UDC 519.68 Journal of Computer and Systems Sciences International, 32(1), 1994

The Construction and Application of Student Models in Intelligent Tutoring Systems*

P.L. BRUSILOVSKIY
Moscow

This paper surveys research results in the development and application of student models in intelligent tutoring systems. A classification of student models is given together with model development methods. Attention is focused on the problems of the application of such models in tutoring systems. A critical analysis of research in this developing field is provided based on extensive Russian and foreign literature (80 references).

Key words: Artificial intelligence, student model, intelligent tutoring systems, intelligent help systems.

INTRODUCTION

According to the artificial intelligence (AI) handbook published in 1982, computer-assisted tutoring represents one of the main applied fields of artificial intelligence [1]. Studies devoted to AI methods and tools for creating more effective tutoring systems were begun abroad in the 1960s and 1970s. The new types of tutoring systems, which have come to be called intelligent systems, were able to overcome the drawbacks of frame-oriented tutoring systems through the use of artificial intelligence. This made it possible to employ the experience of knowledge representation and application accumulated to date in the field of artificial intelligence.

Three types of special knowledge are used to achieve effective human-tutor training: knowledge of the subject matter, knowledge of teaching strategy and methods, and knowledge of the student. Such special types of knowledge may also include standard human knowledge of interaction: speaking, demonstrating objects to the student, and understanding the student's response (words or figures). The fragments of such knowledge required to realize a specific element of a training course in frame-oriented tutoring systems were introduced in fixed form into the text of the individual frames of this course. Required knowledge in intelligent tutoring systems (ITS) is explicitly identified and represented, as a rule, by a variety of artificial intelligence methods and technologies. By using such knowledge, the ITS is capable of performing various tutoring functions (help during problem solving, identification of the causes of student errors, and the choice of an optimal teaching) with nearly the same degree of reasoning as achieved with a human tutor.

One distinguishing feature of a good teacher is the ability to adapt to a specific student during the teaching process. The basic method of adapting to a specific student in a frame-oriented tutoring system involves choosing each successive teaching frame as a function of the preceding response of the student or an assessment of the student from the preceding lesson. Intelligent tutoring systems take into account the entire scope of a student's knowledge of the subject being taught and even the personality features of the student in choosing each successive teaching action. Knowledge of a specific student is represented in an ITS as a student model. The student model (SM) is constantly updated during the teaching process in accordance with the changes in model-reflected characteristics of the student and is employed to adapt the ITS to a specific student.

An ideal ITS must be capable of representing and utilizing all the types of knowledge outlined above. This enables the structure of an ideal ITS to be defined as a set of interacting modules (Fig. 1) in each of which knowledge of one type

*Originally published in Tekhnicheskaya Kibernetika, No. 5, 1992, pp. 97–119.

ISSN1064-2307/94/0001-0070

is concentrated. Such a view of the structure of an ITS was first proposed in [1] while the modules themselves acquired the following names: the expertise module, the student-model module, and the tutoring module. The expertise module in this structure is responsible for knowledge generation and verification that the student solution is correct, the student-model module is responsible for the timeliness of system knowledge of the student reflected in the student-model, and the tutoring module combines the functions of teaching support and student dialog interaction support.

In using a student model, the tutoring module selects the optimal teaching action for the given student and supports the student during the realization of actions. The results of student efforts with the selected action are reflected in the teaching history; the SM support module is also used to update the representation of student knowledge. To determine the present state of knowledge of a particular student, the support module utilizes the teaching history and its own knowledge base of students in general: a generic student model, which may include an ideal student model (in Anderson's sense [2]), knowledge of typical errors, a spectrum of personality traits, etc. Moreover, the support module may also utilize the capabilities of the expertise module of the subject domain (SD) to compare the behavior of the student to the behavior of an expert in the same situation. The new SM status is used by the tutoring module, etc. In general terms, this represents the main tutoring cycle in an ITS. In the cycle, the student model functions as a unique interface between the tutoring module and the student-model module.

The ITS structure proposed in [1] and the student models used in this structure were employed by the authors of a number of ITS surveys [3–5] and many applied articles. This structure is used as the basis for comparing different ITSs, and for identifying the problems faced by ITS designers in survey and applied ITS papers. The names of these modules may vary in different studies, and additional modules may be incorporated, although the popular principle of identifying modules based on the type of knowledge contained within has remained constant. Beginning in 1984, recent surveys [6–12] have identified an additional interface module in which knowledge of student interaction is concentrated (Fig. 1) within the ITS structure.

The subject of this survey is the student model, which has traditionally been considered one of the key components of an ITS. Student models have always been the focus of considerable attention in ITS research. The importance of student models in ITSs was identified in the earliest analytic papers on intelligent tutoring systems [13, 14]. The problems of designing student models were identified as among the three most important groups of problems in the field in the foreword to the first monograph on intelligent tutoring systems [15]. Beginning with the classic paper by Self [16], a number of specialized studies have been devoted to the problems of constructing student models. While not pretending to cover all problems connected with the student model, this review concentrates on methods and using the student model in ITS.

1. Types of Student Models and Their Classification

It was proposed [17] that student models could be classified by the nature and form of information contained in the models as well as the method of interpreting it. From our viewpoint, this proposal can serve as the basis for classifying student models. In this context it is only necessary to point out that the proposed measurement breaks down into three generally independent trends: three criteria for classifying student models in accordance with the nature, form, and method of interpreting information contained in the models.

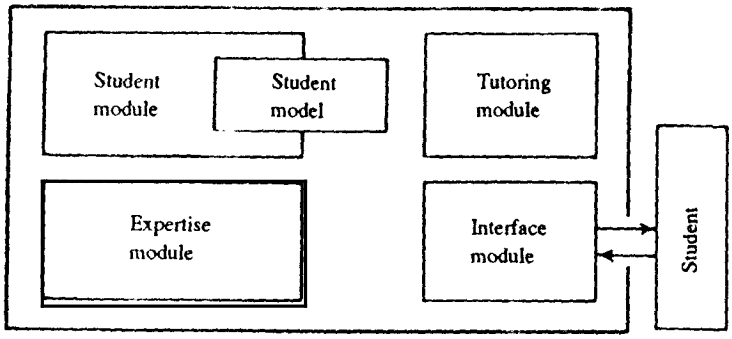

Fig. 1. Structure of an ideal intelligent tutoring system.

The basic criterion for classifying student models is undoubtedly the nature of the information contained in the model. From this viewpoint, all student models can be divided into two major groups [18]: models of course knowledge and models of individual, subject-independent characteristics. These two groups are appreciably different, both in terms of the form of representation of the model and the methods used in its construction and application.

Until recently, insufficient attention has been devoted to models of individual characteristics, and this field has been relatively neglected. During this period, the GRUNDY studies [19, 20] represented some of the most serious research in this field. Given the small number of studies on characteristic models, it is still difficult to make a reasonable classification of these models.

The student knowledge model (SKM) represents a reflection of the student's state and level of knowledge of course material (the subject and degree of knowledge by the student). The simplest form of SKM is a scalar model, which estimates the level of user knowledge of the course material by means of a certain integral estimate such as a number ranging from 1 to 5.

Unlike the scalar model, the overlay model makes it possible to display specifically what the student does and does not know. The overlay model assumes that all knowledge of course material can be divided into certain independent portions or elements. The simplest overlay model assigns a Boolean estimate, 1 or 0, to each element, indicating whether the student knows or does not knqw this element. In this case, student knowledge is represented at each instant of time as a subset of expert knowledge. It is for this reason that this type of model is called an overlay or a cover model. A more complex overlay model provides additional displayed information, indicating the degree to which the student knows such elements. A certain measure of knowledge of each element by the student is assigned to each such knowledge element. This can take the form of a scalar measure (an integer or probability measure) as well as a vector estimate.

Student errors or diving, which cannot be represented within the framework of an overlay model, require the use of bug or Buggy models, which make it possible to define and reflect the causes of incorrect student behavior. Perturbation models represent the most extensively studied form of error models. Perturbation models assume that one or several error elements and perturbations exist for each element of expert knowledge. Incorrect student behavior may, from the viewpoint of this approach, be caused by the systematic application of one of the perturbations of a correct knowledge element in place of the correct knowledge element.

Both an overlay SM and error models simulate the present state of student knowledge, without relying on the structure of expert knowledge. Overlay student models treat student knowledge as a weighted subset of expert knowledge, whereas error models make it possible to reflect erroneous deviations from expert knowledge. However, student knowledge may pass through several levels during the learning process, and on each such level such knowledge may be correct and represent a simplification or a particular case (yet not a subset) of expert knowledge. A type of model that makes it possible to reflect the development (genesis) of student knowledge from the simple to the complex and from the particular to the general has been proposed in a well-known paper [21]. A genetic graph whose nodes function as the elements of procedural knowledge (production rules) with relations representing the interaction between such nodes was proposed as a generic student model. The genetic graph can be treated as a development of a purely overlay student model and an error model. Such a student model provides a more exact reflection of the state of student knowledge.

Therefore, from the viewpoint of the nature of knowledge reflected by the student model, the most significant models are overlay student models (which reflect the possible state of knowledge), error models (reflecting possible errors and fusion), and genetic models (reflecting the possible genesis of student knowledge).

Student models can be divided into executable and nonexecutable models based on the method of interpreting the model [9, 17, 22]. A student model is executable if its present state can be utilized by a certain interpreter to simulate the behavior of the modeled student when the student is solving training problems.

Executable models are referred to as procedural models from the viewpoint of the nature of knowledge reflected. Any model reflecting procedural student knowledge can be made executable when a proper interpreter is available. The most popular form of executable models are purely procedural models whose knowledge elements are production rules. A classical reasoning machine, which is a component part of an expert module, is used as an interpreter. It should be pointed out that executability represents an independent measurement of the SM classification: executable models may be purely overlay models (GUIDON [46]), error models (LMS [30, 66]), and genetic models (WUSOR [21]).*

*The order in which the references are cited is determined by the order in which the methods and models are presented. The sources violate this sequence, however, for specific developed systems (Editor's note).

From the viewpoint of contents, elements of subject domain knowledge which form a generic student model may sent the knowledge of an expert or a novice, may be correct and erroneous, and may be both procedural and nonpro- al in character. From the viewpoint of the structure and form of the entire student model, such elements may be endent of one another (element vector) and may be related by a variety of relations. A generic model in which the ions between model elements are explicitly related is a network (the nodes are the knowledge elements and the arcs he relations), the structure of which reflects the structure of the subject domain under analysis. We shall refer to such els as structural-network models. The structuredness-nonstructuredness also represents an independent measurement M classification: structure models may be both overlay and genetic as well as executable and nonexecutable models. of the most developed structural-network models is the genetic graph noted above, which is an executable student el.

2, Methods of Constructing and Updating Student Models

Three groups of models can be classified in accordance with the information source used to update the student models maintain the model in a timely manner): student models updated based on an analysis of student actions, models directly lated as a result of a student action, and self-updating models. An example of a model that only updates upon student ion is an artificial student model [23]. An example of self-updating models are models that reflect the process of forgetting weakening of knowledge over time) [24, 25] and models simulating the shift of focus of student attention [26]. Several thods of student model updating are commonly used simultaneously in many systems [25–27].

The primary source of data for updating student models is the student himself. A purely citation-based classification ypes of information was proposed in [28] for use in constructing a student model:

—an implicit model based on observing the actions of a student while solving a problem;

—an explicit classification based on a direct dialogue between the system and the student;

—a structural classification based on the structure of interrelations between the elements of knowledge of the subject main under study;

—a background or historical classification [1] based on a common generalized estimate of the knowledge and ipabilities of the student and on his previous experience.

The third and fourth points require a brief explanation based on examples. A variety of relations between knowledge lements may serve as an example of structural information used in constructing student models. Many systems employ "precedence" relations, which denote that knowledge of one of the elements represents a condition for assimilating another nt. If the student has demonstrated knowledge of a certain element, we can conclude that this student also has some f knowledge of a preceding element. On the other hand, a lack of knowledge of a precursor indicates a likely lack ledge of skills deriving from this element. Another method of updating student models is used in, for example, the 40] and QUADBASE [72] systems. Another possible relation can be derived from the fact that a similar element more complex than another (BIP-2 [58] and MFE [17]). If mastery of a more complex skill has been demonstrated, e can conclude that a less complex skill has also been mastered.

An example of background information is the learning factor: a certain accumulated integral estimate of the rate of student progress through a course. If a student who has solved a problem has demonstrated a mastery of a certain element of knowledge, the measure of learning of this element in the student model is elevated in proportion to the learning factor. Thus, the greater the learning factor, the fewer problems that must be solved to achieve the final training goal [24, 27]. The so-called principal mean plays an analogous role in the GCAI system.

Structural and background information only represent auxiliary information and are not a commonly used source for updating a student model. The main source for updating a student model is explicit and implicit information derived from the responses of the student to questions generated by the system, results from problem solving, and observation of the problem-solving procedure employed by the student and the student's operation within the system, which can be derived from an analysis of the student's responses and actions. The procedure for this approach will be described in detail below. Here we briefly note two additional non-traditional methods of updating student models. First, the student can convey to the system his own assessment of his level of knowledge in explicit form. Rich [20] believes that this is not a typical method used in student models, although it can be successfully employed in, for example, the BIP system [57] as an auxiliary method. Second, one or several special tests can be given to the student prior to the beginning of the training course. This

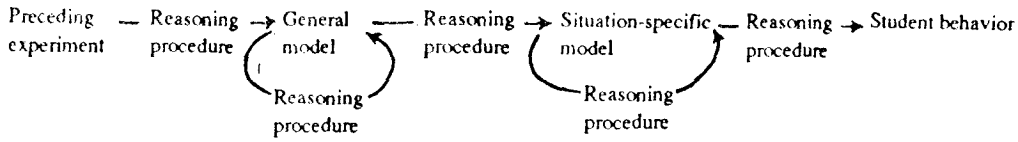

Fig. 2. Classification of information recovered.

method can be used to identify the student's preceding experience and to determine a number of his individual characteristics.

The volume and accuracy of information that the ITS can derive from student responses and from observing student activities will largely depend on the type and complexity of the student model in use. The more advanced the student model used in the ITS the deeper the diagnostics of student knowledge achieved by the system. Clancey [22] proposed a classification of potential information derived from the degree of diagnostic depth. This classification is based on the division of student models employed in this paper into general models, a reasoning procedure (RP) and a situation-specific model (Fig. 2). From Clancey's viewpoint, the general student model is comprised of the student's knowledge of the subject domain employed in any situation to solve any problem. By applying this knowledge in a particular problem, the student constructs a model of this problem (a situation-specific model). An example of such a model is the proof of a specific theorem in mathematics, or diagnosis of a specific patient in medicine.

Thus, based on the student's answers and an analysis of his actions, the intelligent tutoring system can perform the following:

1a) Determine the accuracy of student statements on the problem, i.e., the student's understanding of the problem.

1b) Determine the accuracy of the student's statement on the general model, i.e., essentially his knowledge of primary facts in the subject domain under analysis. This form of information, as a rule, will be represented in explicit form in the student's response to system questions from which it can be easily recovered.

2) Test the consistency of the solution given by the student (the situation-specific models) in light of data on the problem and knowledge of the general model known to the student. For diagnostic training systems, for example, this would mean consistency between the hypothesis of the illness (injury) with symptoms known to the student (SOPHIE, GUIDON) [46, 77]. For systems teaching how to solve the theorems, this includes the correct formulation of each step of the proof (Geometry Tutor) [65]. For program training systems this will include the syntactical and linguistic accuracy of the program solution.

3) Define which elements of knowledge were employed by the student in formulating a solution or hypothesis. There are several technologies for using such information to update an overlay student model. The most well-known technology of differential modeling will be described below.

4) Determine which errors in the general model (of confusion) have been responsible for the observed incorrect student behavior.

5) Determine which errors or nonoptimal reasoning could give rise to the observed incorrect student behavior. There are several technologies for identifying errors and confusion in factographic and procedural knowledge. An overlay model is used in many cases here, although this model only makes it possible to determine one type of error: the absence of knowledge elements. Therefore, a conceptual or procedural error model is required for qualitative diagnostics on this level.

6) Determine the cause of confusion in the general model or the error in the reasoning procedure. Such a diagnostic level requires a knowledge of a certain error production mechanism in the given subject domain.

Methods of student knowledge diagnostics can be classified as a function of the information source employed for the diagnostics. There are two information sources (explicit and implicit [28]): the student's responses to system questions and problems, and the student's behavior. This system can be used to identify methods of analyzing responses and student actions during problem solving. The latter group of methods essentially breaks down into two subgroups depending on whether or not the system knows which problem the student is solving. Finally, methods of analyzing the student's solution programs for program instruction can be classified as a separate group.

Methods of analyzing student responses. Here, student responses refer both to responses to the system's questions and to the results of tutoring problems solved by the student. It is possible to identify two essentially different types of system problems and tasks. Responses to simple questions require from the student mastery of precisely a single knowledge element in the subject domain: a fact, a rule, or a fragment of the course. A response to complex questions requires that the student possess several knowledge elements in the subject domain.

In analyzing the responses to simple questions, it is, as a rule, sufficient to rate the answer as correct or incorrect. In the case of a correct answer, it is possible, for example, to increase the measure of learning of the knowledge element whose mastery has been demonstrated by the student, while in the case of an incorrect answer, to reduce this measure. This method was used in the SCHOLAR [74], MALT [68], GCAI [62], and QUADBASE [72] systems as the basic method and as an auxiliary method in a number of other systems. If the system employs a perturbation error model, we can easily determine from the incorrect response on the part of the student which perturbation of the tested element of knowledge was responsible for the error.

An analysis of responses to questions and problems requiring mastery of several elements of knowledge is much more complex. If, in the case of a correct answer, the system can suggest that the student has demonstrated mastery of all elements of knowledge associated with this problem (and the weights of these elements in the student model have been increased correspondingly), then in the case of an incorrect response, it remains unclear as to the lack of knowledge of which of these elements is responsible for the incorrect response. Formally speaking, all elements required to solve the problem are under suspicion. However, reducing the weights of all these elements in the student model, as is carried out in the BIP system, [57] introduces strong noise into the student model. One method of reducing such noise is to use the structural and background information noted above. For example, knowledge of the relations between elements of the subject domain and preceding experience reflected in the overlay student model make it possible for the BIP-II [58], WISOR [21], and ISOP [27] systems to identify fairly accurately, from a large set of suspicious elements, a much narrower subset, knowledge of which is likely to be responsible for the observed error.

The most effective diagnostics of responses to complex questions can be obtained if the systems use an executable student model (an accessible error model and a simple overlay student model are desirable). In this case, the diagnostic module can, by various methods, select or derive a set of true or erroneous SD elements that upon interpretation yield a response that is identical to the student's response. The resulting set is identified by the present student model.

The DEBUGGY [61], LMS [30], PIXIE [70], and CPT [60] systems represent examples of the substantially different [29] response-based reasoning models. The main problem in deriving or selecting a model is that the response provided to the system can be obtained by a variety of methods, which produces an enormous search space for possible models. Heuristics (DEBUGGY) or special methods of reducing the size of the search can be used to deal with the combinatorial explosion [30].

It is generally possible to avoid problems of searching for possible student models if the system employs a method of deriving a student model from a response. A special debugging technology borrowed from artificial intelligence is used in the CRT system to derive a student model that is a Prolog program.

It should be pointed out that these methods (LMS and CRT) require a knowledge of several student responses to several teaching problems to derive the student model accurately. This is not always convenient in practical systems, the more so as the student's knowledge may change from one problem to another. The most successful systems of this type can be used as purely diagnostic systems [12].

Analysis of the process of solving of a known problem. Observing the student's progress in solving a problem provides the system with additional information on the student's knowledge and makes it possible to avoid a complex search or conjecture. Expert systems have the most advanced methods of observing problem solving. Procedural knowledge on problem solving in the subject domain under analysis together with a subject domain expert who is able to solve any of the problems presented to the student by utilizing such knowledge provide the basis of such systems. An example of such a system is the GUIDON system, which trains medical students in diagnoses. The GUIDON system expert, who knows the problem presented to the student, can solve the problem by observing the problem-solving process used by the student. At each step of the solution, the system knows all elements of procedural knowledge (rules) used at that instant, which includes all permissible correct actions on the part of the student. Hence it is possible to determine on the basis of the student's actions which rules he is using and to introduce the corresponding changes into the overlay student model.

The well-known model tracing technology first introduced in the Lisp Tutor system [2] represents an advancement of the technology proposed in the GUIDON system [46] to the case of an error model. The Lisp Tutor system contains all

possible correct rules that can be used by the student in solving the problem (an ideal student model) together with a catalogue of possible erroneous rules derived from an analysis of protocols. By solving the problem with the student, the system knows all possible correct and incorrect rules used in each solution step. This makes it possible to determine a specific correct or incorrect rule used by a student in the current step of the decision-making process based on the next action by the student. After determining that an erroneous rule served as the basis of the action, the system recognizes that the action itself is erroneous and explains the error to the student by means of additional information from the error catalogue.

The model tracing technology has recently been used in expert intelligent tutoring systems. One drawback of this diagnostic method is a certain degree of limitation on the students' freedom during the solution process. In order for the system to understand the student, the student must use only those actions that are included in the set of correct and erroneous rules of the system in each learning step. In certain subject domains, this limit is not convenient for a student (which is the reason for the critique of the Lisp Tutor system [10]), whereas in other subject domains this represents a reasonable limitation by forcing the student to follow a correct solution procedure (FisioDisk) [52].

Analysis of student actions. The most complex process is the analysis of the student's actions, if the problem being solved by the student is not known to the system.

A particular case of such a situation is the game situation where the student, generally speaking, is not solving a problem but rather is playing a certain collective (WEST) [39] or individual (WUSOR) game [21]. The game context assumes that the purpose of the student player is to gain an advantage, and hence each step in the game can be treated as a certain mini-problem, the response to which is the next move by the student. The game situation can be reduced to one of the preceding situations when the system contains an expert player who, at each step of the game, is capable of choosing the optimal move and, possibly, several permissible, yet nonoptimal moves. In this case, if the knowledge required by the player is commonly procedural knowledge, model tracing technology can be used. Such an approach has been applied in the WUSOR [21] system. One characteristic feature of this system is the use of a genetic knowledge model, i.e., the reflection of possible player knowledge in their early development stages. In following the student's progress, the expert uses knowledge on the same level (plateau) of the model achieved by the human player. This permits a more accurate representation of the diagnosis and also makes it possible to provide advice that is applicable to the level of knowledge of the student.

The interesting "differential modelling" method commonly cited was developed based on a game contest in the WEST system [30]. Each move in the game which this system teaches requires knowledge of several elements in the subject domain. An expert module in the subject domain (the game expert) and a special recognition module, which is capable of determining knowledge of which elements is required in any given move to make this move, are employed for modeling. After the student has made the next move, the expert proposes its move, which is the optimal move for the given situation. The recognition module identifies the set of knowledge elements required to make the moves proposed by the student and by the expert. If the moves and the set coincide, it is assumed that the student has demonstrated knowledge of all elements in the set. If the moves and sets are different, a subset of knowledge elements present in the expert set and absent in the student's set is identified. These elements come under suspicion: It is possible that a lack of knowledge or incomplete knowledge of such sets have impeded the student from making a better move.

This technology is comparatively simple and effective and is used in a number of practical intelligent tutoring systems. An overlay student model in differential modeling systems will, as a rule, contain three counters for each knowledge element; these counters will indicate the number of times the student has appropriately or inappropriately employed the given knowledge element and the number of times the element was not used in an appropriate situation (WEST, GTT) [39, 64] respectively. During differential modeling, the student model accumulates statistics that can be used directly by the tutoring module.

It should be noted that the game context is not a standard context for intelligent tutoring systems, but rather serves as more of a proving ground to test new ideas such as the genetic graph or differential modeling. In the general case of analyzing a student's actions, the context cannot predict which problem the student will solve, and hence expert technology cannot be directly used.

Without knowing the problem to be solved, the system cannot evaluate the actions of the student as incorrect or inexperienced. However, the system can trace all student actions and define in the student model the actual use of such actions or knowledge elements required to perform such actions. In a number of subject domains it is also possible to identify in the course of observing operation, formal errors that will be independent of the problem solved (the analog of syntactical programming errors) and to take these into account in the model (ReGIS, BETS) [40, 6].

To carry out a conceptual yet informal analysis of student actions, the system must recognize the goal and plan of student operations, i.e., understand what the student wants to do and how he plans to do it. Plan recognition is a comparatively new field of research of ITS designers. The first ITS, entitled Macsyma Advisor [69], is capable of student goal and plan recognition and initiates diagnostics on this basis; this system was developed in the early 1980s. Serious efforts in this field expanded in the second half of the 1980s [31, 32] in connection with the development of so-called intelligent help systems [33]. Such systems are designed to provide help to applied system users. Since the user employs any applied system to solve his own specific problem rather than for teaching (i.e., the problem for which the system was built), standard training actions, problems, and tasks are not permitted here. The only possible method of operation is to recognize which problem is being solved by the user and to explain to the user at the appropriate instant such elements as user errors, nonoptimal actions, to offer help, to inform the user of insufficient facts, etc. [34].

For the help system to have sufficient plan recognition capability, it requires knowledge of possible problems to be solved in the given subject domain as well as possible correct and incorrect execution plans. If such knowledge is sufficiently comprehensive, the help system in most cases will succeed in recognizing the student's goal and plan, will compare the student's actions to one of the existing plans, and on this basis can initiate diagnostics and help. A shell was developed within the framework of the well-known EUROHELP project [34] to develop intelligent help systems that can be refined for the required subject domain by incorporating knowledge as noted above [35].

Program analysis methods. A special field of diagnostics is program analysis: analysis of the solutions of teaching problems for programming teaching applications. From the viewpoint of classifying output information, such a program constitutes a response to a problem. However each program essentially is a coded sequence of actions. Hence methods similar to those used to analyze student actions are employed for program analysis [12].

It is sufficient to use the programming language constructions (ISOP) as the subject domain elements for a formal analysis of programs [27]. In this case, the correct utilization of the construction constitutes proof of knowledge of the corresponding element, while errors (syntactical or linguistic) are treated as incorrect knowledge. If the system stores a reference solution for each problem, differential modeling technology can be used for more precise diagnostics.

For a deeper analysis of the program, the system must know which possible path (plan) the student has chosen for his program. Without knowledge of this plan, a conceptual analysis is virtually impossible. One possible approach more (MALT) or less (Lisp Tutor) [67] rigidly limits the student's freedom in choosing a plan to solve the problem by holding the student within the framework of one or a few plans known to the system. This approach is simpler, although it is commonly criticized, since it essentially converts programming training into coding training. In this regard, considerable attention has recently been focused on program plan recognition.

The recognition of the underlying plan of a program is largely similar to the recognition of plans of actions examined above. Research on program plan recognition began as early as the 1970s in the field of artificial intelligence. The PROUST system [71], which is capable of program plan recognition, is the most well known system. This system utilizes a knowledge base of possible plans for solving problems and subproblems, together with a complex goal-plan-subgoal network. All tasks to be handled have a special description that relates these tasks to the goals of the knowledge base. In the absolute majority of cases the system is capable of identifying the most likely plan for solving a problem. By comparing this plan to the plan obtained, PROUST can reasonably accurately diagnose an entire series of fragment miss or operator transposition errors. A special error catalogue in which the underlying confusion associated with the observed error is listed, is employed to diagnose student confusion.

Program analysis constitutes an active subfield of research in intelligent tutoring systems. New ITSs with such capabilities have appeared recently. A more detailed discussion of program analysis methods and certain systems in this field can be found in [12, 36].

3. Methods and Technologies for Using Student Models

Consistent with the traditional view of student models as an interface between the student model module and the tutor module, the actual status of the student model is used by the tutor module to individualize the teaching process supported by the tutor. When engaged in the tutoring process, the tutor module can execute a wide range of different functions performed by a human tutor in traditional training, and the student model can be used for adaptive execution of each of these functions. One excellent and extensively cited classification of tutor-module functions and methods of employing student models for

their adaptive execution has been proposed by Self [37]. According to this classification, all these functions can be divided into six main types:

1) student error and confusion remediation functions;

2) student knowledge development functions that may at present be correctly, yet incompletely, understood;

3) strategic functions that revise the tutor-module strategy itself when necessary;

4) diagnostic functions or functions to achieve an exact determination of the student's state of knowledge;

5) prediction functions, i.e., determining the probable response of the student to a teaching action;

6) functions of assessing the performance of the student, the ITS itself.

It is this functional classification that is used in this section to describe the student model methods and application technologies; only the sequence in which these groups of functions are analyzed has been modified.

3.1. **Knowledge development.** The development of a student's knowledge in the subject domain under study represents one of the main functions of many existing ITSs. The knowledge development process begins when the tutoring module defines the knowledge picture of the student as incomplete, and it consists of four stages:

1) a definition of precisely what knowledge the student lacks (what to teach);

2) the choice of an appropriate moment for knowledge development (when to teach);

3) the choice of a method of knowledge development: a teaching action (how to teach);

4) the execution of the chosen teaching action.

The student model can be used effectively at each of these stages.

Stage: *What to teach?* Three ITS operating modes can be identified in accordance with the method of determining what knowledge the student lacks.

1) Goal-oriented tutoring. In this mode, the system has a certain tutoring goal: a subset of knowledge on the subject domain which must be taught to a particular student. The tutoring module can employ an overlay student model at each stage of goal-oriented tutoring to determine which goal knowledge elements have already been studied and to what degree, and which elements remain relatively or completely neglected. Then, the subset that will be studied in the present stage is chosen by a certain strategy from the entire set of relatively neglected knowledge elements. In systems employing a vector (unstructured) student model, the next test subset is chosen randomly or in accordance with a preestablished training procedure (BIP, ASOD) [57, 80]. If the system has a structural-network student model, the relations between the knowledge elements can establish a reasonable choice of the subset to be studied. For example, elements with unstudied predecessors (BIP-II [58], ISOP [27], Assembler Tutor [56], ReGIS [40], and QUADBASE [72]) are automatically eliminated from this subset. On the other hand, the elements associated with subsets already studied are inserted first. Thus a front of studied knowledge elements essentially propagates through a network student model [21].

2) Active help. In this mode, the system observes the actions of the student solving a certain problem. By identifying the incorrect or nonoptimal behavior of the student, the system attempts to determine the elements, a lack of knowledge of which would result in such behavior of the student. It is these critical elements that are tagged by the student model as unknown or requiring development. The set of critical elements are determined by student behavior using one of the three techniques discussed in Section 2 such as differential modeling.

3) Passive help or the question and answer mode. In this mode, the student explicitly requests information needed from the system. The question can be selected from a menu of possible questions [38] or formulated in a natural language (NL) subset. In the first case, the subset of required knowledge elements must be simply related to the question, whereas in the second case, it is necessary to analyze the NL question to derive this subset. A special generic student model expanded to include natural language words and phrases is required for such an analysis, thereby considerably simplifying the analysis procedure. The majority of natural language analysis methods were developed in the early stages of ITS (SCHOLAR, SOPHIE) [44, 77] and recently less attention has been devoted to natural language dialogue in ITSs, obviously due to the development of new types of student interfaces.

It is interesting to note that the absolute majority of ITSs support one or two of these modes. Essentially the type of ITS and the entire training process will determine which of the knowledge development modes is the primary mode. Systems that support a goal-oriented training mode are usually called "teachers" or "tutors" and the sequence of knowledge studied in this case will be determined by the generic student model and the tutoring goal. Systems supporting the active help mode in solving a problem known to the system are usually called "coaches" (this term was first introduced in [39]). Systems that support the active help mode during independent student operation are called intelligent help systems [33]. As a rule, the question and answer mode constitutes a secondary mode of an ITS and can be combined with all modes described above.

There are ITSs in which the question and answer mode is the main knowledge development mode, although such systems are more accurately called questioning systems rather than training systems.

Stage: *When to teach?* The choice of an appropriate moment for student knowledge development is not a mandatory stage. This stage is important only to active help systems in which student knowledge development requires interruption of the student solution process at a certain instant. Nonetheless, the "when to teach" problem has traditionally been considered a very important problem in ITSs. The simplest approach is to interrupt the student immediately as soon as the error or nonoptimality is identified to indicate the cause or the knowledge not available to the student. This approach has been justified as the most psychologically valid approach in [2]. The opposing approach—the tutoring paradigm—has been substantiated in [39]. Here, errors and nonoptimalities are allowed and are summed in the student model until an error is made in a relevant context. Only then, at the most appropriate moment, will the student learn of incorrect or nonoptimal behavior and be informed of insufficient knowledge. In this case, the student model serves as an error accumulator, while the systems tests at each step whether or not the situation is suitable for disclosure of accumulated errors.

Stage: *How to teach?* The choice of the teaching action (TA) itself may generally precede the choice of the type of teaching action. The choice of type is usually made in the few systems where there are several different types of actions: explanations, tests, examples, and problems (ReGIS, ISOP) [40, 27]. A characteristic model [18, 40] is, as a rule, used at this stage.

In the TA selection stage, the system must generate or select from a special database the teaching action of required complexity that covers the elements of knowledge from the critical set defined in the first stage. In simple subject domains, where each teaching action is precisely equated with a single knowledge element, the transition from an element set to teaching actions presents no problems and does not require a student model (GSAI [62], SCHOLAR [74], QUADBASE [72]). Special student-model based technologies (MALT, BIP, BIP-II, ReGIS, ISOP) are used in complex subject domains (such as in programming) in which the teaching action (example or problem) is related to several knowledge elements to select or generate teaching actions. For example, in the MALT system, the next problem is chosen from elementary subproblems that in this case represent knowledge elements, with the probability of the subproblem-element being incorporated into the generated problem inversely proportional to the probability of mastering this element as reflected by the student model. In the ISOP system, all teaching actions stored in the TA bank are related to the generic student model by means of spectra. The SM spectrum is simply a list of knowledge elements used. A special strategy utilizing TA spectra and the state of the student model [41] is employed to choose a teaching action of optimal complexity for the critical set of concepts.

A free-style tutoring mode is an alternative mode used in a number of systems; in this case, the student himself determines what to do next. In the case of free-style tutoring, Stages 1 and/or 3 are executed by the student himself, where the student can define as the next elements either the studied course elements or the next teaching actions. However, in this case, a student model will be useful: It can be used, for example, to construct a list of knowledge elements available for study or permissible reasoning procedures (ISOP) [27]. The relative difficulty of a teaching action calculated by means of a student model may even be indicated in the menu of permissible teaching actions to provide further orientation for students.

Stage: *Implementation of teaching action.* In the majority of systems, teaching actions are implemented without a student model. However, certain ITS examples have demonstrated that a student model can be used successfully in this stage as well. Thus, the degree of detail of an explanation of a certain concept is defined by means of a student model in the QUADBASE and ISOP systems; the more the concept is studied, the less the detail provided in the explanation. In the same manner as in the GSAI system, the degree to which a concept has been studied will determine the degree of detail in the interaction occurring during the solution of the problem relating to this concept. The MALT system and the system discussed in [42] make it possible to accelerate the decision-making process by automatically performing those steps in the solution that are mastered by the student according to the student model.

In concluding this section it should be noted that overlay student models are quite sufficient for student knowledge development. In this case, it would be desirable to employ a network form of representation of a generic student model, since the relations between knowledge elements may be of considerable assistance in the different stages of knowledge development.

3.2. **Error remediation.** Error and confusion remediation functions occupy first place is Self's classification [37]. Consistent with the most common understanding of an ITS in use today as a system that observes student problem solving and identifies student errors and confusion, the remediation function is a primary function in an ITS (see Section 4.1). It is not surprising that a number of error remediation methods have been substantiated to date. Self has identified eight error

remediation methods: error definition, explicit remediation, implicit remediation or prompting, counter examples, demonstration of a solution method, access to previous experience, repeated attempt, tactical retreat.

A system requires an error model to perform the majority of these functions.

Error definition involves generating a word description of an error and additional explanations and recommendations to correct the error for the student. As a rule, an ITS will use an error and confusion catalogue for this purpose; this catalogue represents a component part of the error model. In the simplest case (the MALT) the catalogue contains, in addition to each error, an NL message which is also displayed when an error is identified. In the more complex case (PROUST and Lisp Tutor) the catalogue will only store message patterns that are adjusted prior to generation for the context in which the error occurred. Self considered as an ideal case message generation directly from the error rule. Such systems as GIL [63] appeared after Self's paper [37] was written.

Explicit remediation involves explicit presentation to the student of knowledge or skills the absence of which was responsible for the observed error (this version was considered in the preceding section), or the system's execution of a step in the solution process which the student could not correctly perform. Most commonly, explicit remediation is employed when the system is capable of identifying the actual error but is not capable of classifying this error (the error is not included in the catalogue or there is no error catalogue). For example, the Lisp Tutor system, after a student twice performs actions that are not described in the generic student model, will itself take the next step for the student, thereby avoiding confusion on the part of the student.

Implicit remediation, unlike explicit remediation, only provides a student in difficulty with prompts of correct knowledge or actions. It is even possible to set up several levels of prompts with increasing particularity in such ITSs as a proof training system (UPSM [78] and Geometry Tutor).

A counter example is a system-generated situation or problem in which the erroneous knowledge of the student will lead to an explicitly erroneous result from the viewpoint of the student himself [43]. A counter example is the most complex yet effective method of confusion remediation. The WHY system [79] is an example of a system using this method.

One possible remediation method is to show the student the solution path generated by the system based on the student response using the DEBUGGY or the LMS method.

Occasionally, accessing previous experience stored in the student model may help the system to define the cause of error and may help the student correct the error. Thus, when errors occur in work utilizing knowledge previously used, appellation to problems specially selected by the student model from problems solved successfully may assist a student deal with difficulties.

A repeated attempt involves presenting the student with a similar problem for solution. From the viewpoint of the student model, the new problem must be similar to the old problem in terms of its constituent knowledge elements. Such a problem can be chosen or generated by one of the methods described in Section 3.1 using the student model.

By using the student model, the system may temporarily retreat from the domain causing confusion and may temporarily exclude from the spectra of proposed teaching actions those difficult knowledge elements and insert in their place elements preceding the difficult elements in the student model for their reinforcement.

Knowledge diagnostics. Knowledge diagnostics refers to the maximum possible accurate definition of the state of knowledge. In the majority of cases, such a problem arises in a diagnostic ITS for which deep diagnostics were used [12]. For example, assume that the system knows that the student is utilizing knowledge elements *P1* or *P2*, but not precisely understood which element is used. If we are sure that the student is utilizing only one of these elements, the student model can be used to generate a test problem (DEBUGGY, LMS) or to generate additional questions (CRT) enable a precise determination of the element utilized to be made. If, due to weak knowledge, the student uses first one and then the other element, the student model can be used to choose an appropriate example that demonstrates to the student both of the elements must be used.

3.4. **Strategic functions.** The tutoring module functions examined above should be treated as tactical functions. However, tactical actions to choose the next teaching action can only represent part of a more global plan or teaching strategy in systems employing goal-oriented training. The strategic functions of the tutoring module may include plan modification or strategy functions if they are ineffective for this student.

Very few existing ITSs have the ability to modify a teaching plan or strategy in the full sense of this word. Good yet fully realized proposals in this direction can be found in [6, 44, 45]. There is, however, a whole series of systems that support several student operating modes (styles) and are capable of modifying this state by adapting to a specific student. The student model can be used as the basis for determining the need to modify a mode or to select a new ITS. In this

case, the change in interaction style is commonly dictated by the data accumulated by the student model on the general lack of success of the student in an old mode (the threshold number of unsuccessful attempts has been exceeded; the mean assessment is low), and the choice of a new style is most commonly determined by a model of individual characteristics. Strategic knowledge relating to the choice of a training state is usually represented in a tutoring module by means of production rules, with the predicate of the state of the knowledge and characteristic models appearing on the left side of the rules and the corresponding training mode given on the right side [6, 18, 40, 46].

3.5. **Prediction and assessment functions.** Self divides prediction and assessment functions into separate groups of functions. Prediction functions are divided into prediction of the student's behavior and prediction of the student's learning path. Assessment functions are divided into assessment of the student and an assessment of the ITS itself. It is sufficient to have an executable student model to predict student behavior during problem solving. An "artificial student"-type model that simulates student acquisition of knowledge is needed to predict the course of study. This model can be used, for example, by the tutoring module to select an optimal teaching action and to attempt to try it out initially on an artificial student.

The student assessment function, which was once a very important function in the paradigm of programmable teaching, has lost its role in ITSs. Nonetheless, by using a student model, an ITS can precisely assess the student performance on any given scale, and this is in fact performed on several systems. Moreover, the ITS makes it possible to realize complex testing strategies (TEST) [45] that take into account the percentage of goal success by the student, the average level of a group of students, and even their personality traits. An "artificial student" model can be used to evaluate the operation of the entire tutoring system. For example, it is possible to compare the effectiveness of two ITS versions assigning identical "artificial students" to these versions and to compare the results obtained over an identical time [37].

4. The Old and New Paradigms for Using the Student Model

No ITS has yet been developed that uses the entire spectrum of methods of student model construction and application. As a rule, existing student models were developed within the framework of a specific paradigm, a specific view of the role and functions of an ITS in the tutoring process. Each such paradigm has its own specific approaches to constructing and using a student model.

4.1. **The expert paradigm.** The expert paradigm is probably the most popular paradigm developed to date for constructing an ITS. According to this paradigm, the primary function of an ITS is to support student solutions to problems. The role of the ITS in this process is student error and confusion diagnostics, i.e., checking student solutions, identifying errors that occur during the solution process, and identifying confusion underlying errors and their causes. The authors of ITS [8] considered the identification of the causes of errors to be their primary goal. It is assumed that diagnostic results are used for error and confusion remediation by employing one of the methods in Section 3.2. The role of the ITS within the framework of this paradigm can be compared to the role of a private tutor whose purpose is to train the student to solve problems in a specific subject domain correctly.

The most successful subject domain for such a paradigm is a rather formalized subject domain in which the solution to a problem is achieved by a sequence of formal operations. The two most popular subject domains can be identified as an example: column subtraction and elementary algebra. The knowledge required for problem solving can be represented as a set of "condition-action" rules (a knowledge base) and erroneous knowledge can be represented by an analogous set of "false rules." A procedural expert in the subject domain–a reasoning machine–is the basis of the problem-solving system in such a subject domain; this machine, by interpreting the knowledge base, is itself capable of solving any of the problems in the specific subject domain. A subject domain expert can be used to solve any of the problems in the given subject domain. The subject domain expert makes it possible to set up a demonstration of sample solutions to problems and to help a student to solve a problem. The expert may propose a prompt, the next step, or can complete a solution started by the student.

An optimal student model in such domains is a procedural error model in which the knowledge elements are true and false rules. In this case, precise student knowledge diagnostics will involve determining the set of rules and false rules that the student uses in solving problems. This set is called the present student model. It is derived from the observed behavior of the student and/or the student-generated results.

This expert paradigm was developed by the early 1980s under the influence of expert systems research. The main ideas and methods within the framework of this paradigm were developed in creating such help systems as SOPHIE I-III

Subject domain paradigms

Features	Expert paradigm	New paradigm
Method of construction	Conjectures; reconstruction of missed steps; set of rules and false rules	Provide the student with an environment in which he would use the computer in a manner such that all his actions, goals, etc., were explicit
View of a) the student model, b) its accuracy	The most accurate reflection of the state of student knowledge & confusion; the more details the better	More a reflection of what the student believes. Accuracy: no greater than what is usable
The student model and the student	Student model hidden (latent)	Student model open
Application	Mainly student error and confusion remediation	Multiple use methods
Common view	System: teacher, diagnostician, nothing greater	The system: assistant, colleague

[77], WEST, and Guidon WUSOR, and such knowledge diagnostic systems as DEBUGGY and LMS. Such research comprised most of the contents of [15] and became a commonly used handbook of ITS designers for over a decade, giving a new name to this field (before the term ITS, the term ACAI, intelligent computer-assisted instruction, was used). It is not surprising that many researchers in the ITS field have come to believe that this paradigm is the only possible paradigm, and the procedural error model is the ideal form of student model to which one must aspire.

4.2. **Critique of the expert paradigm.** Subsequent research to advance the expert paradigm and attempts to employ this paradigm in new subject domains began to reveal its obvious drawbacks and limitations and the underlying procedural error model. For example, it was determined that it is not possible to construct a good error model in less formalized subject domains, and that it is generally not clear for a number of subject domains how to construct such a model; for example, what to classify as rules and false rules [4]. The process of constructing a fairly comprehensive error model in formalized subject domains became very difficult since compiling the error catalogue requires analyzing a large number of problem-solving protocols. Even for such a simple subject domain as column subtraction, a complete error model was constructed only after several man-years of work. In this case, the more advanced the error catalogue, the more complex the process of deriving a model of a specific student in the enormous space of possible models [30].

Finally, let us assume that a good catalogue has been compiled and the system can derive from observed actions an exact student model, student errors, and the causes of such errors. It became clear that there are not many methods of easily using the derived information [47, 48] and these are principally error remediation methods. The majority of such methods of using the model do not require such information (Section 3).

The psychological validity (objectivity) of the error model is doubtful. Special research [47] has shown that the possible sets of false rules may be appreciably different for different groups of students; moreover, it is difficult to construct a stable set of false rules even for the same student. Certain doubts were also expressed on the pedagogical validity of the error model. The concept that the student problem-solving model underlying the paradigm reflects the formal approach to problem solving under which the student manipulates symbols in problem solving without understanding the meaning of the transformations has been substantiated in [48]. It is this fact that is responsible for the multitude of confusing errors for which false rules are used for representation in the error models.

Hence, from the viewpoint of the critics of the present paradigm [47, 48], error models have turned out to be complex, psychologically indefensible, and, moreover, of little use to ITSs. It is not surprising that recently many researchers, who have grown disillusioned with the existing paradigm and error models, have generally doubted that student models are required for ITSs [49]. At the same time, this has made it possible to avoid the difficulty of constructing a student model. The following are the most important proposals.

1. The student model should not be derived from the final student answer, but rather user-friendly interfaces for problem solving should be constructed such that the student can use such interfaces in explicit form to inform the system of his intermediate problem-solving steps.

2. The accuracy of student diagnostics cannot exceed the level usable by the ITS, since diagnostics does not represent a goal in itself, but rather a means of individualizing tutoring.

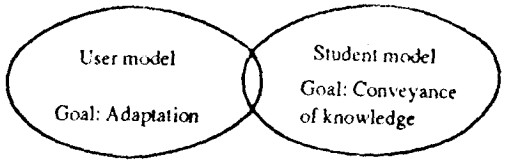

Fig. 3. Goals of user and student models.

3. It is necessary to avoid viewing student models only as a means of confusion correction: this represents only one area of application of SMs (Section 3).

4. It is necessary to avoid viewing ITSs as an infallible tutor that knows everything about the student and provides absolutely correct advice. It is more accurate to treat the ITS as an assistant or tool in the training process.

The proposals collected in Self's study [47] essentially represent the main characteristics of the new ITS paradigm. This paradigm began to appear in the late 1980s and has gradually replaced the expert paradigm. The main differences between the new and the old paradigm are listed in the table. The new paradigm enables the traditional problems of student model design to be overcome, thereby increasing the total utility of the student model. The role of assistant does not require tutor infallibility, which makes it possible to reduce the accuracy requirements on the SM and to transfer the main attention to achieving model utility. In particular, a system-assistant can make the contents of the student model accessible to the student and can even convert the SM into one of his tools. One of the most important tasks within the framework of the new paradigm has been to expand the number of ITS functions supported by the student model.

An analysis of a number of new, yet well-known ITSs (BRIDGE [59] RBT, GTT, SMITHTOWN [76], and SHERLOCK [75] enable the main ITSs characteristics to be represented within the framework of the new paradigm. Such an ITS mainly provides the student with a user-friendly environment for independent operation and experiments in the subject domain. The pedagogical (teaching) component within the framework of this environment can support goal-oriented teaching of a part of a course, generation of teaching problems, and problem-solving support, as well as knowledge diagnostics. It is significant that the director in the system is more likely to be the student than the tutor. Tutoring and the problems themselves are not related to one another, but rather are presented to the student. As a rule, the student himself will request the next element of the training process or the next training problem from the system. The most important function of the new type of ITS is to support independent investigations of the student within the environment. The ITS must address the "wandering" of the student, provide him with help, and develop the student's knowledge and skill both within the subject under study [34] and in independent research (SMITHTOWN). The student model must support this entire range of ITS functions.

5. The Student and User Models

In attempting to identify methods of increasing the utility of the student model in an ITS, it is necessary to focus on the experience of a neighboring field: the construction of user models in interactive systems. So-called adaptive interactive systems, i.e., systems that attempt to adapt to the new level of knowledge and the capabilities of a specific user, have a need for user models [17, 50, 51].

5.1. **User models and adaptation in interactive systems.** Generally, any interactive system requires user adaptation to maintain a successful interaction. As noted in [51] the human operator and the interactive system form a closed human-computer system the successful operation of which requires mutual adaptation of parts of this system. The majority of interactive systems do not have any adaptation capabilities and hence the human is forced to adapt to the system. However, the capabilities of human adaptation are not unlimited. As special research has shown [52], different human capabilities resist change to different degrees. It is easiest for a human to modify his current knowledge of a system by examining its capabilities. It is much more difficult, for example, for him to change his cognitive style of operation. Finally, it is nearly impossible to change such personality factors as extraversion-introversion.

The incapacity of a certain user to adapt to a rigid operating style of the interactive system reduces its operating efficiency and may lead to a breakdown of interaction with the system. In order to avoid this outcome, the interactive system

itself must be capable of adapting to the features of a specific user. For this purpose, the system requires information on the specific user, which is then presented in the user model. By using information from the user model, different components of the interactive system can attempt to adapt to the characteristics of a specific user by varying the form, structure, and even the contents of dialogue interaction [51].

The experience of developing student model in ITSs has had a considerable effect on the development of adaptive interactive systems and user models [17]. User models and methods of employing such models are largely similar to student models. However, it should be pointed out that the purpose of constructing a user model from the outset has been the adaptation and not the construction of an exact representation of user knowledge or characteristics. The "adaptation axiom" was incorporated at the outset in constructing user models: It is not necessary to construct a user model that is more complex than can be used in the system for adaptation (compare with principle 2 from Section 4.2). The difference in goals is also responsible for the difference in the methods of using student and user models.

5.2. **The application of student models to adaptation.** In most ITSs, student models are only used as tutoring modules for individualization and support in the teaching process, which is consistent with the primary goal of an ITS: the conveyance of knowledge to the student [9]. In interactive systems, a user model will employ all components to adapt their behavior to the user, and hence the main goal of interactive systems is to achieve effective user function in the interactive dialogue. The goals of knowledge conveyance and adaptation are not contradictory, but are likewise not identical (Fig. 3).

Unfortunately, the problem of adaptation to a student, like adaptation to a user, has been essentially neglected in ITSs, although such systems represent a special class of interactive systems. Adaptation support functions (the main function of user models) do not exist even in a carefully compiled list of ITS functions (Section 3). Moreover, an analysis of several ITSs has shown that the same form of ITS can be successfully used for training individualization and for adaptation, which extends the usefulness of the student models in accordance with the new paradigm. Student models can be used to support adaptive message transmission (QUADBASE and UMFE) and for adaptive control of problem solving (MALT and GCAI). It is necessary to develop ITSs in which all modules, not only the tutoring module, employ student models for adaptation to a specific student. The experience from developing adaptive interactive systems may provide some assistance here.

An attempt was made to use a student model both for tutoring management and for adaptive support of independent user operation in the ISOP system, which combines the capabilities of a tutoring system, an electronic handbook, and an environment for independent operation in the field of programming. For example, in the handbook model, the degree of detail and comprehensiveness of information provided to the student on the structures of the language being studied depends on the degree to which this construction has been studied, as reflected in the overlay student model. The better the construction has been assimilated, the more comprehensive though more laconic the information provided to the student. When the student operates in a program development and debugging environment, the student model controls diagnostic output and the degree of detail in program display to the student for program interpretation by means of a special visualizing interpreter. The lower the level of analysis of the language construction, the more detailed the display of its operation, which enables the student to understand its semantics.

6. A New View of the Role of the Student Model in ITSs

We will attempt to represent the structure and functions of a student model in an interactive tutoring system in which all components use a student model for adaptation to a student (Fig. 4). The central aspect of this system will be the student model itself which reflects the individual features of the user-student and the ongoing picture of student knowledge both on the subject matter and on the system itself, the system structure, and interaction.

Each of the ITS components using a student model employs only a part of the information reflected in the student model and commonly in simplified or modified form. In the ISOP system, the adaptive visualization interpreter must know the degree of mastery by the student of only those knowledge elements reflected in the student model that are language constructions. The interpreter has only four degrees of visualization detail and hence the integer-valued weight of mastery over the language construction in the student model must be projected onto the set (1, 2, 3, 4). Generally, we can state that each module has its own "view" of the student model which we shall refer to as the student model projection. The degree of complexity of this projection, consistent with the adaptation axiom, will be determined by the adaptation limits of the module.

From the organization viewpoint, the student model projections for all modules must be derived from the student

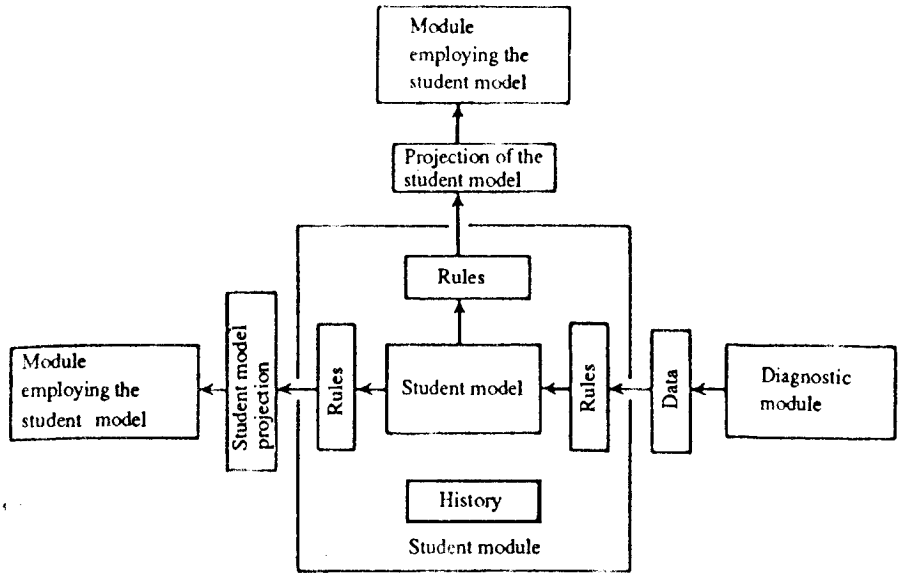

Fig. 4. Structure of the student model.

model and included in the modules themselves. In this case, the student module will contain the student model and procedural knowledge on how to update and support the student model and how to construct an SM projection from the SM itself. This makes it possible to achieve independence of the adaptive modules from the student model, which is turn makes it possible to develop individual ITS modules independently, and even to use several modules in different ITSs. With this structure, a student module that functions as the nucleus of an IRS can itself be independent of the problem-oriented components of the ITS. This creates a path to constructing a standard student module that can be used in different subject domains [52, 53]. Systems exist today (FiSioDisk) in which the student module was borrowed from earlier systems with insignificant modifications.

Even today it is possible to formulate the problem of constructing a model shell that can be used to revise a student module and the nucleus of an ITS for a required subject domain. Such a shell can be constructed since the student model application and support methods as well as possible student model structure are well known and are essentially independent of the subject domain. The most popular approach to constructing standard ITS structures and shells is the rules-based approach [53, 55] or the approach that makes extensive use of rules [18, 52]. In this case, the majority of functions of the student module is implemented as a rule set. Certain sets make possible student model updating based on the history of the tutoring process as well as diagnostic data, while other sets update the student model projections on the basis of the student model itself. Of course, such a two-stage structure is not at all mandatory. Because the student module is a type of black box, with diagnostic data at its input and the student model projection at its output, its internal structure can be constructed on the basis of entirely different principles. For example, if it is possible to construct sets of rules that directly convert diagnostic data and history into student model projections, then there is generally no need for a student model as data structure. In principle, the student model in many systems today is no longer a picture of student knowledge but rather is a log [48]: a history of student interaction with the system that is structured in terms of knowledge elements and processed history.

CONCLUSION

In conclusion it should be noted that considerable attention has been devoted recently to the problems of constructing shells and developing designers for ITSs [18, 35, 52, 53, 55]. Such shells can be used to reduce the labor intensity of ITS construction and can encourage their broad use, as has occurred in the field of expert systems.

1. Barr, A. and E.A. Reigenaum (Eds.). Application Oriented AI Research: Education. The Handbook of Artificial Intelligence, Kaufmann, Los Altos, CA, 1982.
2. Reiser, B.J., J.R. Anderson, and R.G. Farrell. Dynamic student modeling in an intelligent tutor for LISP programming. Proc. 9th International Joint Conference on Artificial Intelligence, Los Angeles, 1985.
3. Bottino, R.M and M.T. Molfino. From CAI to ICAI: An educational and technical evolution. Education and Computing, Vol. 1, No. 4, 1985.
4. Jones, M. Application of artificial intelligence within training. Comput. and Mathemat. with Applications, Vol. 11, No. 5, 1985.
5. Duchastel, P. ICAI systems: Issues in computer tutoring. Computers and Education, Vol. 13, No. 1, 1989.
6. Woolf, B. and D. McDonald. Building a computer tutor: Design issues. Computer, Vol. 17, No. 9, 1984.
7. Ibragimov, O.V. and V.A. Petrushin. Ekspertno-obuchayushchiye sistemy (Expert Tutoring Systems). Ukr. Akad Nauk SSR, Kiev, 1989.
8. Dedé, Chr. A review and synthesis of recent research in intelligent computer-assisted instruction. Intern. J. Man-Machine Studies, Vol. 24, 1986.
9. Wenger, E. Artificial Intelligence and Tutoring Systems. Computational Approaches to the Communication of Knowledge. Morgan Kaufmann, Los Altos, 1987.
10. McCalla, G.F. and J.E. Greer. The Practical Use of Artificial Intelligence in Automated Tutoring: Current Status and Impediments to Progress. Research Report 87–12. Department of Computational Science, University of Saskatchewan, Saskatoon, 1987.
11. Richel, J. Intelligent computer-aided instruction: A survey organized around system components. IEEE Trans. Systems, Man and Cybernetics, Vol. 19, No.1, 1989.
12. Brusilovskiy, P.L. Intelligent tutoring systems. Informatika. Nauchno-tekhnicheskiy sbornik, No. 2, 1990.
13. Hartley, J.R. and D.H. Sleeman. Toward more intelligent teaching systems. Intern. J. Man-Machine Studies, Vol. 5, No. 2, 1973.
14. Hartley, J.R. The design and evaluation of adaptive teaching systems. Intern. J. Man-Machine Studies, Vol. 5, No. 2, 1973.
15. Sleeman, D. and J.S. Brown. Intelligent tutoring systems. In: Sleeman, D.H. and J.S. Brown (Eds.). Intelligent Tutoring Systems. Acad. Press, London, 1982.
16. Self, J. Student models in computer-aided instruction. Intern J. Man-Machine Studies, Vol. 6, 1974.
17. Sleeman, D.H. UMFE: A user modeling front end system. Intern. J. Man-Machine Studies, Vol. 23, 1985.
18. Vassileva, J.A. A classification and synthesis of student modeling techniques in intelligent computer-assisted instruction. In: Norrie, D.H. and H.W. Six (Eds.). Computer-Assisted Learning, Lecture Notes in Computer Science. 438. Proc. of the 3rd International Conference, ICCAL '90, Hagen, FRG, June 1990, Springer-Verlag, Berlin, 1990.
19. Rich, E. Building and exploiting user models. IJCAI-79, Proc. Sixth International Joint Conference on Artificial Intelligence, Tokyo, August 20–23, 1979.
20. Rich, E.A. Users are individuals: Individualizing user models. Intern. J. Man-Machine Studies, Vol. 18, 1983.
21. Goldstein, I.P. The genetic graph: A representation for the evolution of procedural knowledge. Intern. J. Man-Machine Studies, Vol. 11, No. 1, 1979.
22. Clancey, W.J. The role of qualitative models in instruction. In: Self, J. (Ed.). Artificial Intelligence and Human Learning, Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
23. Self, J. The application of machine learning to student modelling. In: Lawler, R.W. and M. Yazdani (Eds.). Artificial Intelligence and Education, Learning Environments and Tutoring Systems. Ablex Publishing, Norwood, 1987.
24. Winkels, R.G.F. User modeling in help systems. In: Norrie, D.H. and H.W. Six (Eds.). Computer-Assisted Learning. Lecture Notes in Computer Science, 438. Proc. 3rd International Conference, ICCAL '90, Hagen, FRG, June 1990, Springer-Verlag, Berlin, 1990.
25. Rastrigin, L.A. and M.Kh. Erenshteyn. An adaptive tutoring system with an adaptable student model. Kibernetika, No. 1, 1984.

26. Wachsmuth, I. Modelling the knowledge base of mathematics learners: Situation-specific and situation-nonspecific knowledge. In: Mandl, H. and A. Lesgold (Eds.). Learning Issues for Intelligent Tutoring Systems. Springer-Verlag, New York, 1988.
27. Brusilovskiy, P.L. An intelligent environment for teaching the principles of programming. In: Ispol'zovaniye kompyuternykh tekhnologiy v obucheniy (The Use of Computer Techniques in Training). Institute of Cybernetics, Ukrainian SSR, Academy of Sciences, Kiev, 1990.
28. Carr, B. and I.P. Goldstein. Overlays: A Theory of Modeling for Computer Aided Instruction. AI Memo 406. MIT, Cambridge, 1977.
29. Gilmore, P. and J. Self. The application of machine learning to intelligent tutoring systems. In: Self, J. (Ed.). Artificial Intelligence and Human Learning, Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
30. Sleeman, D. Inferring student models for intelligent computer-aided instruction. In: Michalski, R.S., G. Carbonell and T. Mitchell (Eds.). Machine Learning, an Artificial Intelligence Approach. Springer-Verlag, Berlin, 1984.
31. Woodroffe, M.R. Plan recognition and intelligent tutoring systems. In: Self, J. (Ed.). Artificial Intelligence and Human Learning, Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
32. Jones, J., M. Millington, and P. Ross. Understanding user behavior in command-driven systems. In: Self, J. (Ed.). Artificial Intelligence and Human Learning. Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
33. Erlandsen, J. and J. Holm. Intelligent help systems. Information and Software Technology, Vol. 29, No. 3, 1987.
34. Breuker, J. Coaching in help systems. In: Self, J. (Ed.). Artificial Intelligence and Human Learning. Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
35. Breuker, J.A., R.G.F. Winkles, and J.A.C. Sandberg. A Shell for Intelligent Help Systems. IJCAI 87, Proc. 10th International Joint Conference on Artificial Intelligence, 1987.
36. Brusilovskiy, P.L. Computer-aided tutoring of computer programming: concepts and problems, Moscow, 1985. Unpublished Paper Deposited in the All-Union Institute of Scientific and Technical Information (VINITI), Archives, File No. 8405-V, 6 Dec. 1985.
37. Self, J. Student models: what use are they? In: Ercoli, P. and R. Lewis (Eds.). Artificial Intelligence Tools in Education. Proc. IEP TC3 Working Conference on AI Tools in Education, Frasacati, May 26–28, 1987.
38. Hartley, J.R. and M.J. Smith. Question answering and explanation diving in on-line help systems. In: Self, J. (Ed.). Artificial Intelligence and Human Learning. Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
39. Burton, R.R. and J.S. Brown. An investigation of computer coaching for informal learning activities. Intern. J. Man-Machine Studies, Vol. 11, 1979.
40. Heines, J. and T. O'Shea. The design of a rule-based CAI tutoral. Intern. J. Man-Machine Studies, Vol. 23, No. 1, 1985.
41. Brusilovskiy, P.L. Subject domain and student models for tutoring management. In: Metody i sredstva kibernetiki v upravlenii uchebnym protsessom vysshey shkoly (Cybernetic Methods and Hardware in Secondary School Training). Riga Polytechnical Institute, Riga, No. 5, 1989.
42. Brusilovskaya, V.G. An intelligent environment for support of differentiation training. In: Ispol'zovaniye kompyuternykh tekhnologiy v obucheniy (Application of Computer Technology in Training). Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev, 1990.
43. Ohlson, S. Some principles of intelligent tutoring. Instructional Science, Vol. 14, 1986.
44. Peachey, D.R. and G.I. McCalla. Using planning techniques in intelligent tutoring systems. Intern. J. Man-Machine Studies, Vol. 24, 1986.
45. Babenko, V.A. An expert system for quality testing of student knowledge. Preprint, Institute of Applied Information Sciences, USSR Academy of Sciences, Moscow, 1990.
46. Clancey, W.J. Tutoring rules for guiding a case method dialog. Intern. J. Man-Machine Studies, Vol. 11, 1979.
47. Self, J. Bypassing the intractable problem of student modelling. In: Frasson, C. (Ed.). ITS-88. Proc. Intelligent Tutoring Systems, Montreal, June 1–3, 1988. Univ. Montreal, Montreal, 1988.
48. Laurilard, D. The pedagogical limitations of generative student models. Instructional Science, Vol. 17, No. 3, 1988.

49. Newman, D. Is a student model necessary? Apprenticeship as a model for ITS. In: Bierman, D., J. Breuker, and J. Sandberg (Eds.). Artificial Intelligence and Education. Proc. 4th International Conference on AI and Education, May 24–26, Amsterdam, 1989.
50. Cooper, M. Interfaces that adapt to user. In: Self, J. (Ed.). Artificial Intelligence and Human Learning. Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
51. Benyon, D. and D. Murray. Experience with adaptive interfaces. Computer J., Vol. 31, No. 5, 1988.
52. Winkels, R.G.F., W. Achthofen, and A. van Gennip. Methodology and modularity in ITS design. In: Dierman, D., J. Breuker, and J. Sandberg (Eds.). Artificial Intelligence and Education. Proc. 4th International Conference on AI and Education, May 24–26, Amsterdam, 1989.
53. O'Shea, T., et al. Tools for creating intelligent computer tutors. In: Elitorn, A. and R. Banerjii (Eds.). Artificial and Human Intelligence. Elsevier, London, 1984.
54. Murray, W.R. Control for intelligent tutoring systems: A blackboard-based dynamic instructional planner. In: Bierman, D., J. Breuker, and J. Sandberg (Eds.). Artificial Intelligence and Education. Proc. 4th International Conference on AI and Education, May 24–26, Amsterdam, 1989.
55. Crews, P. The EXPERT: A case study in integrating expert system technology with computer assisted instruction. Proc. Third International Conference on Data Engineering, 1987.
56. Swigger, D.M. and D. Evans. A computer-based tutor for assembly language. J. Computer-Based Instruction, Vol. 14, No. 1, 1987.
57. Barr, A., M. Beard, and R.C. Atkinson. The computer as tutorial laboratory: The Stanford BIP project. Intern. J. Man-Machine Studies, Vol. 8, No. 5, 1976.
58. Wescourt, K.T., M. Beard, and L. Gould. Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. Proc. 1977 Annual ACM Conference, Seattle, October 1977.
59. Bonar, J. and R. Cunningham. Bridge: An intelligent tutor for thinking about programming. In: Self, J. (Ed.). Artificial Intelligence and Human Learning, Intelligent Computer-Aided Instruction. Chapman and Hall, London, 1988.
60. Kawai, K., R. Mizoguchi, O. Kakusho, and J.A. Toyoda. A framework for ICAI systems based on inductive inference and logic programming. New Generation Computing, Vol. 5, 1987.
61. Brown, J.S. and R.R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, Vol. 2, 1978.
62. Koffman, E.B. and J.M. Perry. A model for generative CAI and concept selection. Intern. J. Man-Machine Studies, Vol. 8, 1976.
63. Reiser, B.J., M. Ranney, M.C. Lovert, and D.Y. Kimberg. Facilitating student's reasoning with causal explanations and visual representations. In: Bierman, D., J. Breuker, and J. Sandberg (Eds.). Artificial Intelligence and Education. Proc. 4th International Conference on AI and Education, May 24–26, Amsterdam, 1989.
64. Swigger, K., H. Burus, H. Loveland, and T. Jackson. An intelligent tutoring system for interpreting ground tracks. AAAI-87, Proc. 6th National Conference on Artificial Intelligence, Seattle, July 13–17, 1987.
65. Anderson, J.R., C.F. Boyle, and G. Yost. The geometry tutor. Proc. 9th International Joint Conference on Artificial Intelligence, Los Angeles, 1985.
66. Sleeman, D.H. and M.J. Smith. Modeling student's problem solving. Artificial Intelligence, Vol. 16, 1981.
67. Anderson, J. and G.B. Reiser. The LISP teacher. In: Real'nost i prognozy iskusstvennogo intellekta (The Reality and Future of Artificial Intelligence). Mir Press, Moscow, 1987 [Russian translation].
68. Koffman, E.B. and S.E. Blount. Artificial intelligence and automatic programming in CAI. Artificial Intelligence, Vol. 6, 1975.
69. Genesereth, M.R. The role of plans in intelligent tutoring systems. In: Sleeman, D.H. and J.S. Brown (Eds.). Intelligent Tutoring Systems. Acad. Press, London, 1982.
70. Sleeman, D. PIXIE: A shell for developing intelligent tutoring systems. In: Lawler, R.W. and M. Yazdani (Eds.). Artificial Intelligence and Education, Learning Environments and Tutoring Systems. Ablex Publishing, Norwood, 1987.
71. Johnson, W.L. and Z. Solouis. PROUST (An automatic debugger for Pascal programs). In: Real'nost i prognozy iskusstvennogo intellekta (The Reality and Future of Artificial Intelligence). Mir Press, Moscow, 1987 [Russian translation].

72. Hudson, P.V. and J.A. Self. A dialogue system to teach database concepts. Computer J., Vol. 25, No. 1, 1982.
73. Woolf, B.P. Representing complex knowledge in an intelligent machine tutor. In: Self, J. (Ed.). Artificial Intelligence and Human Learning. Intelligent Computer-Aided Instruction, Chapman and Hall, London, 1988.
74. Carbonell, J.R. AI in CAI: An artificial intelligence approach to computer-aided instruction. IEEE Trans. Man-Machine Systems, Vol. MMS-11, No. 4, 1970.
75. Lesgold, A., et al. A coached practice environment for electronics troubleshooting. In: Larkin, J., et al. (Eds.). Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration. Lawrence Erlbaum, Hillsdale, N.J., 1990.
76. Shute, V.J. and R.A. Glaser. A large-scale evaluation of an intelligent discovery world: Smithtown. Intelligent Learning Environments, Vol. 1, 1990.
77. Brown, J.S., R. Burton, and J. de Kleer. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III. In: Sleeman, D.H. and J.S. Brown (Eds.). Intelligent Tutoring Systems. Acad. Press, London, 1982.
78. Baldwin, J.T. and L. Siklossy. An unobtrusive computer monitor for multi-step problem solving. Intern. J. Man-Machine Studies, Vol. 9, No. 3, 1977.
79. Stevens, A.L. and A. Collins. The goal structure of a socratic tutor. Proc. 1977 Annual ACM Conference, Seattle, October, 1977.
80. Raats, Yu.Yu. and A.Yu. Tolmacheva. An adaptive computer-aided design system for teaching differentiation. Avtomatika i vychisliel'naya tekhnika, No. 3, 1980.