

FITS: A Framework for ITS—A Computational Model of Tutoring

MITSURU IKEDA AND RIICHIRO MIZOGUCHI

I.S.I.R., Osaka University

8-1, Mihogaoka, Ibaraki, 567 Japan

This paper summarizes our research activities concerning FITS, a framework for ITS, and discusses the major results obtained thus far. FITS is a domain-independent framework developed to examine what functions can be realized within a domain-independent framework among those which are needed in ITS. In the current implementation, FITS is composed of six building blocks, each of which covers an essential task for teaching. This means that FITS is a realization of a configuration of the ITS task structure from the viewpoint of generality. FITS has been implemented in Common ESP on a UNIX workstation. Two prototype systems for geography and chemical reactions have been developed using FITS through which the feasibility of FITS is proved.

INTRODUCTION

It has been demonstrated by several pioneering studies of ITS that achievements in knowledge engineering can contribute to realization of educational activities (Brown, 74; Carbonell, 70). Since then, a large number of useful techniques has been proposed for the development of ITSs, for the construction of the student model and representation of the expert knowledge. Those achievements contributed not only to the promotion of ITS research itself but also to the establishment of various fundamental tech-

niques in artificial intelligence (Wenger, 87). It is only in recent years, however, that there have been discussions on the general problems in the design of ITS, such as the performance required for knowledge bases in ITS and the inference schema needed in ITS. Vivet (1990), for example, developed a shell for ITS called AMALIA, which is composed of two expert system shells. One is for problem solving and the other is for pedagogical decision making. Finer decomposition of the tutoring activity can be found in GTE (Van Marcke, 1990) and COCA (Major & Reichgelt, 1992), in which tutoring expertise is represented by a set of generic control schema composed of tutoring task components of relatively smaller granularity. Both of them are motivated by the work concerning the generic task concept in knowledge base systems (Chandrasekaran, 1986).

The major purpose of this research is to discuss the general design methodology for ITS through the design and development of a domain-independent framework for ITS posing a number of questions, such as "What kinds of inference schema are needed in ITS?", "What can be the framework to control the system?", and "Does there exist a domain-independent tutoring strategy?" To answer these questions, it is necessary to view the problem in a top-down way from the standpoint of generality. We know that it is rather difficult to find universally correct answers to such questions. For example, the answers from the artificial intelligence standpoint are not always correct from the educational standpoint. Through this research, however, we aim at finding the possibilities and limitations of the current state of art of computer science, especially artificial intelligence techniques applicable to educational software.

FITS, which stands for Framework for ITS, is a domain-independent framework developed to examine what functions can be realized as a domain-independent framework among those which are needed in ITS (Mizoguchi et al., 1991). In the current implementation, FITS is composed of six building blocks, each of which covers an essential task for teaching. The functionality needed for the student model module, for example, is realized as a domain-independent inductive inference algorithm and that for the tutoring module as 20 tutoring strategies, in which Socratic tutoring is included. This means that FITS is a realization of a configuration of the ITS task structure from the viewpoint of generality. The final goal of the domain-independent framework is to enumerate the generic tasks inherent to ITS and to provide the sophisticated building blocks for implementing integrated environments for teaching/learning systems.

The authors have been involved in the investigation of ITS for several years (Ikeda et al., 1988a, 1988b, 1988c, 1989, 1993; Kawai, 1987; Kono et al.,

1993b; Mizoguchi et al., 1988, 1991; Okuhata et al., 1992; Shimazaki et al., 1992). This paper summarizes our research activities concerning FITS and discusses the major results obtained thus far. The next section presents the underlying philosophy and an outline of the framework composed of student model, expertise and tutoring modules and a scheduler. Later, we discuss the student model and the tutoring modules, respectively. The scheduler is discussed in the final section of the paper.

OVERVIEW OF THE RESEARCH

Research Objectives

Our research has been conducted aiming at the following two major goals:

1. To reveal the inherent structure of ITS, and
2. To make it easy to build an ITS.

As a result, much attention has been paid to designing a domain-independent framework for ITS. Most of the existing ITSs are domain-dependent, for their architecture and the knowledge embedded is related to respective domain knowledge. Therefore, it is not clear which knowledge and which parts of the system can be reused to tutor another domain knowledge.

How much do we know about the common architecture of ITS? What architecture, what knowledge, and what mechanisms can be reused in developing teaching other material? FITS is designed based on the results of our investigations. FITS represents the inherent problem solving structure of tutoring independently of each teaching material, which makes it easy to build an ITS, since control structures common to most ITSs are already embedded in the framework. In an ideal case, the developer has only to describe the knowledge to teach declaratively.

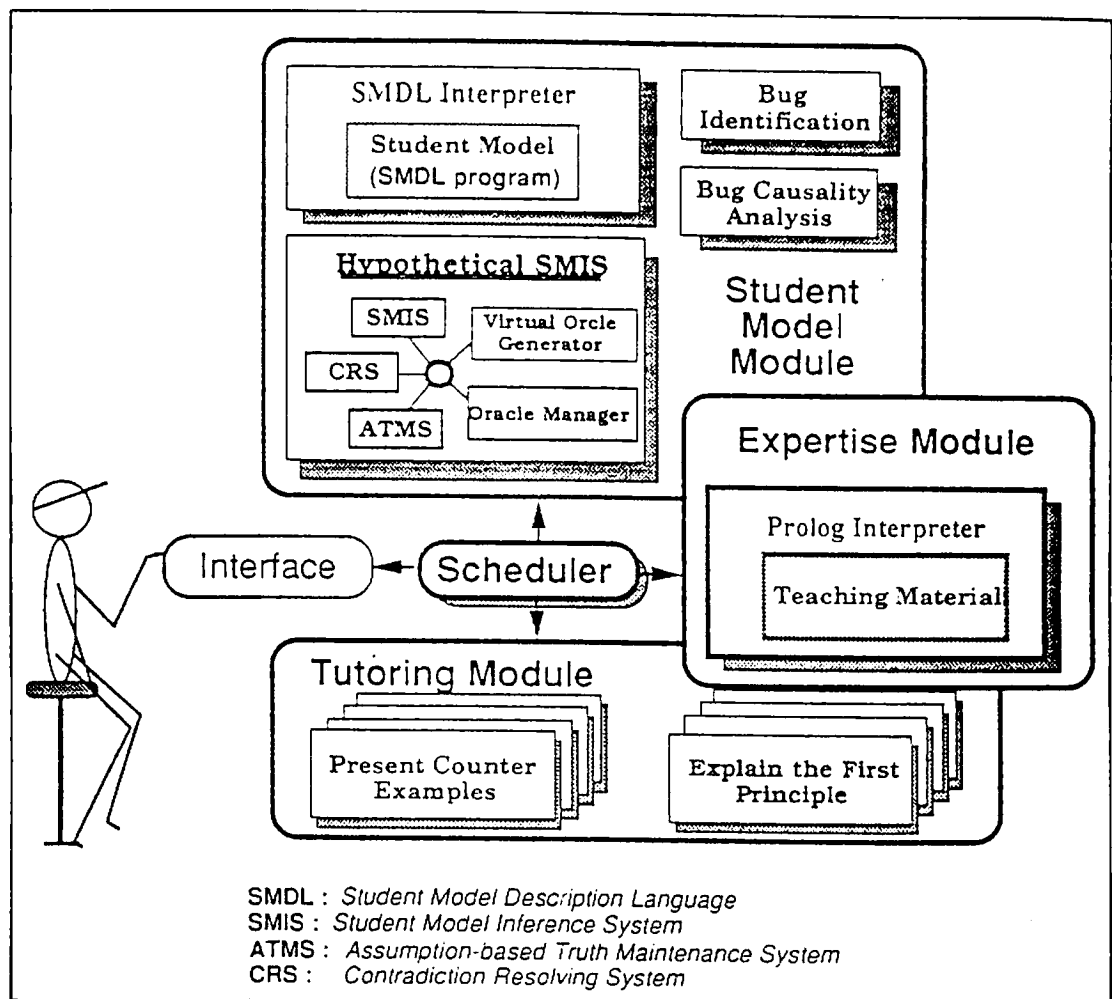


Figure 1. System architecture of FITS

The Framework

An ITS is composed of the following three major functional modules:

- Expertise module,
- Student model, and
- Tutoring strategy module.

These are further divided into primitive tasks. Figure 1 shows the FITS architecture containing these tasks. Among these modules, the one for student modeling plays an essential role, since performance of an ITS depends largely on how well it knows the student. We developed a modeling language called SMDL (Student Model Description Language) based on Prolog (Ikeda et al., 1993). In our framework, the student model is represented as an SMDL program, so modeling students can be considered as a program syn-

thesis problem and instruction can be considered as debugging of the SMDL program. Therefore, tutoring can be formulated as a two-step procedure composed of program synthesis and its debugging as shown below, which is a skeletal idea of our framework.

Intelligent tutoring = Program synthesis + Debugging.

Now a serious question arises: How can we synthesize an SMDL program? An inductive inference system called HSMIS (Hypothetical Student Model Inference System) has been developed as an answer to this question, and it works as a central engine of the framework. In the current implementation, the expertise module consists of only one building block, that is, the Prolog interpreter which interprets the teaching material described in Prolog. Our framework also has an efficient scheduler which makes an appropriate decision on what to do next. It is designed based on the SOAR architecture (Laird et al., 1987) to realize highly flexible scheduling.

FITS also has a set of domain-independent tutoring strategies. One may think tutoring cannot be domain-independent, since it seems to require a lot of domain-dependent data or facts. It is true that tutoring strategies require domain-specific data and facts in explanation and other interactions. By domain-independent tutoring strategy, the authors mean the mechanism is domain-independent, not the data used. Furthermore, when and what strategies the system should take is also determined domain-independently. The strategies consist of two major groups: deductive ones based on explanation and inductive ones based on hints using domain-specific examples (data). The detailed description of the tutoring strategy is given in a later section of this paper.

Building Blocks of the Framework

Since an ITS can be viewed as an expert system in education, expert system technologies can be applied to the construction of ITSs. Most of the conventional expert systems are constructed using production rules which provide fairly low-level descriptions of expertise. Systems based on production rules do not reflect the inherent structure of the task, so one may have a lot of difficulty in describing expertise in terms of rules. The concept of generic tasks (Chandrasekaran, 1986) attracts much attention, and these can be building blocks of expert systems. By generic tasks, we mean domain-independent but task-dependent chunks of control structures. In order to obtain a generic framework, all the modules have to be designed independently of

teaching material. We investigated the primitive modules to identify domain-independent structures which act as building blocks of ITS.

Each building block is designed as a domain-independent problem solver for its corresponding generic task. Implementation of building blocks is based on a problem space approach. It consists of a problem space, a domain-independent problem solver and heuristics. A building block can solve any problem defined in the problem space, where definition of the space is very important to realize generality. Heuristics specific to the domain, if available, enable it to solve the given problem more efficiently. Thus the design process of building blocks consists of the following three steps:

1. Formal description of the problem space,
2. Implementation of a general problem solver for that space, and
3. Definition of heuristic knowledge available.

When formulation of the problem space is made at an appropriate level of abstraction, high generality and efficiency can be attained simultaneously. Moreover, users of the framework can encode the knowledge in terms of vocabulary at an appropriate conceptual level.

A FRAMEWORK FOR ITS

As is shown in Figure 1, FITS is composed of several building blocks. Those building blocks are activated whenever necessary by the scheduler. The scheduler determines the next tutoring action and activates the corresponding building blocks. In the following, the inference mechanism of the building blocks shown in Figure 1 is outlined.

Student Model Module

The tasks assigned to the student model module are the identification of the student's understanding (construction of the student model), the identification of the incorrect knowledge of the student (bug identification), and the analysis of the incorrect knowledge of the student (bug causality analysis). As a tool for constructing the student model, the student model description language SMDL and the inductive inference algorithm SMIS for

SMDL, which is based on MIS (Shapiro, 1982), are employed in the proposed framework (Ikeda et al., 1993; Mizoguchi et al., 1991). In SMDL, not only "true" and "false" can be used as the truth values, but also "unknown" and "fail." The language inherits the features of Prolog as a logic-based language and appropriately represents the understanding state of the student. HSMIS is a non-monotonic inductive inference algorithm. By embedding ATMS in HSMIS, it can cope with inconsistent responses of students (Ikeda et al., 1993). We will explain the details of SMDL and HSMIS in the accompanying paper (Kono et al., 1994).

In the bug identification task, the bug in the student model is identified. SMDS (Student Model Diagnosis System), which is a subsystem of SMIS, is a general problem-solver in this building block. When a mismatch arises between the responses of the student and the student model, SMDS identifies the rule causing the mismatch. The bug identification task can extract the bug by diagnosing the student model based on the correct answer derived from the expert knowledge.

Bug analysis is a task where the correct knowledge and the correct knowledge corresponding to the buggy knowledge of the student is identified based on a similarity measure between pieces of knowledge.

SMDL: Student Model Description Language. SMDL is a student modeling language which is executable and can simulate the problem solving behavior of the student. It is an extended version of Prolog and takes four truth values: true, false, unk, and fail. The first three values correspond to "success" in Prolog and the last to "fail." True stands for the student answering yes, unk for unknown, false for no, and fail for the system not knowing what answer the student will return. Discrimination among the four truth values is done by introducing an auxiliary argument interpreted by a meta-interpreter.

SMIS: Student Model Inference System. SMIS is an inductive inference system for SMDL. It is an extended version of MIS (Shapiro, 1982). A pair of a problem and an answer to it is called an oracle and is used for inductive inference as data to be covered by the model obtained.

To model the student from her behavior, SMIS asks some questions which contribute to disambiguation of alternative hypothesis. This means that SMIS knows what it doesn't know about the student's understanding and has the capability to automatically generate questions. The dialogue with the student in constructing the model can take various styles, as shown in Figure 2. All the dialogue can be transformed into oracles as shown in the parentheses.

SMIS applies two operations repeatedly to the model—(a) removal of an incorrect rule and (b) addition of a new rule—until the model comes to be able to cover all the oracles given. An incorrect rule and an uncovered oracle to be covered by the model are identified by SMDS (Student Model Diagnosis System). SMDS traces the computation process of the current model and checks the results with the student's answers. To cover an uncovered goal, SMIS searches for a rule to be added to the current model. Candidates for the rule to be added are generated by refinement operators, which are defined by modifying those defined in MIS.

HSMIS: Hypothetical SMIS. Tutoring is intended to guide students toward a better understanding of teaching material. This means that the learning process is essentially attained through the change of their minds, and hence the consistency of students' answers can be easily lost. Therefore, student modeling methods should be able to automatically manage the consistency of a student's answers in order to follow the student's change. This requires nonmonotonic inductive inference. In our framework, ATMS: Assumption-based Truth Maintenance System (deKleer, 1986) is employed for this purpose. The system augmented with ATMS is referred to as HSMIS (Hypothetical SMIS). The main task of ATMS is to manage the consistency of a set of assumptions used by the problem solver, SMIS in our case. Assumptions in ATMS are data believed by the system independently of other data. In HSMIS, assumptions are oracles, since every activity in HSMIS is dependent on oracles and inconsistency appears among oracles.

The latest version of the student modeling module of FITS is called THEMIS (Kono et al., 1992, 1993b, 1994), which has the capability to model the student who has contradictory knowledge in his/her head.¹

Modeling Heuristics. From the pragmatic viewpoint, a student modeling method is required to run in real time and with a reasonable number of questions given to students. To attain this, domain-dependent heuristics may be easily incorporated into the modeling mechanism. Note here that we can introduce the buggy knowledge into HSMIS. Given some typical bugs specific to the teaching material under consideration, the search procedure in HSMIS can begin the search for rules starting from these bugs, which makes search very efficient. When it fails, it resumes the search from the original root of the search tree. Therefore, the introduction of heuristics does not lose any generality.

```

Does rice grow in Australia?
Answer? no. [grow(rice,australia,T):false]
Where does rice grow?

Answer? Japan. [grow(rice,japan,T):true]
Please select the countries where rice grows.

a. Japan b. Canada c. Australia
Answer? a,c [grow(rice,japan,T): true,
             grow(rice,canada,T): false,
             grow(rice,australia,T):true]

```

Figure 2. Discourse for student modeling

Bug Identification. Basically, our framework does not have a concept of bugs, so we have to analyze the student model built in order to know what bugs he/she has. SMDS, a module in SMIS, is again used for this purpose. In student modeling, it is used for identifying incompleteness and incorrectness of the model using the student's answers as oracles, since its objective is to obtain a model which explains the behavior of the student. In bug identification, however, the model is assumed to represent the student correctly and what we have to do is to find out bugs in it. Bugs are defined as differences between the model and the expertise (teaching material). So, SMDS checks the model using the answers of expertise as oracles. Thus, SMDS identifies incorrect and missing rules in the model.

Bug Causality Analysis. After bug identification, what to do next is to correct the bugs. For that purpose, the tutor should know where the bugs come from and why they occur, since such information helps give the student appropriate instruction. This task is referred to as bug causality analysis. It is an important but difficult task in ITS. Our framework deals only with the former information; that is, it identifies the correspondence between buggy and correct rules but discards the reason why the student comes to have them. For the example shown below, the generic problem solver of this module identifies that (S1) and (S2) correspond to (E1) and (E2), respectively, and (E3) is missing in the student model.

Student model(SMDL)
(S1) A:-B,C.
(S2) A:-D,E,F.

Expertise knowledge(Prolog)
(E1) A:-B,D.
(E2) A:-D,E.
(E3) A:-F.

The algorithm to calculate the similarity between two rules is realized as a general mechanism. The basic idea in this algorithm is that the coincidence between the sets of instances derived by the rules can be used as a measure of the similarity between the rules. By this algorithm, the size of the difference set between the instance sets is reflected by the similarity and hence the semantic difference between rules is reflected by the similarity. The details of the decision procedure for the similarity are omitted (for the details, see Mizoguchi et al., 1988). It is designed to consider the errors which often are produced as deformation of the configuration of rules (such as exchange, insertion, and omission of predicate). Higher priority is given to exchanges of concepts with similar semantics, such as "hot" and "warm."

Expertise Module

The expertise module plays a critical role in ITSs because it contains the domain knowledge which students have to learn. Viewing from a computational point of view, the role of this module is rather static. Needless to say, the tutoring behavior of an ITS is composed of two kinds of control structures—static and dynamic ones. By static control structure, we mean some pedagogically important ordering of topics to teach as well as the organization of domain knowledge itself which is determined during the design phase of the ITS. By dynamic control structure, we mean that the control of the behavior is made adaptively during the tutoring phase according to the student behavior. Basically, the former requires pedagogical expertise more than the computational techniques to realize the control, while the latter requires an advanced computational framework which supports highly adaptive behavior. As described earlier, FITS is designed as an advanced computational framework supporting various behaviors necessary for realizing highly individualized tutoring.

For these reasons, the expertise module of FITS has a very simple structure; that is, it has only a Prolog interpreter and expertise is written in Prolog declaratively. The functions this module has are to solve problems, to answer questions about the domain knowledge, and to evaluate the correctness of the student's solution. All these functions are implemented based on a Prolog interpreter without any substantial augmentation. Descriptions of these mechanisms are omitted here, since they are rather straightforward. As an example of domain knowledge, a portion of geographical knowledge represented in Prolog is shown below.

```

grow( Plant, Place ):-
    suitable_temp( Plant, Place ),
    suitable_soil( Plant, Place ),
    suitable_lay( Plant, Place ),
    has_irrigation( Place ).
has_irrigation( Place ):-
    natural_irrigation( Place ).
has_irrigation( Place ):-
    artificial_irrigation( Place ).
suitable_temp( rice, japan ).
suitable_soil( rice, japan ).
suitable_lay( rice, japan ).
natural_irrigation( japan ).
artificial_irrigation( japan ).

```

The predicates in the rule's body are in AND-relation. The first rule whose head is "grow" means "The Plant grows in Place, if the Place satisfies all the four conditions in the rule's body." The rules with the same head such as the second and third rules are in OR-relation. A rule without a body part is called a fact. The first fact in the above example means "The temperature of Japan is suitable for rice to grow".

For a simple chemical reaction domain, facts and rules are written in Prolog as follows:

```

salt( X, Y ):-
    base( X, _ ),
    acid( Y, _ ).
react( X, Y, Z ):-
    salt( X, Y ),
    precipitation( X, Z ).
precipitation( ca, so4 ).
precipitation( ba, so4 ).
acid( cl, 1 ).
acid( so4, 2 ).
base( ca, 2 ).
base( ba, 2 ).

```

The first and second rules mean that if X is a base and Y is an acid, then XY is a salt, and if XY is salt and XZ precipitates, then reaction $XY + HZ \rightarrow XZ + HY$ occurs, where H denotes a hydrogen atom, respectively. For example, when $X = Ca$, $Y = SO_4$, and $Z = Cl$, the reaction $CaSO_4 + 2HCl \rightarrow CaCl_2 + H_2SO_4$ occurs, since $CaSO_4$ is a salt and $CaCl_2$ precipitates.

Both of the above two kinds of domain knowledge are different from each other at the conceptual level, while they share the same characteristics

at the implementation level, which guarantees FITS can deal with them both in the same manner. Readers can easily imagine the SMDL programs corresponding to the above two Prolog examples because of the similarity between SMDL and Prolog.

There are alternatives to the framework design concerning which functions should be provided and placed in what module. We know that some other functions, such as explanation and qualitative simulation, are missing in FITS and are crucial in some cases. These issues remain for future work.

Tutoring Strategies

The tutoring strategy is one of the most important items of knowledge in an ITS, since it is responsible for direct interaction between the student and the system. FITS has several kinds of domain-independent tutoring strategies which are described in this section. The operation of the strategies is conducted by the Scheduler described in the next section.

Because the tutoring strategy seems to depend largely on teaching material at a first glance, one may think it is rather difficult to design generic tutoring strategies. Although it is almost impossible to implement a tutoring module without knowing teaching material at all, it must have some properties inherent to tutoring itself. An obvious example is one which gives the student correct answers immediately when he/she makes mistakes. This strategy is based on a simple mechanism for printing the file containing the correct answers, though it refers to the domain-specific data in the file. The key idea employed here is to put all the domain-dependent information into data. We have identified 20 tutoring strategies shown at the bottom of Figure 7. Some of them are explained below.

- *Presentation of deep knowledge* guides students to the correct understanding by providing the first principle which supports the target knowledge.
- *Explanation of derivation process of a correct answer* explains the derivation process which justifies a correct answer.
- *Suggestion of trace of solution process* suggests that students trace the derivation process of their answers.
- *Presentation of an example conflicting with the student's understanding* aims at letting students notice their buggy knowledge by giving a single characteristic example conflicting with their understanding.

- *Presentation of some examples of common factors of interest* gives some examples to students which promote their refinement process of their own knowledge, expecting them to think of the flaw in their own knowledge inductively.
- *Presentation of attributes of examples* provides the key attribute of examples to refine their knowledge, for example, the attribute concerning the missing condition.
- *Presentation of intermediate solution* encourages students to resume thinking by presenting an intermediate solution.

One can synthesize many macro-strategies using these strategies. FITS has a macro-strategy referred to as Instance-Based strategy (IB) and one referred to as Extended IB strategy (EIB).

IB tries to guide students' self-correction of their knowledge by providing them with some critical examples from which a contradiction is directly derived. If the student misses a correct rule, it gives him/her some problems from which the rule can be induced. EIB gives a student practice in the inductive way of thinking to refine his/her understanding conducted by IB strategy.

The IB strategy building block contains a subsystem for generating appropriate problems. It generates problems according to the generate and test paradigm. Candidate problems are easily generated by, for example, instantiating some rules in the expertise model.

Table 1
Problems Generated by the Tutoring Module

B	C	Expertise A (A:-B,C)	Student A (A::B)	instances
true	true	true	true	p_1, p_2, \dots
true	false	false	true	q_1, q_2, \dots
false	true	false	false	r_1, r_2, \dots
false	false	false	false	s_1, s_2, \dots

The operation of the strategy IB is described in the following, using the dialogue as an example, where the student understands incorrectly the rule if B and C then A as if B then A. Table 1 shows the instance (corresponding to a problem in teaching) for A, as classified according to the truth values of the predicate B and C. $\{p_i\}$, $\{r_k\}$, $\{s_j\}$ are the problems for which correct responses are anticipated from the model, and $\{q_j\}$ is the problem for which incorrect responses are anticipated. First, by giving $\{q_j\}$ to the student, the strategy guides him/her the self-contradiction. Then, it helps student to inductively think of the missing predicate C by giving $\{p_i\}$ and having him/her consider the difference between $\{p_i\}$ and $\{q_j\}$. For other types of bugs, such as missing-rule, extra-rule and extra-condition, we define similar strategies in a domain-independent manner.

For the generation of the instance to be presented to the student, it is possible to define the problem space formally and to introduce the additional domain-dependent knowledge to it. When the correspondence between rules (the rule in the teaching material knowledge and the rule in the student model) for generating instances to be presented are given together with the truth values, the SMDL interpreter returns a set of satisfying instances. The additional knowledge is introduced as the knowledge to determine whether or not each of the generated instances should be employed, such as the popularity, indicating to what extent the instance is known to ordinary students. For example, Alaska and Kiev are famous for their cold weather, and so on.

<pre>grow(X, Y, T₁): :- suitable_soil(X,Y,T₂), suitable_lay(Y,T₃), has_irrigation(Y,T₄). (A) Student model</pre>	<pre>grow(X,Y):- suitable_temp(X,Y),suitable_soil(X,Y), suitable_lay(Y), has_irrigation(Y). (B) Expertise</pre>
--	---

Figure 3. Examples of student model and teaching material

An example of tutoring dialogue conducted by the proposed system is shown in the following. After constructing the student model, the error analysis is executed. Then educational utterances are generated. Figure 3 shows the rules (A) and the teaching material knowledge (B), for which the correspondence is established. The student model is described by SMDL and the teaching material is described by Prolog.

Comparing the above two rules, it is seen that the predicate $suitable_temp(X,Y)$ (Place X has the temperature suited to the growth of crop Y) is missing in the student model. When the two rules are placed in correspondence as the tutoring object, the dialogue shown in Figure 4 is generated by the system. In this case, an instance problem must be presented to make the student aware that $suitable_temp(X,Y)$ is missing and convince him that the condition is necessary.

In Figure 4 (a), an instance for which the student model and the teaching material give the same answer (Tokyo) and instances for which they give different answers (Kiev and Beijing) are presented. The purpose of these questions is to make the student pay attention to the instance appearing in the following dialogue and to verify that there is no change in the understanding of the student. In (b), from the strategy *presentation of an example conflicting with the student's knowledge*, it is pointed out that there is an error in the understanding of the student, and an example conflicting with her understanding is given.

Then (c), (d), and (e) present the set of cities which satisfy all conditions for the growth of rice (Tokyo, Nan-ching, and Jakarta) and the set of cities for which the growth of rice is rejected only by the missing predicate. The student is asked whether or not he can inductively think of the factor that determines the possibility of rice growth between those two sets of cities in (f). For the student who answered that he/she cannot recall the factor that determines the difference, the system provides a hint to estimate the missing factor in (g). This part of the dialogue is directed by the strategy *presentation of some examples of common factors of interest*.

In (h), the common factors between the rules in the student model and the rules in the teaching material are listed, urging the student to deductively find the factor and the incorrectness of her knowledge is explained by the strategy *explanation of incorrectness of knowledge*. If the student indicated in (a) to (h) that he/she recalled the factor, that factor is selected in (i). If the student cannot identify the missing factor through those dialogues, the correct knowledge is described in (j).

- (a) Does rice grow in Tokyo?
 Answer ? *yes*
 Does rice grow in Bei-jing?
 Answer ? *yes*
 Does rice grow in Kiev?
 Answer ? *yes*
- (b) You made some mistakes. For example, rice does not grow in Kiev. Can you understand the reason? Answer ? *no*
- (c) You made 2 mistakes. Now, let us consider the problem.
- (d) Rice grows in the following places :
 Tokyo Nan-ching Jakarta
- (e) Rice does not grow in the following places :
 Kiev Bei-jing Detroit
- (f) Do you understand differences between above two examples?
 Answer ? *no.*
- (g) Can you think of a common factor which does not hold among Kiev, Bei-jing and Detroit?
 Answer ? *no.*
- (h) All places in (d) satisfy soil condition, lay condition and irrigation condition. But rice does not grow in Kiev, Bei-jing and Detroit since it does not satisfy another factor you missed out.
- (i) Please select a description which explains these differences from (1)-(3).
- (1) The temperature of Tokyo is suitable for rice to grow but the temperature of Kiev is not suitable for rice to grow.
 - (2) The people of Tokyo live on rice but the people of Kiev do not live on rice.
 - (3) Tokyo is not an inland area but Kiev is an inland area.
- Answer ? *2.*
- (j) It's not correct. Rice grows in places satisfying the following conditions:
- (1) The temperature of the place is suitable for rice to grow.
 - (2) The soil of the place is suitable for rice to grow.
 - (3) The lay of the place is suitable for rice to grow.
 - (4) The place has irrigation.

Figure 4. An example of educational dialogue

The primitive strategies included in IB are classified into two types. One group is to promote the inductive way of thinking and the other the deductive way of thinking. For example, the series of strategies, (c),(d), and (e), intend to have the students think of the missing factor inductively, while (g) and (h) help the student to solve the problem deductively.

All the utterances in both the IB strategy and the EIB strategy are generated by building blocks which are activated by the scheduler. To guide the students to better understanding, it is important to activate appropriate tutoring actions for each student's understanding state. In the next section, we will explain in detail the mechanism for scheduling and some kinds of knowledge sources used by it .

MODEL OF TUTOR'S DECISION MAKING

In this section, the reactive behavior of FITS will be illustrated through the description of the architecture and various knowledge sources for the scheduler. There are important knowledge sources for the decision making which determine tutoring reactive actions appropriate for various situations. Those are, for example, "when the understanding state of the student is not correctly estimated, his/her bug cannot be corrected until another cue is taken," "If the student does not have the prerequisite knowledge, it is difficult to present him a new topic," and so on.

Architecture of Scheduler

There can be various kinds of information that help the decision of the scheduler. Typical sources of such information are the student model, tutoring history, and teaching material. It may happen that one tutoring action is supported by some information and another by different information at the same time. The deterministic conflict resolution mechanism is not suited to such a situation. Figure 5 shows an architecture to solve the problem when there exists mixed information from various knowledge sources.

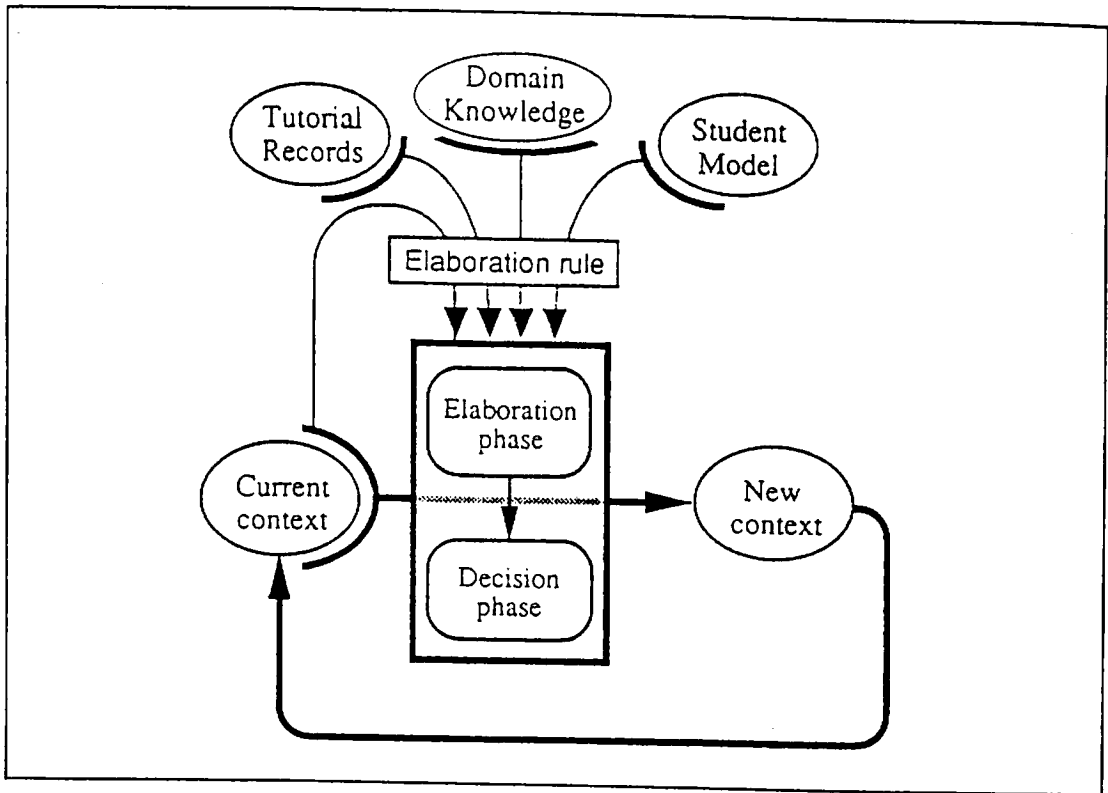


Figure 5. A mechanism for the scheduler

This architecture is based on the generic problem-solver SOAR by Laird et al. (1987). A cycle of execution on this architecture is composed of two phases, that is, the elaboration phase and the decision phase. This cycle is iterated until the decision-making is completed. As is shown in Figure 6, when a cycle is completed, the content of a slot of the context is determined. When all the slots of a context are determined, the context represents the tutoring action to be performed by the tutor.

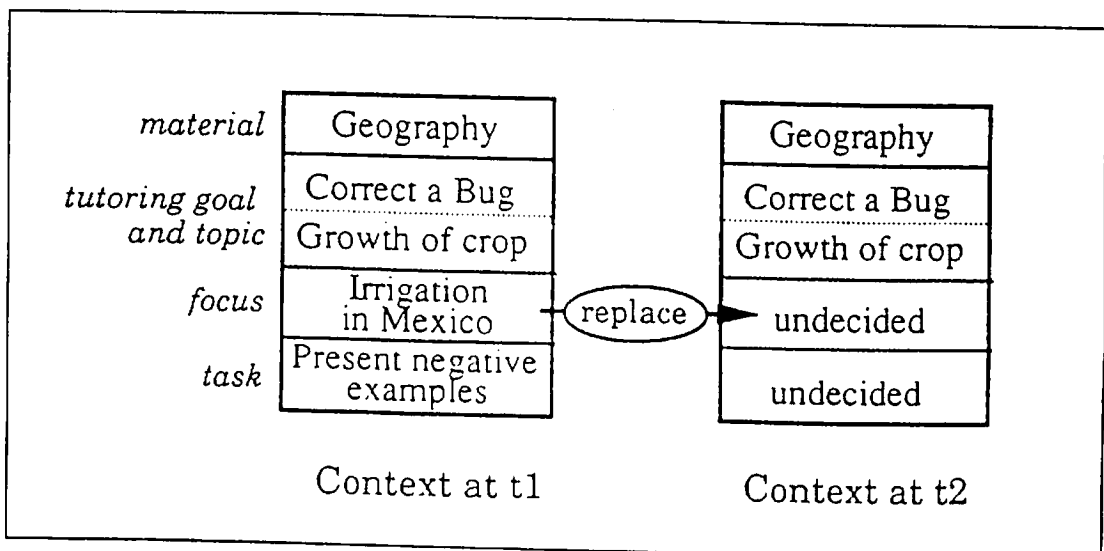


Figure 6. Context change caused by a decision cycle

In the elaboration phase, the information for decision making is collected from various viewpoints with if-then style rules called E-rules (i.e., elaboration rules, following the literature Laird et al., 1987). All the information is transformed into the normalized representation schema (preference), which essentially represent the partial ordered relation among the candidate objects for a slot. The decision phase selects the maximal object on the partial-ordered relation represented by the preference.

When the next tutoring action is determined, the problem space is set for the corresponding building block, and the control is transferred to the building block. The execution results of the building block are reflected in the tutoring history and student model, and are used as the information source in the next cycle. By employing such a decision procedure, many kinds of knowledge sources can co-exist without a contradiction (the competition is not considered as a contradiction, and interpreted properly by the decision procedure).

Elaboration Space

An architecture for scheduling has been discussed above. We next present knowledge necessary for scheduling. The scheduler has three kinds of information spaces for the elaboration phase:

- S1 space: This contains raw information dependent on the tutoring of specific teaching material. It contains, for example, "the student misunderstands item D," "an action for item D was taken at time T," and "item D is difficult to learn."
- S2 space: Abstract information of the tutoring status is useful for domain-independent decision making. While information described in S1 is obtained directly from the working memory, this space contains a description of abstract states of the student, such as "the student is weak at deep consideration," "the student does not understand a bug correction method," and "the student lacks basic knowledge."
- S3 space: Final information about tutoring actions is described in this space, where the description is made in terms of preference such as "acceptable," "reject," "better than," "worse than," "best," and "worst." These constitute a set of actions in a partial order. A typical example is "action A is better than B."

Knowledge Sources for Scheduling

The scheduling architecture shown above requires sophisticated knowledge to realize intelligent interaction between students and the system. For this purpose, we analyzed domain knowledge and tutoring strategies to obtain operation rules for scheduling all activities in FITS-built ITS. Those activities include student modeling, bug identification, bug analysis, and tutoring strategies. In this paper, however, we concentrate on the tutoring strategies, because they play the most important role in the tutoring processes. In this section, we first introduce the concept of goals of respective strategies and organize them based on their respective goals. The organization can be regarded as the specification of the reactive tutoring behavior of FITS (Shimazaki et al., 1992).

Goals of Tutoring Actions. To embody the sophisticated tutoring process, tutoring systems should have a precise grasp of the situation in a tutoring context and select a tutoring action most appropriate to the situation. To attain this goal, we need to know which strategy can support which part of the learner's learning process, based on a deep understanding of cognitive process of learning. We have analyzed some pedagogical dialogues and arranged the results as a network representing the learning process, in which a node corresponds to a learner's state and a directed link to a transition from one state to another. From the tutor's standpoint, nodes and links in the network can be regarded as target states and effects of the tutoring actions, respectively.

In other words, the goal of a tutoring action is to have the student reach the target state. We can design an inherent structure of the tutor's decision-making process by paying special attention to the tutoring goals. Figure 7 shows a hierarchical structure of tutoring goals. Note that the leaf nodes are tutoring strategies described earlier. The tutoring system makes a decision on how to interact with the student refining an abstract goal to a more specific goal according to the structure from root to leaf. For example, the path drawn by thick lines in Figure 7, shows a refining process from the root to the strategy *presentation of some examples of common factors of interest* in the case where the student misses some conditions of a rule. The tutor's intention in selecting the strategy is represented by the tutoring goals on the path, that is, "correct a bug," "remind student of a common attribute of examples," and "provide negative examples." This means that the tutor tries to remind her of the missing condition by giving negative examples to correct her buggy knowledge by herself.

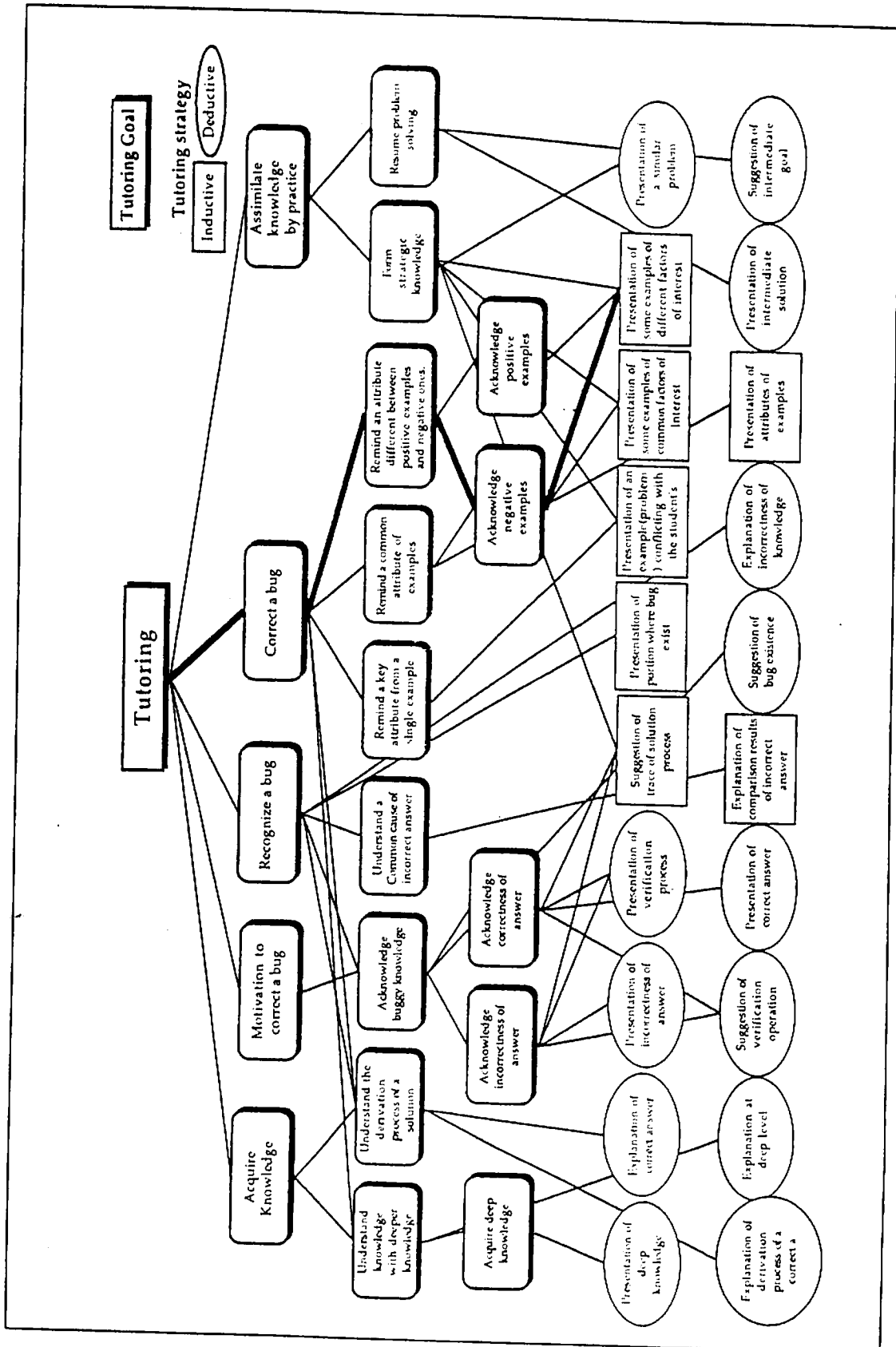


Figure 7. A hierarchical structure of tutoring goals

The decision-making process is realized by the scheduling architecture, the information about the situation is gathered by a set of E-rules, and the maximal object—that is, the most appropriate tutoring goal—is selected by the decision procedure. The candidate set of tutoring goals in a situation are the children nodes of the upper tutoring goal selected in the last decision cycle. The goals that appear in the refining process occupy the tutoring goal slots of the context.

Organization of Domain Knowledge. Although FITS is designed domain-independently, its applicability has some limitation. In other words, the strength of its applicability depends on the domain knowledge. We first examined the applicability of all the tutoring strategies identified, and found out the strength of effect of the application is different among them, though all are applicable to any domain knowledge (Okuhata et al., 1992). This suggests to us a good way of understanding the characteristics of domain knowledge.

In order to operate the above 20 tutoring strategies efficiently, we have to have a representation scheme for describing the characteristics of each domain knowledge. We analyzed several kinds of domain knowledge to realize an appropriate operation of the strategies. By analyzing the difference of the application effect, we identified four criteria as follows:

1. Ease of getting a guideline of error correction,
2. Ease of acquiring necessary attribute,
3. Applicability of verification operation, and
4. Amount of resource consumption of problem solving.

These four criteria suggest the four following dimensions for characterizing domain knowledge.

- **Declarative/procedural:** Suppose the tutoring strategy of *presentation of the correct answer* is activated. When the domain knowledge is declarative, this message suggests that the student should generalize (specialize) his/her knowledge when he/she answered “no(yes)” to the problem whose correct answer is “yes(no).” The message helps him/her a lot in bug correction, since it means he/she should add (delete) a rule to the set of knowledge or delete (add) a condition from (to) the rule. When the domain knowledge is procedural, on the other hand, this suggests no correction action other than something wrong with his/her procedure.

- **Abstract/concrete:** Considering the same strategy mentioned above again, when the concepts appearing in the domain knowledge are abstract, such as numbers, the message does not help the student remember the necessary attribute of the concept of interest. When the concepts are concrete, on the other hand, he/she can notice the attribute. For example, the message "Rice does not grow in Alaska" helps the student remember the attribute "weather," since Alaska is very cold.
- **Formal operations on the objects are defined or not:** The applicability of verification operations is dependent on whether formal operations on objects in the domain are defined or not. When such operations are defined, verification operations are applicable. For example, a linear equation is solved through an equivalent transformation of the equation and verified by substituting the solution for the variable and executing the calculation. In the case of the problems of the domain where no formal operations are defined, a solution is basically obtained through a generate and test method in which a dumb generator is used, that is, the verification operation mentioned above. Therefore, there is no other method of solving the problem.

The only way to verify the solution is to experiment. The feasibility of an experiment is usually dependent on the domain knowledge. Some are possible in real time, though some are not. Geography is a typical example in which an experiment is not feasible in real time, while chemical reaction and physics are possible, where experiments act in the same role as verification operations.

- **The size of search space:** The last dimension is the size of the search space of the problem. According to this, the "suggestion of intermediate goal" strategy is effective when the space is large.

On the basis of the above observation, we organize several kinds of domain knowledge as shown in Table 2.

Table 2
Classification of Domain Knowledge

Domain knowledge	Declarative/ procedural	Abstract/ concrete	Formal operations (experimental)	Search space
Subtraction	procedural	abstract	yes	small
Linear equation	procedural	abstract	yes	small
How to cook	procedural	concrete	no(easy)	large
Factorization	proc./decl.	abstract	yes	small
Concept of number	declarative	abstract	yes	small
Geometry proof	declarative	concrete	no(easy)	large
Chemical reaction	declarative	concrete	no(easy)	middle
Kinetics	declarative	concrete	no(easy)	middle
Growth of crops	declarative	concrete	no(easy)	small

Operation of the Tutoring Strategies. One of the most important factors for tutors in making a decision is to recognize each situation in the tutoring process appropriately. The task is not a trivial one, since the situation has to be defined from a variety of aspects. In our framework, a set of E-rules is responsible for the recognition task. This means that the quality of the model of the tutor's decision making largely depends on the expressiveness of the E-rule. Therefore, we identified information sources which are used for representing the rules for the operation of the strategies proposed. They are shown in Table 3. In order to realize domain-independent tutoring, we build an operation rule base to apply the above mentioned 20 strategies effectively.

Table 3
Information Sources for E-Rule

Domain Knowledge	Student Model	History of Learning	History of Tutoring
Precedence relation of topics Characteristics Difficulty of topics Curriculum	Degree of mastery Types of bug Reliability of the model Capability	History of Understanding History of bugs Time taken for solving problems Elapse time	History of problems Number of examples Kinds of examples Current tutoring goal

Operation rules are written as elaboration rules in the Scheduler described above. As discussed earlier, the scheduler has two kinds of rules for elaboration: E1-rules are for translation of information in S1 to that in S2, and E2-rules are from S1 and S2 to S3. Three examples of elaboration rules are shown in Figure 8. Both E1 and E2 rules are invoked until there is no rule to fire.

The enumeration of the information sources is important for us, to specify the reactivity of each building block, especially the tutoring strategy. However, we cannot describe the exact computational semantics of each entity of the information sources because it is defined by a large set of E-rules. So we will only give one example here: The application of the tutoring strategy, which promotes the inductive way of thinking by giving examples, should be applied only to the domain knowledge which is grounded on concrete examples. In the case of the ungrounded domain knowledge in which an abstract entity is dealt with, it is difficult to find good examples which have enough information for students to refine their knowledge. This is a typical example of the specification of the reactivity of a building block and is represented by the second rule in Figure 8.

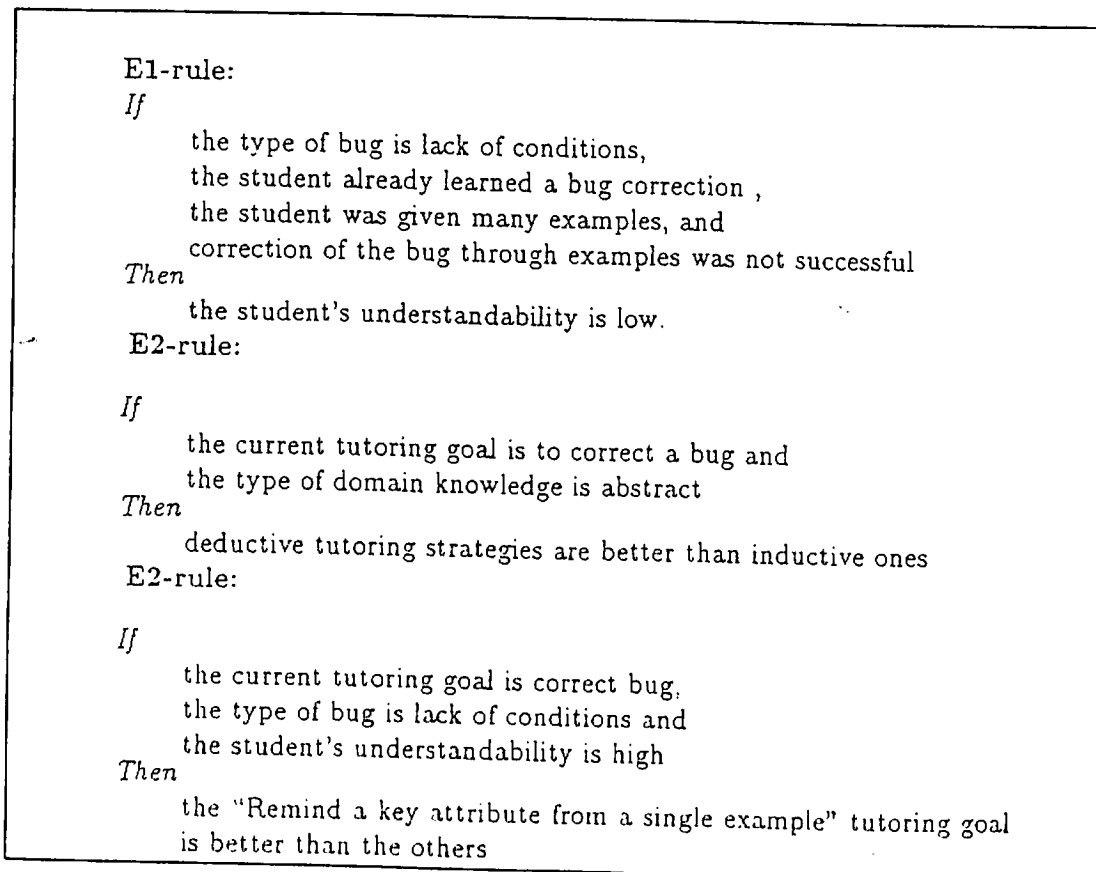


Figure 8. Some examples of elaboration (operation) rule

DISCUSSIONS

Table 4 summarizes functions and domain-dependent knowledge necessary for each building block. The mechanisms for HSMIS, SMDS, the calculation method of similarity between rules, and the example generation are designed as complete general problem-solvers. By this structure, the weakness of a knowledge-intensive system such as an expert system where the inference fails when the knowledge is incomplete is overcome. At the same time, the functions needed in ITS are realized formally.

Table 4
Organization of Generic Tasks

Generic Tasks	Problem Solvers	Domain Knowledge
Student modeling	SMDL+SMIS+ATMS	knowledge for search control and contradiction resolution
Bug identification	SMDS	_____
Bug analysis	Similarity calculator	knowledge of bug property
Tutoring strategy	Example generator	knowledge for example selection

In general, it is considered, however, that the formal framework cannot realize an efficient (intelligent) behavior without additional heuristic knowledge. This point can be solved by introducing the domain knowledge in parallel to the formalization of tasks. Table 4 shows the domain-dependent empirical knowledge that can be introduced into the building blocks. In constructing the student model, the knowledge concerning the bugs of the student can be introduced as the search-control knowledge for the refinement graph (i.e., the more plausible bugs should be searched with higher priority, etc.). When the model inference fails due to the inconsistent responses of the student, the search to dissolve the contradiction can be controlled based on the reliability of the student's responses (such as the function of the elapsed time).

In the bug analysis, the efficient analysis can also be realized by handling the correspondence between the empirically given bug knowledge and

the expert knowledge with the priority. In the tutoring strategy, high-quality examples that contribute to the bug-correction teaching can be generated by introducing the domain-dependent knowledge in the selection of the example. Thus, by introducing the complete problem-solver augmented by the domain-dependent knowledge, one can expect that the building blocks achieve the complete problem-solving with high efficiency.

CONCLUDING REMARKS

This paper has discussed the design principles and the mechanisms of the building blocks of a framework for ITS. Through this research, we aimed at contributing to the advancement of AI technology applicable to education. In general, such a goal will be achieved by computationalizing the functions lacking in conventional software. In the student modeling module, for example, there are many tough but valuable problems to be tackled in both computational and theoretical senses. However, these problems have not been addressed sufficiently on a well-defined computational/theoretical basis. The development of HSMIS contributes to building the computational basis for the student model module, that is, to formulate the diagnosis process and question generation process, to cope with the nonmonotonic property of a student's learning process, and so on.

Another objective of this research, that is, the enumeration of the computational agents needed for implementing reactive behaviors of tutoring systems, is achieved through the development of the tutoring module and the scheduler. The 20 tutoring strategies and the variety of knowledge sources for scheduling clearly specifies the reactivity of FITS.

FITS has been implemented successfully in Common ESP (Extended Self-contained Prolog) on a SPARC station (AIR, 1990). Two prototype systems concerning geography and chemical reactions have been implemented in the framework. We developed FITS by building the geography system. After the completion of the development, another system for chemical reaction was implemented in only two man-weeks, since the framework is almost completely domain-independent. The implementation required domain knowledge representation in Prolog, substitution of messages in the interface module, and the domain-specific knowledge shown in Table 4. The range of the verification is too small to conclude its validity from the educational viewpoint. Nonetheless, we think the experiment is enough to show that the aim of this research has been achieved, that is, to build a domain-independent computational model of an educational task.

We have taken two important criteria into consideration in the design process of FITS. One is that the problem addressed should be well-formulated. Needless to say, we can benefit from a discussion on the theoretical basis, if we try to formulate and solve the important problems which we come up with in the design process of intelligent tutoring systems. Development of a new computational mechanism as a solution to a well-formulated problem contributes to building the reusable building block library for ITS development.

The other criterion is that the important problems should not be avoided. Real contributions of artificial intelligence research to education will not be made until we solve the hard problems which the traditional methodologies cannot cope with.

However, there seem to be few accumulated results of efforts for overcoming the difficult but important problems in the AI-ED research area except for, for example, Hoppe (1994), Huang et al. (1991a), and Self (1993). One of the basic philosophies behind the research on FITS is that a computational framework commanding a wide-range view of ITS is one of the most important research issues in the AI-ED area for the next decade. The authors believe that to challenge the important but difficult problems is a good motivation for research which stimulates the research activities.

References

- AI Language Research Institute. (Ed.). (1990). CESP Language Guide, 5-1-1, Ohuna, Kamakura, Kanagawa 247, Japan.
- Brown, J.S., Burton, R.R., & Bell, A.G. (1974). *SOPHIE: A sophisticated instructional environment for teaching electronic troubleshooting (An example of AI in CAI)*. BBN Rep. 2790.
- Carbonell, J.R. (1970). An artificial intelligence approach to computer-aided instruction. *Trans. of IEEE, MMS-11*, 4, pp. 190-202.
- Chandrasekaran, B. (1986). Generic tasks in knowledge based reasoning: high-level building blocks for expert system design. *IEEE EXPERT*, 1 (3), 23-30.
- de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28, 127-162.
- Hoppe, H.U. (1994). Deductive error diagnosis and inductive error generation for intelligent tutoring systems. *Journal of Artificial Intelligence in Education*, 5(1), 27-49.
- Huang, X., McCalla, G.I., Greer, J.E., & Neufeld, E. (1991). Revising deductive knowledge and stereotypical knowledge in a student model. *User Modeling and User-Adapted Interaction*, 1, 87-115.
- Ikeda, M., Mizoguchi, R., & Kakusho, O. (1988a). Design of a general framework for ITS. *Proc. of ITS '88* (pp. 82-89), Montreal.

- Ikeda, M., Mizoguchi, R., & Kakusho, O. (1988b). A hypothetical model inference system. *Trans., on IEICE of Japan*, J71-D, 9, pp. 1761-1771 (in Japanese).
- Ikeda, M., Okuhata, K., Nomura, Y., Mizoguchi, R., & Kakusho, O. (1988c). Scheduling mechanism for intelligent CAI systems. *Symposium on Intelligent methodologies in Education*. IPS of Japan, pp. 63-70, Tokyo (in Japanese).
- Ikeda, M., & Mizoguchi, R. (1989). Student model description language SMDL and student model inference system SMIS. *Trans., on IEICE of Japan*, J72-D-II, 1, pp. 112-120 (in Japanese).
- Ikeda, M., Kono, Y., & Mizoguchi, R. (1992). Nonmonotonic model inference: A formalization of student modeling (Technical Report AI-TR-92-10). Artificial Intelligence Research Group, ISIR, Osaka University, Ibaraki, Osaka, Japan.
- Ikeda, M., Kono, Y., & Mizoguchi, R. (1993). Nonmonotonic model inference: A formalization of student modeling. In *Proc. IJCAI'93*, Chambery, France, pp.467-473.
- Kawai, K., Mizoguchi, R., Kinoh, H., Ganke, S., Kakusho, O., & Toyoda, J. (1987). A framework for intelligent CAI systems based on logic programming and inductive inference. *New Generation Computing*, 5(1), 115-129.
- Kono, Y., Ikeda, M., & Mizoguchi, R., (1992). To contradict is human—Student modeling of inconsistency—. In C. Frasson, G. Gauthier, & G.I McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 451-458).
- Kono, Y., Tokimori, T., Ikeda, M., Nomura, Y., & Mizoguchi, R. (1993a). A student model building method based on formalization of nonmonotonicity. *Journal of Japanese Society for Artificial Intelligence*, 8(4), 488-498. (in Japanese).
- Kono, Y., Ikeda, M., & Mizoguchi, R. (1993b). A modeling method for students with contradictions. In *Proc. AI-ED 93* (pp. 481-488), Edinburgh, Scotland.
- Kono, Y., Ikeda, M., & Mizoguchi, R. (1994). THEMIS: A nonmonotonic inductive student modeling system. *Journal of Artificial Intelligence in Education*, 5 (3), 371-413 .
- Laird, J.E., Newell, A., & Rosenbloom, P.S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.
- Mizoguchi, R., Ikeda, M., & Kakusho, O. (1988). An innovative framework for intelligent tutoring systems. In P. Ercoli & R. Lewis (Eds.), *Artificial Intelligence Tools in Education* (pp. 105-120).
- Mizoguchi, R., & Ikeda, M. (1991). A generic framework for ITS and its evaluation. In R. Lewis & S. Otsuki (Eds.), *Advanced research on computers in education* (pp. 63-72). North-Holland.
- Major, N., & Reichgelt, H. (1992). COCA: A shell for intelligent tutoring systems. *Proc. of ITS'92* (pp. 523-530).

- Okuhata, K., Shimazaki, K., Ohta, Y., Nomura Y., & Mizoguchi, R. (1992) Classification of teaching material from the viewpoint of tutoring strategy Trans., on IEICE of Japan, J75-A,2, pp. 305-313 (in Japanese).
- Self, J. A. (1993). Formal approaches to student modeling. In G.I. McCalla & J Greer (Eds.), *Student modelling*. Springer-Verlag.
- Shapiro, E. Y. (1982). Algorithmic program debugging. MIT Press.
- Shimazaki, K., Okuhata, K., Sakane, K., Nomura, Y., Ohta, Y., Ikeda, M., & Mizoguchi, R. (1993). Operation of tutoring strategies for FITS: A generic framework for intelligent tutoring systems. *Journal of Japanese Society for AI*, 8 (2), 212-221. (in Japanese)
- Van Marcke, K. (1990). A generic tutoring environment. *Proc. of ECAI '90* (pp 655-660).
- Vivet, M. (1989). Knowledge-based tutors—Towards the design of a shell. *International Journal of Educational Research*, 12, 839-850.
- Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Morgan Kaufmann Publishers.

Note

1. THEMIS is not completely incorporated into FITS yet.

Acknowledgements

We wish to thank Dr. Y. Kono, Mr. K. Okuhata, Mr. K. Shimazaki, Mr. T. Tokimori, Mr. K. Sakane, Mr. K. Morihiro, and Mr. N. Murotani for their contribution to this work. This work is supported in part by Grant-in Aid for Scientific Research on Priority Areas of the Ministry of Education, Science, and Culture of Japan under Grant No. 03245106.