' [5]. Therefore, we developed a graphical editor for hierarchically structuring hypertext [6]. On each level there is a limited number of possible documents, so the structures cannot become too vast and incomprehensible. The already developed learning units for our system normally consist of ~300 documents. with this editor, authors are able to structure the hypertext, define the guided tour through the hypertext, type documents and add keywords. Other aids are different analysis tools, which examine the developed hypertext structures on consistency, the lack of links, gaps in the guided tour and so on. We are still developing a knowledge-based adviser which evaluates all the analysis results and gives concrete design advice.

For the planning techniques which are used in the system, authors should build concept graphs. These graphs show the prerequisite relationships between the concepts. If there are concepts with the same prerequisites, an author can choose the order of these concepts in the learning session. This order is motivated by pedagogical concerns, but this is the only pedagogical information an author should implement in the learning material. All other pedagogical and didactical strategies are inherently implemented in the heuristics for the planning. The building of concept graphs is also supported by a graphical tool, because the problems, like the complexity of graphs, are the same as for structuring hypertext.

If the learning material is structured on these two levels, an author should additionally fill the hypertext documents with content. Authors are confronted with a lot of problems. Writing hypertext is much different from writing linear texts. For example, a notable problem is the text cohesion. We cannot refer to an antecedent proper name by a personal pronoun if they are not on the same page, because the order in which students go through the hypertext documents changes. Nevertheless, the text which results from the order of the hypertext documents must remain coherent. These problems are difficult to analyse automatically. Therefore it is necessary to have guidelines for authors. In our system, these guidelines will be implemented as a passive, knowledge-based adviser.

For examples and more detailed information see [7].

## SMALLTUTOR TOOLS

The ITS and the authoring tools are not bound to a special subject matter. We use the system to teach object-oriented programming (OOP) as a way to evaluate the environment and its methodologies. For this aim, some tools have been developed to support the learning of OOP. In the following, the visualization, animation and critique tools are presented.

We have use Smalltalk-80 [8] as our programming language because of its uniformity and elegance. The learning of the syntax and the environment appear to be mastered easily. However, the learning of the class library and the correct understanding of the dynamic behavior of programs seem to be difficult (e.g. the role of inheritance and polymorphism in the message-passing mechanism).

The visualization of the program characteristics is an important factor in its comprehension. We based our graphical representation on Booch's approach [9], which presents classes and respective objects together with their methods and variables. We made some simplifications and adaptations, which turn it into a Smalltalk-like representation (e.g. classes with their class categories, methods with their protocols, etc.).

The notation exposes only the part of the Smalltalk program which appears to be relevant to the students. For example, the internal information of a class/object (like its methods and variables) and the associations between them (like is-a, instance and used relationships) are exhibited. Other irrelevant information is not presented, because it is not essential for a first glance at a program and can easily be obtained from the source code.

We believe that an approximate and intuitive comprehension can rapidly be obtained from a graphical visualization. We can quickly recognize features that are familiar to us and have random access to any part of a picture. Text, on the other hand, is sequentially recognized. We can easily identify the relationships between the elements in a picture. With a text, this is much more difficult.

The algorithm animation, in turn, reveals the dynamic characteristics of a program. Among other things we can inspect variables and follow the control flow of a program. This is important for the comprehension of its behavior, because some aspects cannot be easily recognized by the student, e.g. through inheritance or polymorphism. We believe that in this way the student's own experiences in trying different solutions for a problem could be better supported.

The animation is based on the visualization, which defines a graphical debugger for Smalltalk programs. In extension to its browser, one can trace and investigate both data and control structures at run time.

Additionally, a learning environment should provide a feedback about the quality aspects of the developed programs. Such a facility is not commonly incorporated into many systems, but plays a significant role as a mechanism to judge and to meliorate the learner's designs.

SmallTutor aims to increase programmer productivity by providing a feedback during the learning phase. This is achieved by:

- evaluating the programmer's software by applying object-oriented software metrics;
- giving the programmer feedback concerning quality aspects of the developed programs.

Embedding a critic module in a learning environment allows the following:

1. Feedback is given to a learner concerning the quality of the created programs, since he/she as a beginner does not have sound criteria to evaluate a program, or he/she possesses misconceptions.
2. A good programming style is presented to the learner, as a good learning environment should do.
3. The boundaries between a critic module and a learning environment for programming are very closely related. When the learner makes a mistake or does not achieve a good solution, the system should provide help. This can be done by presenting learning material with the reason and an explanation of the errors made.
4. Quality aspects like understandably, reusability and maintainability of a program should be guaranteed.

A novice of a programming language and its environment is confronted with some problems [10]:

- When is an object-oriented program in a good style?
- Are there rules one can apply to develop good object-oriented programs?
- Which metrics should one employ in order to determine if a program is 'good' or not?

The measurement of the size and complexity of a software system can be used to aid in evaluating the quality aspects of its implementation and therefore be useful as a learning tool for staff members who are new to this paradigm.

There is much criticism about the use of metrics in the object-oriented software development process. One of them is that their objectives are not well understood and their application in a development environment requires great effort. To overcome such problems, we introduce *SmallCritic*, a tool to evaluate and meliorate object-oriented programs written in Smalltalk. It is embedded in the environment for the learning of object-oriented programming [11].

SmallCritic analyses object-oriented programs by applying construction rules that distinguish between:

- the static and dynamic structure of a class or an object;
- the static and dynamic relationships between classes and/or objects.

For examples and more detailed information see [12].

## IMPLEMENTATION

The implementation is realized in Smalltalk-80 release 4.1 from ParcPlace [8].

There is a lack of flexible interfaces in commercial hypertext systems, and therefore we developed our own hypertext system and also our own expert-system shell. Another reason was to facilitate the public use of this Intelligent Tutoring System and the authoring tools.

The software is available via ftp:

```
ftp iassnb.ias.uni-stuttgart.de
login: ftp
passwd: your e-mail address
```

*Directory: pub/lernsys*

This directory contains the file readme.text with information on the SmallTutor project as well as three subdirectories:

1. The directory *authoringsystem* contains the latest version of the ITS with the planning module but without the SmallTutor tools. There is a *readme* file with installation instructions.
2. The directory *smalltutor* contains the latest version of the ITS without the planning module, thus only hypertext-based, but with all SmallTutor tools. There is a *readme* file with installation instructions.
3. The directory *papers* contains some more publications on this subject. Read *INDEX.txt* for a summary.

For printed documentation (only available in German) please write to Martin Seidel, IAS, Pfaffenwaldring 47, D-70550 Stuttgart, Germany.

## CONCLUSION

We have shown the architecture of an ITS and an 'authoring system' for this learning system. Authors of learning material have to be experts only in the subject matter. Several authors (mainly students) have already developed learning units for the ITS using the existing tools and the underlying methodology. There has been great acceptance and positive reactions. The ITS is already being used for teaching students object-oriented programming.

*Acknowledgements*—We would like to thank the German Research Society (Deutsche Forschungsgemeinschaft DFG) as well as the Brazilian Council for Research (CNPq) for their financial support of this work.

## REFERENCES

1. Peachey and McCalla, Using planning techniques in intelligent tutoring systems. *Man-machine Stud.*, **24**, 77–98 (1986).
2. E. Wilkins, *Practical Planning*. Morgan Kaufman Publishers (1988).
3. Conklin, Hypertext: an introduction and survey. *IEEE Comput.*, September, 17–41 (1987).
4. G. A Miller, The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychol. Rev.*, **63**, 81–97 (1956).
5. T. Ross, Structured analysis (SA): a language for communicating ideas. *IEEE Trans. Software Engng*, 16–34 (1977).
6. Zekl, Authoring support for the development of intelligent tutoring systems. *Proceedings of the EAEEIE 93*, Prague, September 1993.
7. Zekl, Rechnerunterstützung für Autoren bei der Erstellung von Lernprogrammen für Intelligente Tutorielle Systeme, PhD thesis, University of Stuttgart (1995).
8. *Objectworks\Smalltalk Release 4.1, Users Guide*. ParcPlace Systems (1992).
9. Booch, *Object-Oriented Design with Applications*. Benjamin Cummings Publishing Co. (1991).
10. K. Lieberherr and I. Holland, *Assuring Good Style for Object-Oriented Programs*. IEEE Software (1989).
11. J. Morschel, About methods and tools to master object-oriented programming. *Proceedings of TaTTOO 94*, Leicester, UK, January 1994.
12. I. Morschel, Ein integriertes wissens-basiertes Tutorsystem für die Ausbildung in objektorientierter Programmierung, PhD thesis, University of Stuttgart (1995).

**Andreas Zekl** is an electrical engineer, graduating in January 1992 from the University of Stuttgart. He obtained a doctoral degree in 1995 at the University of Stuttgart, Institute for Automation and Software-Engineering (IAS). He is currently working for IBM Germany, Education and Training as project manager for the development of Multimedia Computer-Based Training programs. Since, June 1994 he has been a council member of the European Association for Education in Electrical and Information Engineering (EAEEIE).