The Acquisition of Functional Planning- and Programming Knowledge: Diagnosis, Modelling, and User-Adapted Help

An Overview of the Project
"Development of an Adaptive Knowledge Diagnosis and Error Explanation System for the Acquisition of Programming Knowledge for ABSYNT"
(Grant No. Mo 292/3 1-3)

Claus Möbus, Olaf Schröder

Introduction

The development of runnable models of human information processing and their empirical validation has become an important research area, especially concerning processes of knowledge acquisition. Since models of this kind explicitly represent hypothetical knowledge structures and processes, they enable detailed hypotheses concerning the representation and change of knowledge, and they enable detailed and sometimes surprising predictions. Thus they contribute to psychological theorizing and to basic research. On the other hand, modelling the acquisition and change of knowledge is also one necessary condition for the design of individualized instruction and help for computer based systems. Individualized, user-centered online instruction and help not only enhances knowledge acquisition processes but is also important for the design and development of various software tools. For example, the *acceptance* of such tools may depend on their sensitivity to the actual knowledge and intentions of the user.

The topic of our project has been to empirically investigate and to model processes of the acquisition, utilization, and optimization of knowledge while working with the *ABSYNT Problem Solving Monitor*. (*PSM*). The ABSYNT PSM is designed to support the acquisition of basic functional programming concepts by supplying learners with individualized, adaptive online help and proposals. ABSYNT ("Abstract Syntax Trees") is a functional visual programming language developed in the project. The ABSYNT PSM provides help for the learner constructing ABSYNT programs to given tasks.

The design and development of a system like the ABSYNT PSM is not possible without creating models representing knowledge states and knowledge acquisition processes of learners. Our system is embedded in a *three level approach*:

• First there is a theoretical framework of problem solving and knowledge acquisition: the *ISP-DL Theory* (Impasse - Success - Problem Solving - Driven Learning). Its purposes are

- to describe and to explain the continuous stream of actions and verbalizations of the learner while working on programming tasks

- to represent a guideline for design decisions of the system

- to motivate and constrain models of learners´ knowledge states and acquisition processes.

• An Internal Model or *State Model* diagnoses the actual domain knowledge of the learner (rules and schemas) at different states in the knowledge acquisition process. It is designed to be an integrated part of the ABSYNT PSM ("internal" to it) and to perform *online* knowledge diagnosis based on computer-assessable data provided by the learner´s interactions with the system. The State Model is the basis for user-adapted help.

• An External Model or *Process Model* is designed to simulate the knowledge acquisition processes, problem solving heuristics, and (in future) motivational processes of the learner on a level of detail not available to the State Model. The Process Model is not part of the ABSYNT PSM ("external" to it) but supports the design of the State Model. The Process Model is an *offline* model designed to bridge the gap between the ISP-DL Theory and the State Model by providing hypothetical *reasons* for the knowledge state changes represented in the State Model. For example, the Process Model should hypothesize in which kinds of situations which kinds of weak heuristics are preferred by the learner.

Thus ISP-DL Theory, State Model and Process Model are designed to be mutually consistent but serve different purposes. Figure 1 summarizes their interrelationships: The *ISP-DL Theory* controls the development of the *Process Model* and the *State Model*. *Data* available from the learner while working with the system are: stating and testing *hypotheses* about created solution proposals, programming actions and the time needed for them, and verbalizations. Such data are used for online construction and updating of the State Model, and / or for offline development and testing predictions of the Process Model. The State Model is in part based on an *Expert Model* representing planning knowledge and implementation knowledge of the ABSYNT domain. Both the State Model and the Expert Model are designed to provide user-adapted *help* and *explanations*.

First we will briefly describe the ISP-DL Theory since it provides the "roof" for the design of the ABSYNT PSM, the State Model, and the Process Model. Then the ABSYNT PSM is presented. In the next section we will describe the State Model and empirical analyses of some of its predictions. In addition, we show which empirical predictions follow from the State Model for different kinds of help information if the adaptivity of the information to the learner´s knowledge is varied. Finally, the Process Model is described.

The ISP-DL Theory

As indicated, the ISP-DL Theory is intended to describe the continuous stream of problem solving and learning of a student as it occurs in a sequence of, for example, programming sessions. Empirical analyses of students´ sessions working with ABSYNT

(Möbus & Thole, 1990; Schröder, 1990) showed that such streams can be described as an interplay of problem solving, impasse-driven learning, and success-driven learning. Several approaches have been concerned with these processes, like van Lehn´s theory of Impasse-Driven Learning (van Lehn, 1988; 1990; 1991b), SOAR (Laird, Newell & Rosenbloom, 1986; 1987; Newell, 1990; Rosenbloom et al., 1991); or knowledge compilation mechanisms as for example incorporated in ACT* (Anderson, 1983; 1986; 1989) or described by Wolff (1987; 1991). Consequently, the ISP-DL Theory is an attempt to give an integrated account of the phenomena described by these and other approaches. The ISP-DL Theory has the following features (for more detail see Möbus, 1991b; Möbus, Schröder & Thole, in press):

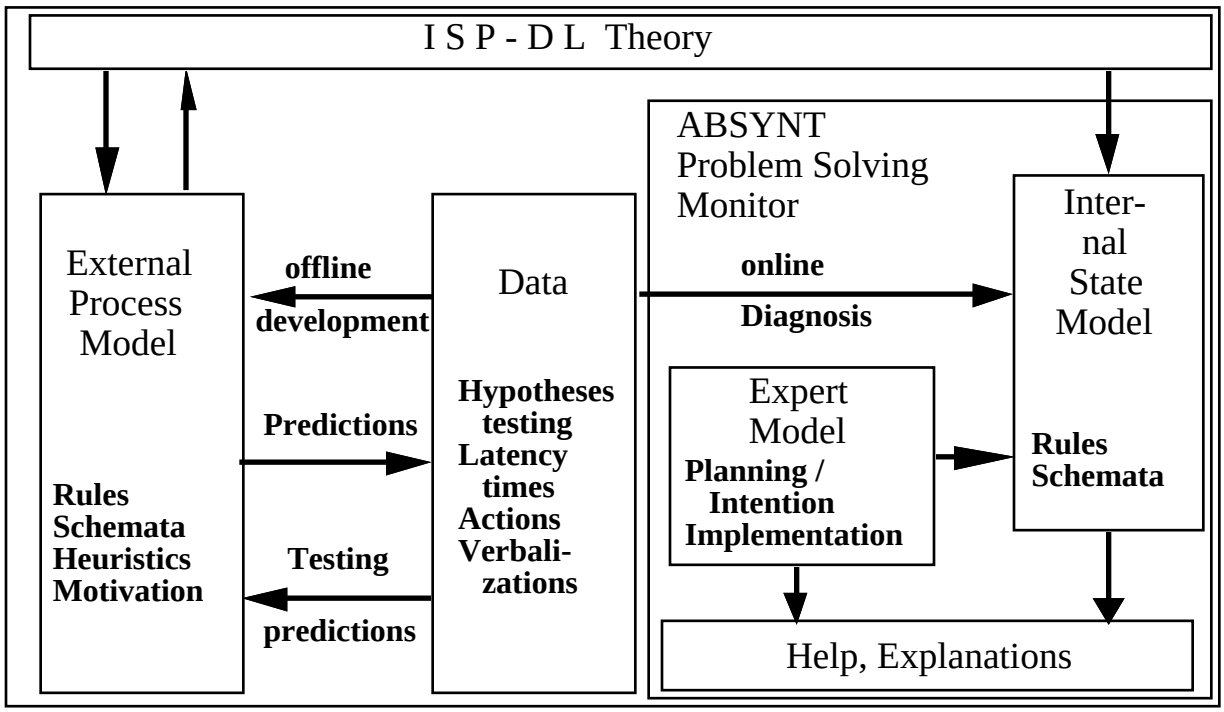

Figure 1: The interrelationships between ISP-DL Theory, Process Model, State Model, Expert Model, data of the learner, and help and explanations

• The distinction of different problem solving phases (according to Gollwitzer, 1990; 1991): In the *Deliberate Phase*, the problem solver (PS) thinks about several goals. A goal is a set of facts which the PS wants to become true (Newell, 1982) and can be expressed as a predicative description. It ends when the PS chooses one goal to pursue: The PS is now committed to an intention. In the second phase, the PS *plans* a solution for the chosen goal. A plan is a partially ordered sequence or hierarchy of subgoals which may be created by domain-specific operators or by weak heurtistics. In the third phase, the plan is *executed*, and finally the result is *evaluated*. For example, the PS may check whether the obtained result satisfies the predicative description representing the main goal.

• *The impasse-driven acquisition of new knowledge*. Roughly, an impasse is a situation where "the architecture cannot decide what to do next given the knowledge and the situation that are at its current focus of attention." (van Lehn, 1991b, p. 19). Impasses may arise at different

points in problem solving (Newell, 1990, p. 174). For example, the PS may not be able to decide for a goal, or to synthesize a plan, or the plan might not be executable, or there are no criteria to evaluate a result. In response to impasses, the PS applies weak heuristics, like asking questions, looking for help, cheating, and so on (Laird, Rosenbloom & Newell, 1987; van Lehn, 1988; 1989; 1990; 1991b). As a result, new knowledge may be acquired that leads to overcoming the impasse. So impasses are situations where the learner is likely to actively look for *help* (van Lehn, 1988). But it is also possible that the information obtained is not helpful. For example, information intended as help may be too complicated, or confusing, or leading into a wrong direction. So instead of solving the impasse, a secondary impasse may arise, as already described by Brown & van Lehn (1980).

• *The success-driven improvement of acquired knowledge*. Successfully used knowledge (knowledge which application did not lead to an impasse) may be improved so it may be used more effectively in the future. More specifically, by *rule composition* (Anderson, 1983, 1986; Lewis, 1987; Neves & Anderson, 1981; Vere, 1977), the number of control decisions and subgoals to be set may be reduced, because less rule selections have to be performed. In contrast to these authors, our composition is based on resolution and unfolding (Hogger, 1990), see Möbus, Schröder & Thole (1991; in press). This has theoretical and empirical advantages.

The ABSYNT Problem Solving Monitor

ABSYNT ("Abstract Syntax Trees") is a functional, visual programming language based on ideas stated in an introductory computer science textbook (Bauer & Goos, 1982). ABSYNT is a tree representation of pure LISP and is aimed at supporting the acquisition of basic functional programming skills, including abstraction and recursive systems. The design of ABSYNT as a visual programming language was based on

- *two alternative runnable specifications* of the ABSYNT interpreter (Möbus & Thole, 1989; Möbus & Schröder, 1990) which were developed according to cognitive science principles and constraints (Larkin, Simon, 1987; Pomerantz, 1985)
- empirical studies (Colonius et al., 1987; Schröder, Frank, Colonius, 1987) concerning the mental representation of and misconceptions about functional programs.

This work served to prepare the development of the ABSYNT PSM (Kohnert, Janke, 1988/91) according to principles of visual learning environments (Glinert, 1990). The motivation and analysis of ABSYNT with respect to properties of visual languages is described in Möbus & Thole (1989).

The ABSYNT PSM provides an *iconic programming environment* (Chang, 1990). Its main components are a visual *editor*, a visual *trace*, and a *help component*: a *hypotheses testing*

environment. The design of the ABSYNT PSM is motivated by the ISP-DL Theory in several respects:

• According to the ISP-DL Theory, the learner will look for and appreciate help if he or she is caught in an impasse. Without an impasse there is no need for help. So the ABSYNT PSM does not interrupt the learner (see for example also Winkels & Breuker, 1990), but *offers* help. As a side effect, this design principle provides valuable data about the student´s impasses.

• According to the ISP-DL Theory, the learner should be prevented from trapping into secondary impasses which may lead away from the original problem solving. This may be done by letting the learner make use of his pre-knowledge at impasses as much as possible. In the ABSYNT PSM, this principle is realized by the *hypotheses testing approach*. The learner may state hypotheses about which part of his current solution proposal he considers correct. The system then analyzes the hypothesis and gives feedback. The student can also ask the system for completion proposals (see below). Another reason for the hypotheses testing approach is that in programs it is usually not possible to absolutely localize bugs. Often the bug consists of an inconsistency between program parts, and there are several ways to fix it. The hypotheses testing approach leaves the decision how to change a buggy program to the *PS*. Again, a side effect of the hypotheses testing approach is that it provides a rich data source of the learner´s problem solving.

• According to the ISP-DL Theory, help should be provided at different levels of problem solving. The ABSYNT PSM supports the problem solving phases of *planning*, *executing*, and *evaluating* solution proposals. A solution proposal may be *planned* first by using goal nodes. So the learner may create a plan and test hypotheses about it without bothering about its implementation at this stage. The *implementation* of the goals (thus creating an executable program) may be done later. Finally, *evaluation* is again supported by hypothesis testing.

Figure 2 depicts snapshots from the ABSYNT PSM. Figure 2a shows the *visual editor* where ABSYNT programs can be created. There is a head window and a body window. On the left side of Figure 2a, there is the tool bar of the editor: The bucket is for deleting nodes and links. The hand is for moving, the pen for naming, and the line for connecting nodes. The "goal" node will be explained below. Next, there is a constant, parameter and "higher" operator node (to be named by the learner, using the pen tool). Constant and parameter nodes are the *leaves* of ABSYNT trees. Then several primitive operator nodes follow ("if", "+", "-", "*", ...). Editing is done by selecting nodes with the mouse and placing them in the windows, and by linking, moving, naming, or deleting them. Nodes and links can be created *independently*: If a link is created before the to-be-linked nodes are edited, then shadows are automatically created at the link ends. They serve as place holders for nodes to be edited later. Shadows may also be created by clicking into a free region of a window.

Constant, parameter and operator nodes are *implementation* nodes. A syntactically correct ABSYNT program is runnable if it consists only of implementation nodes. Implementation nodes have three horizontal parts: an input stripe, a name stripe, and an output stripe. (Constant nodes have only two stripes because name and output are identical.) In the *visual trace* of the ABSYNT PSM (not depicted), input and output stripes are filled with computation goals and obtained values, so each computational step of the ABSYNT interpreter can be visualized (Möbus & Schröder, 1989; 1990; Möbus & Thole, 1989).

But as already indicated, in ABSYNT there are also *goal* nodes designed to support *deliberating* and *planning*. Clicking on the "goal" symbol in the tool bar (Figure 2a, on the left) causes the tool bar to switch to the actual goal nodes. Goal nodes represent more abstract plan fragments which may be implemented in several ways by implementation nodes or subtrees. Visually, goal nodes have no internal structure. In Figure 2a, "LIST EMPTY" and "CASE" are examples of goal nodes. Each goal node is precisely defined as a predicative description for the yet to be implemented program fragments. For example, "LIST EMPTY" represents the goal to test whether a list is empty. The "CASE" node represents the goal to program conditionalized expressions, that is, condition-expression pairs. The ABSYNT goal nodes are based on a task analysis which applies the *transformation* approach developed in the Munich CIP Project (Bauer et al., 1987; Partsch, 1990). The transformation approach ensures that solution can be derived to a given task that is correct with respect to the task description. Currently ABSYNT supports 42 programming tasks. For each task, there is a goal node with a predicative and a verbal task description. Data types are numbers, truth values, and lists.

In Figure 2a, a wrong solution proposal for an ABSYNT program reversing a list is just being created. There are nodes not yet linked or even completely unspecified (shaded areas). As Figure 2a shows, goal nodes and implementation nodes can be mixed ("mixed terms") within a proposal. The solution proposal in Figure 2a means:

```
If L is equal to the value of a yet unknown expression,
then the value of REVERSE is NIL,
else if L is an empty list,
then    if the value of a yet unknown expression is the empty list,
        then the value of REVERSE is the value of L,
        else the value of REVERSE is obtained by CONSing the values of two yet
                unknown expressions together.
```

In the *hypotheses testing environment* the learner may state hypotheses (bold parts of the program in Figure 2b) about the correctness of a solution proposal or parts thereof for a given programming task. The hypothesis is: "It is possible to embed the boldly marked fragment of the program in a correct solution to the current task!". The system then analyzes the hypothesis.

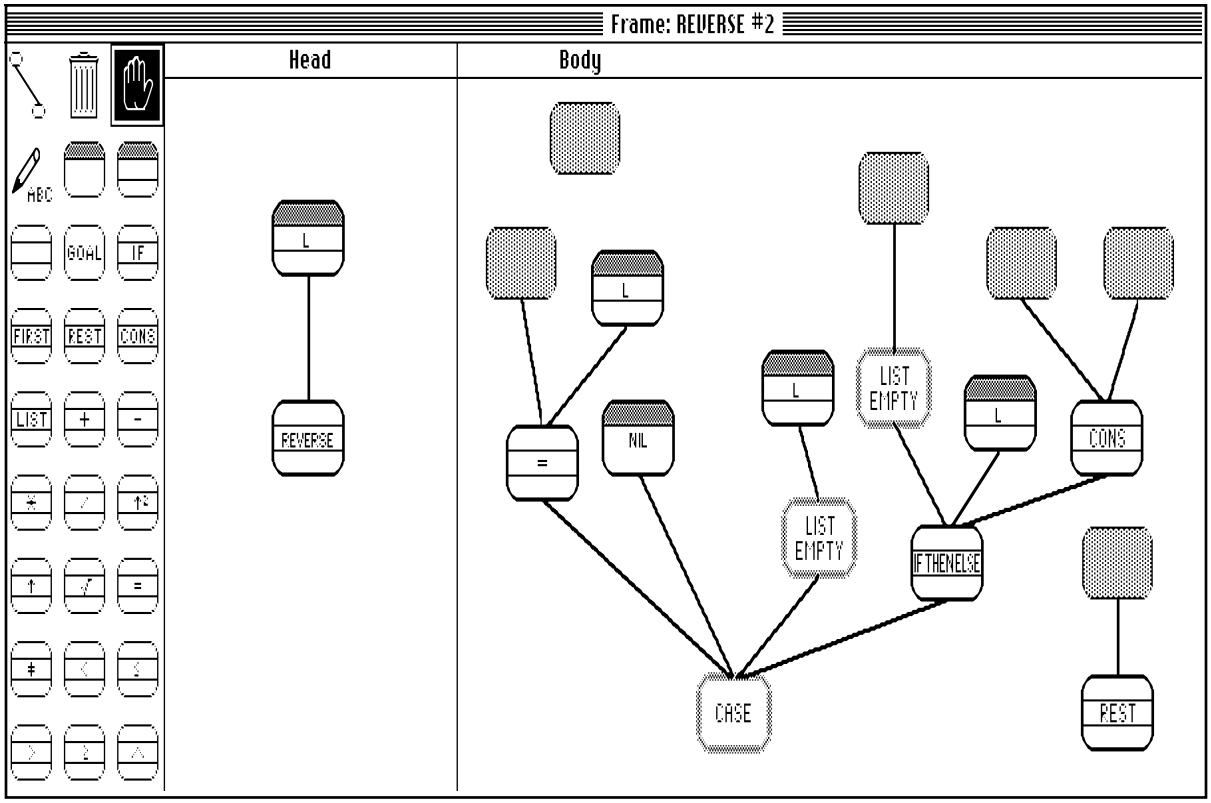

Figure 2a: Student´s erroneous and overly complicated proposal in the visual editor

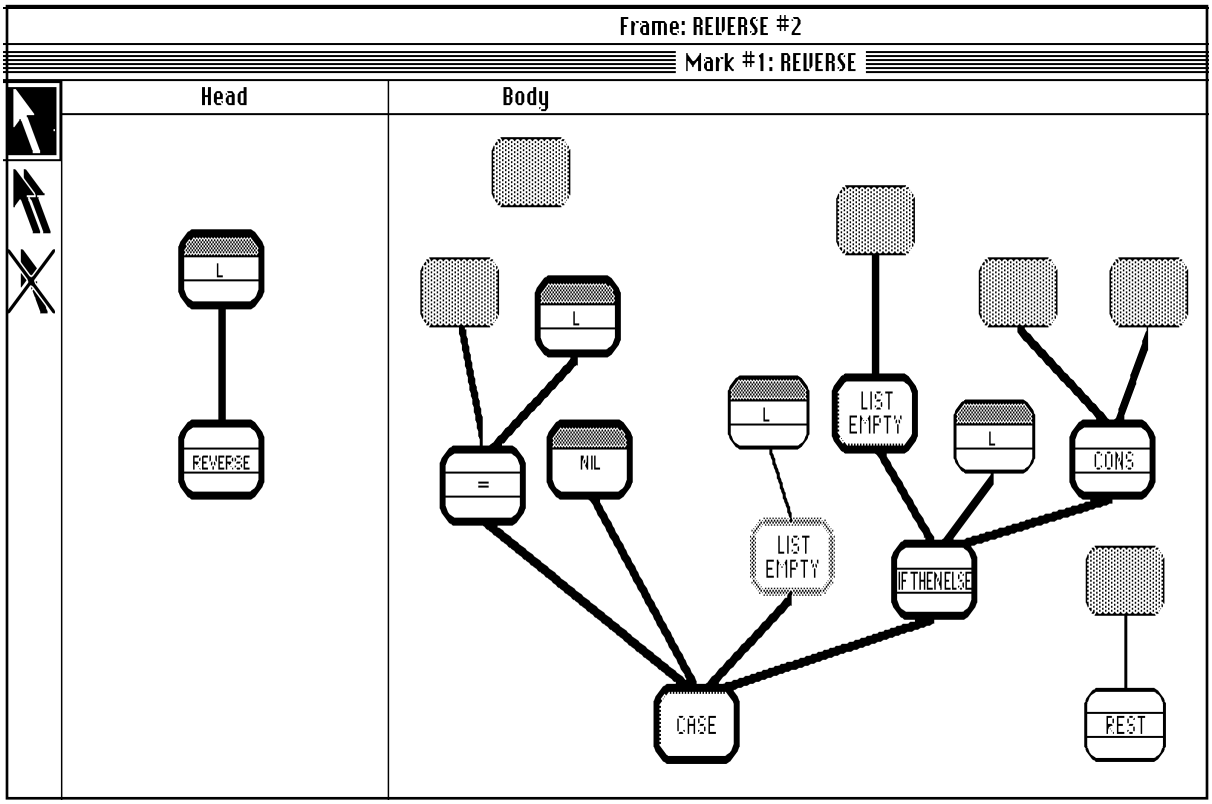

Figure 2b: Student´s hypothesis (bold nodes and links)

Figure 2: Snapshots of problem solving with ABSYNT: Student´s erroneous proposal to the REVERSE problem (Figure 2a), student´s hypothesis proposal (Figure 2b), feedback of the ABSYNT system (Figure 2c), and completion proposal of the ABSYNT system (Figure 2d)
continued on the next page

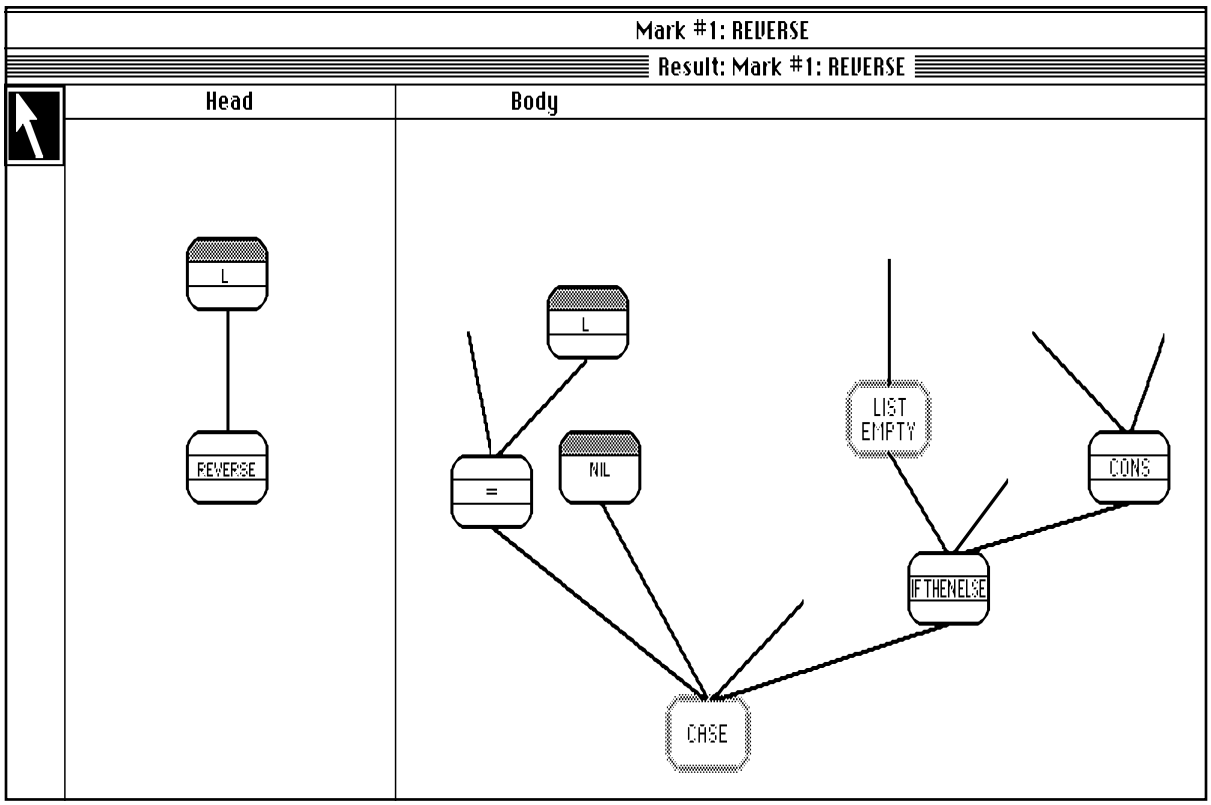

Figure 2c: Positive Feedback of the ABSYNT system to student´s hypothesis

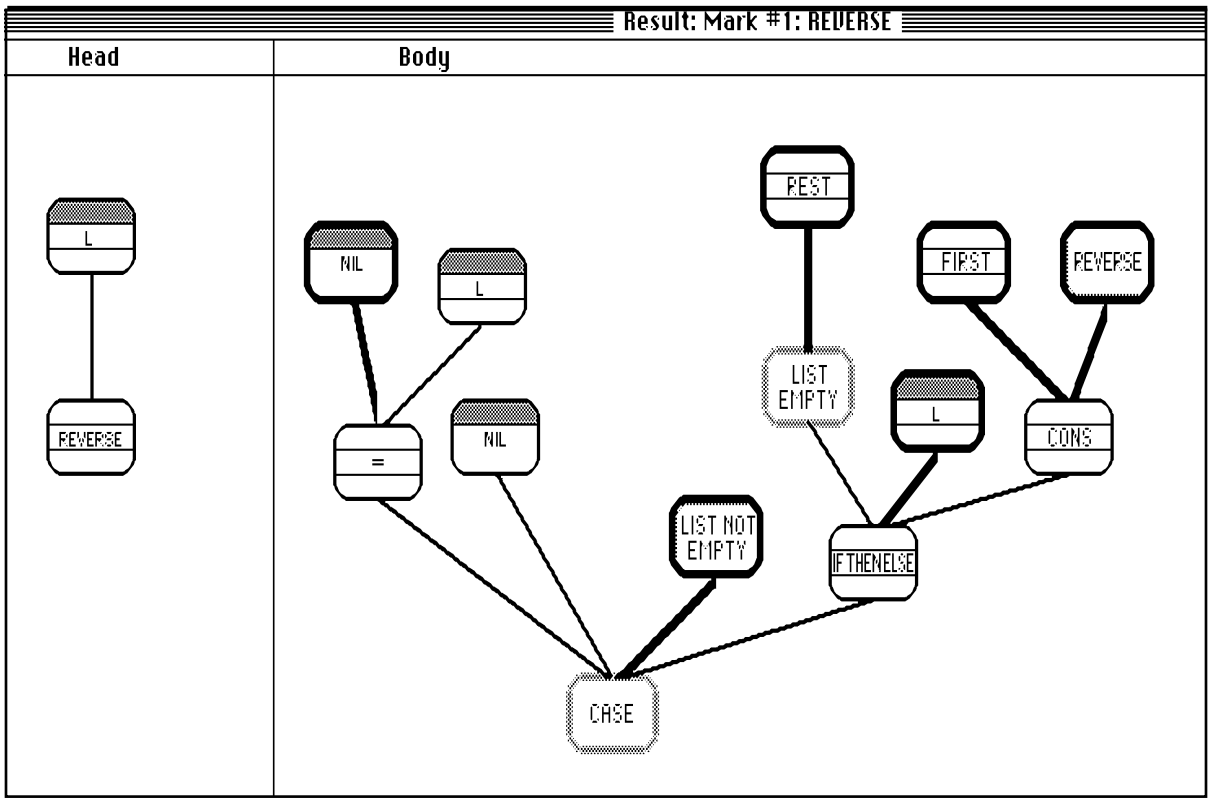

Figure 2d: Completion proposals of the ABSYNT system on student demand

If the hypothesis can be confirmed the PS is shown a copy of the hypothesis (Figure 2c). If this information is not sufficient to resolve the impasse, the PS may ask for more information (completion proposals, Figure 2d). If the hypothesis cannot be confirmed the learner receives the message that the hypothesis cannot be completed to a solution known by the system.

In Figure 2b the learner stated a hypothesis which covers a fragment of the proposal created so far for the "reverse" programming task. The hypothesis contains goal nodes and implementation nodes. The system recognizes the hypothesis as embeddable, indicating this by returning a copy of the hypothesis to the student (Figure 2c). If this information is not sufficient for solving the impasse, the student may ask the system for completion proposals at the open links. In Figure 2d, the student asked for and received six completions (bold). Two of them are goal nodes, the others are implementation nodes. The "REVERSE" goal node represents the task goal. As far as possible, the system tries to generate completions consistent with the student´s proposal. At one point, the system disagrees with the student´s proposal: The system proposes "LIST NOT EMPTY" at the third input link of the CASE node, whereas the student´s original proposal contains "LIST EMPTY" at this point (Figure 2a). Internally, the system has created a complete solution but the student always gets only minimal information.

The hypotheses testing environment is the most significant aspect where the ABSYNT PSM differs from other systems designed to support the acquisition of functional programming knowledge, like the LISP tutor (Anderson & Swarecki, 1986; Anderson, Conrad & Corbett, 1989; Corbett & Anderson, 1992), the SCENT advisor (Greer, 1992; Greer, McCalla & Mark, 1989), and the ELM system (Weber, 1988; 1989; 1992). As indicated, one reason for the hypotheses testing approach is that in programming a bug usually cannot be absolutely localized. Hypotheses testing leaves the decision which parts of a buggy solution proposal to keep to the student and thereby provides a rich data source about the learner´s knowledge and intentions. Single subject sessions with the ABSYNT PSM revealed that hypotheses testing was heavily used. It was almost the only means of debugging wrong solution proposals, despite the fact that the subjects had also the visual trace available. This is partly due to the fact that in contrast to the trace, hypotheses testing does not require a complete ABSYNT program solution. Hypotheses testing is possible with incomplete solutions, with goal nodes, and with mixed terms. So the student may obtain feedback whether he is on the right track at very early planning stages.

The answers to the learner´s hypotheses are generated by rules defining a *goals-means-relation (GMR)* (Levi & Sirovich, 1976; Nilsson, 1980). A subset of these rules may be viewed as "pure" expert domain knowledge not influenced by learning. Thus we call this set of rules EXPERT. Currently, EXPERT contains about 1300 planning rules and implementation rules. The planning rules elaborate goals, and the implementation rules describe how to realize goals by ABSYNT implementation nodes. The goal decomposition done by the planning rules

follows the CIP transformation approach mentioned earlier. EXPERT is able to analyze and to synthesize several millions of plans and solutions for the 42 tasks (Möbus, 1990; 1991; Möbus & Thole, 1990). We think that such a large solution space is necessary because we observed that especially novices often construct unusual solutions due to local repairs.

The completions shown in Figure 2d (bold program fragments) were generated by EXPERT rules. EXPERT analyzes and synthesizes solution proposals but is not *adaptive* to the learner´s knowledge. Usually EXPERT is able to generate a large set of *possible* completions. For example, EXPERT could generate a large number of alternatives for the "LIST NOT EMPTY" goal node in Figure 2d: for example, the goal nodes NOT, EQUAL, GREATER THAN 0 (with respect to the length of the list), or the implementation nodes =, ≠. Thus the problem is to *select* the most appropriate completion proposal. This is the main function of the internal State Model. It represents the actual knowledge state of the learner and consists of rules derived from EXPERT. The State Model should choose a completion which is maximally *consistent* with the learner´s current knowledge state. This should minimize the learner´s surprise to feedback and completion proposals. The State Model is implemented but not yet integrated into the ABSYNT PSM. It will be described in the next section.

To close this section, Figure 3 gives a summary of the components involved in planning, programming and hypotheses testing in the ABSYNT PSM. The learner solves problems, acquires new knowledge due to impasse-driven learning, and optimizes knowledge due to success-driven learning. In Figure 3, the PS created a partial plan to the "even" task: "program that tests whether a natural number is even." The situation shows a very early stage of program development. As can be seen from the screen the PS is trying to do case analysis. His decision was to split the "even" problem into two cases. The PS defined the first case as "N EQUAL 0". Under this condition the goal "EVEN" has still to be solved. There is no proposal for the second case. The PS knows only that under the condition of the second case the original "EVEN" problem has to be solved. When the PS states a hypothesis, the system analyzes it, that is, it diagnoses the intentions (planning steps) and the actions (implementation steps) of the PS. The diagnosis of actions and intentions leads to an updating of the State Model, and to positive feedback, error messages, or help (completion proposals) which are tailored to the learner. The learner will process this information (acquire new knowledge, optimize knowledge) and continue with programming, planning, or hypotheses testing. Figure 3 also shows the place of the internal process model in this framework.

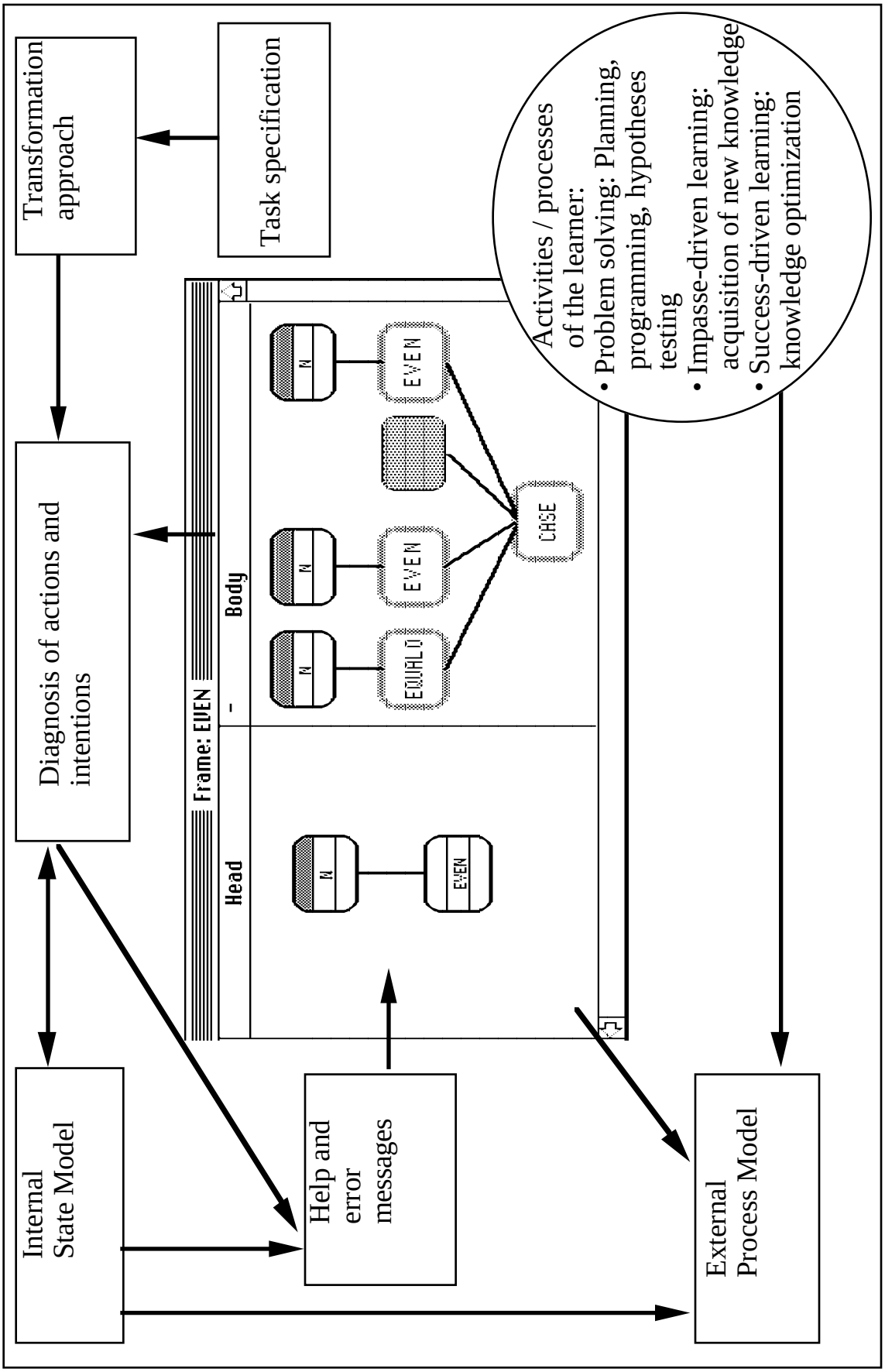

Figure 3: Planning, programming, and hypotheses testing with the ABSYNT PSM

The Internal State Model

The State Model is designed as an integrated part of the ABSYNT PSM. It represents the actual hypothetical domain knowledge of the learner at different points in the knowledge acquisition process. The hypothetical domain knowledge is organized as a partial order of *micro rules*, *schemas*, and specific *cases*. Micro rules represent knowledge newly acquired by *impasse-driven learning* but not yet optimized. They describe small planning or implementation steps in the ABSYNT domain. Schemas and cases are created by rule composition according to the resolution method (Möbus, Schröder, Thole, in press).

The State Model is created and updated by automatically inspecting the single editing steps performed by the user while constructing ABSYNT programs (Möbus, Schröder, Thole, 1991; in press). It is constructed according to the following, simplified description:

• After each programming task solved, the action trace of the PS is parsed by the micro rules, schemas, and cases of the State Model and (as far as needed) by EXPERT rules. (For example, before the learner has solved the first task, the State Model is empty, so the first solution has to be parsed completely by EXPERT rules.) According to Corbett, Anderson, Patterson (1988) we call this process *model tracing*.

• The composites of all rules just used for parsing are created.

• Each rule used for parsing and each new composite just created is checked for *plausibility*. A rule is considered *plausible* with respect to an action trace if the ABSYNT program fragments specified by this rule are contained in the action trace in an uninterrupted temporal sequence.

• Each rule just found plausible is put into the State Model if it is not yet contained in it. For each plausible rule already in the state model, its credit value is raised. For composites (schemas and cases), there are two additional requirements: The part of the subject´s action trace described by a composite must have been performed *earlier* already, and the actual sequence has to be performed *faster* than at that earlier time. This is because composites are designed to represent knowledge optimization.

Figure 4 illustrates some of the features of the State Model. It depicts a continuous fragment of a sequence of programming actions performed by a subject. These data and associated times are stored in the action trace. To give an example, there is a micro rule in EXPERT which describes the following four actions: Placing an ABSYNT if-then-else node, and creating three input connections to it. In Figure 4, these four actions are performed at 11:15:52, 11:15:58, 11:16:46, and 11:16:55 (fragments with bold margins). So the actions corresponding to the "if-then-else-node" rule is interrupted at 11:16:42 and 11:16:50. So this rule is not plausible. This example illustrates that the State Model is created online based on detailed data: individual programming actions.

As indicated, the purpose of the State Model is to provide individualized help and completion proposals. It also gives rise to empirical predictions concerning verbalizations and order constraints on programming actions. For example, it says that as the learner shifts from novice to expert (that is, acquires successively more schemas and cases), there will not only be less verbalizations and more performance speedup, but it also different order constraints for solution steps.

Figure 5 illustrates this. The upper part of Figure 5 shows a set of four micro rules, and a set of two schemas. Each rule or schema is abstractly represented as a goals-means-pair (a goal tree *g*... and an ABSYNT program fragment *a*...). The State Model postulates that if the learner applies a certain rule or schema, then the corresponding programming action(s) should be performed, and vice versa. If the learner verbalizes an intention, there should be a corresponding goal in the rule assumed to be applied. Conversely, for goals not part of the rule, there should be no verbalizations at all (middle part of Figure 5). Together with the concept of plausibility built into the State Model, this means that if the learner applies a certain rule or schema, then the corresponding action sequence of actions and goal verbalizations should be performed in an uninterrupted sequence. Actions and verbalizations stemming from different rules *should not interleave* (*no-interleaving hypothesis*). So if the State Model contains the set {r1, r2, r3, r4} of micro rules, then an empirical sequence like [g2, a1, a2, ...] should *not* be observable (Figure 5) because a1 interrupts the sequence of events [g2, a2] explained by r2. But an event sequence like [g2, a2, cv, a4, g4, ...] is consistent with the set {r1, r2, r3, r4}: there is no interleaving of events explained by different rules. "cv" denotes actions and verbalizations, like moving and rearranging nodes and links, which are indicators of control or heuristic processes. These control events could occur "between" events of different rules. They also should not interrupt an event sequence explained by one rule. Control events are to be explained by the external Process Model.

For the two schemas {s1, s2}, the prediction is that the data explained by s1 should not interleave with the data explained by s2. Again, control events could occur "between" schema events. As Figure 5 shows, the event chain [g2, a1, a2, ...] which was inconsistent with {r1, r2, r3, r4} is *consistent* with {s1, s2}, but an event chain like [a1, g3, g2, ...] is not, because a1 and g2 stem from the same schema but are interrupted by g3. So *schemas lead to weaker order constraints than micro rules*. But there are more predictions of the State Model: The application of the schemas should be faster than the application of the micro rules (bottom of Figure 5) because less control decisions have to be made by the learner (*time hypothesis*). For the same reason, we would expect less control actions and verbalizations, that is, less moving, repositioning, and rearranging of nodes and links (*rearrangement hypothesis*).

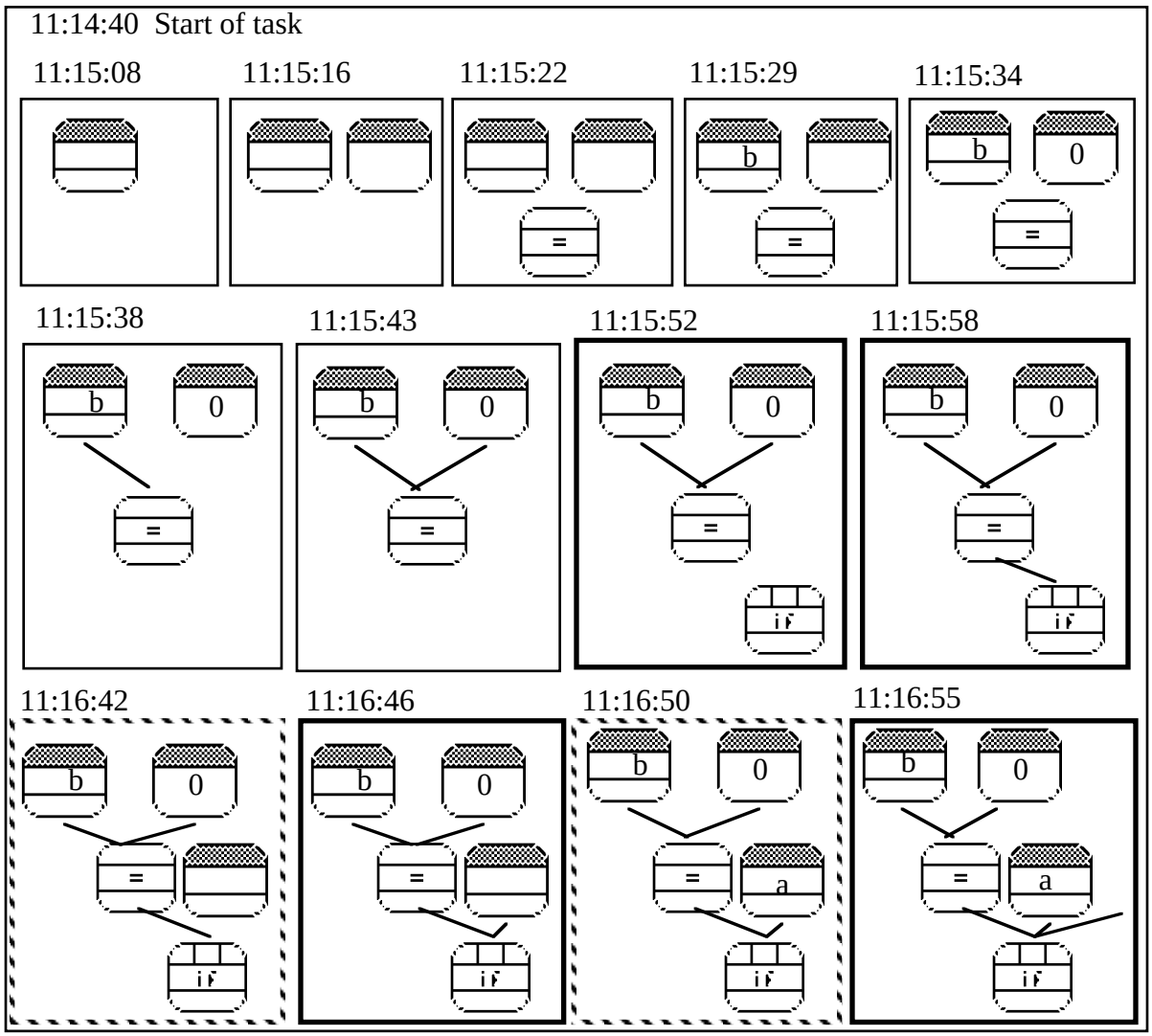

Figure 4: A continuous fragment of a sequence of programming actions performed by a subject

Some of the predictions of the State Model have been tested empirically (Möbus, Schröder, Thole, 1992). The no-interleaving hypothesis was investigated in the following way: The action and verbalization sequence of a single subject working with the ABSYNT PSM was videotyped and categorized, and the State Model was run offline based on the solutions created by the subject. This led to a sequence of consecutive hypothetical knowledge states of the subject. Based on this state sequence, it was predicted which actions and verbalizations of the subject should occur in uninterrupted sequences. This resulted in the *model trace* (Figure 6, on the right) which consists of sets. Each set of the model trace corresponds to the application of one State Model rule. For example, at the state depicted in Figure 6 the State Model contains a rule that describes three programming actions: to place an ABSYNT "product" node, and to draw two input links for it. So the actions and verbalizations of each model trace set are expected to occur in a continuous uninterrupted sequence. The *subject trace* (on the left of Figure 6) is the actually observed action and verbalization sequence. The subject trace was compared to the model trace in the following way: For each adjacent pair of events of the subject trace, if both events are contained in the same model trace set, then a "+" was assigned, otherwise a "-" was assigned.

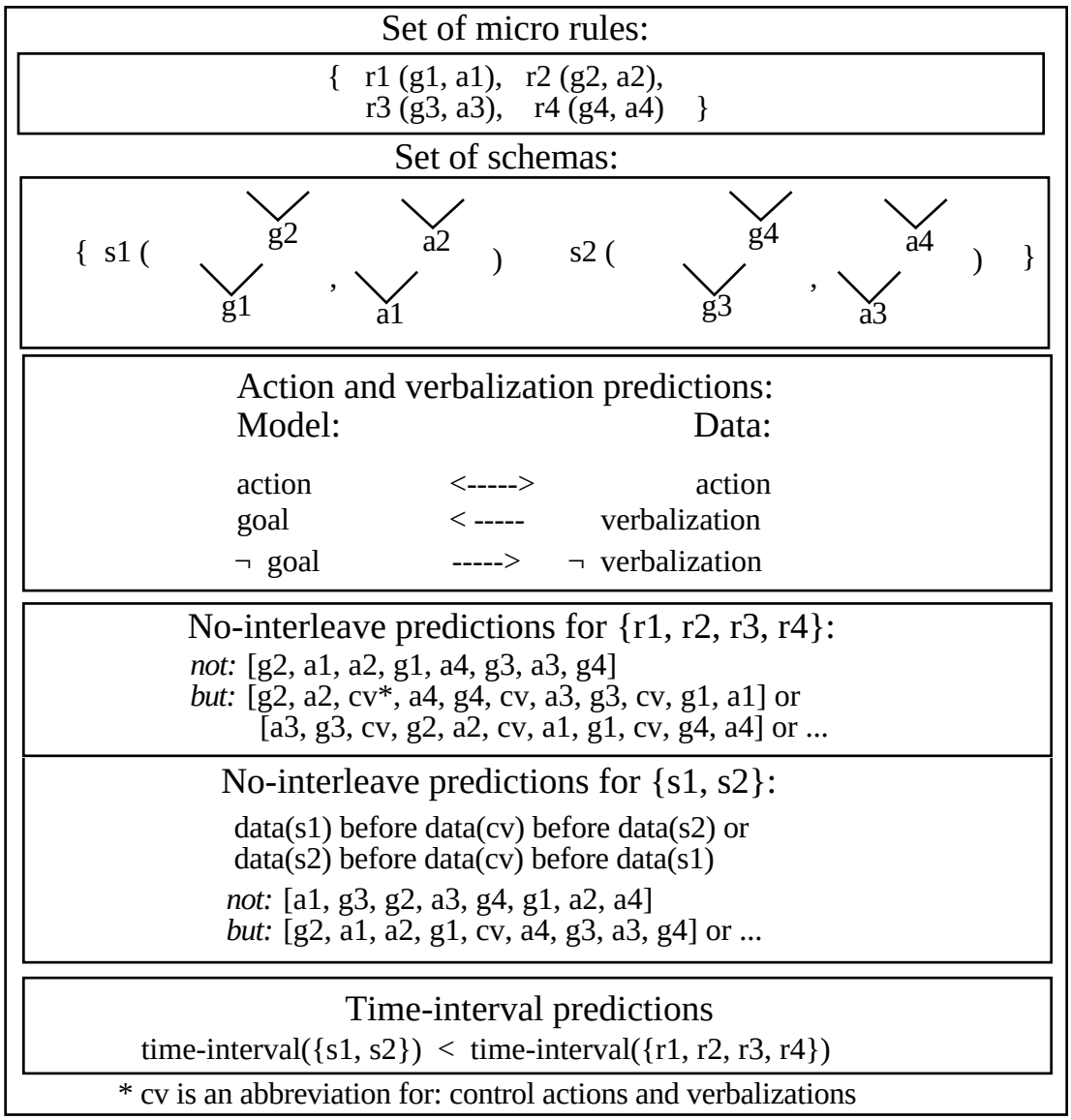

Figure 5: Empirical predictions of the State Model for micro rules vs. schemas

For example, in Figure 6 action events 36 and 37 belong to different model trace sets, so a "-" is assigned. (On the left of the subject trace, a slightly different but equivalent method of assigning "+" and "-" is shown.) So "+" denote correspondencies of model trace and subject trace, and "-" denote contradictions. The whole subject trace contained 76 "+" and 60 "-". Since more "+" should lead to longer and thus fewer runs (continuous sequences of "+" and "-") than an equal distribution of "+" and "-", the Runs test was applied. There were 42 runs ($p < 0.001$), which is consistent with the no-interleaving hypothesis. In addition, many of the discrepancies ("-") between model trace and subject trace are due to the constant and parameter node rule (see for example events 36 and 40 in Figure 6).

How can the State Model contribute to user-adapted help generation within the ABSYNT PSM? Obviously, it should select the completion proposal to present to the learner, so the learner will receive proposals which are maximally consistent with his pre-knowledge.

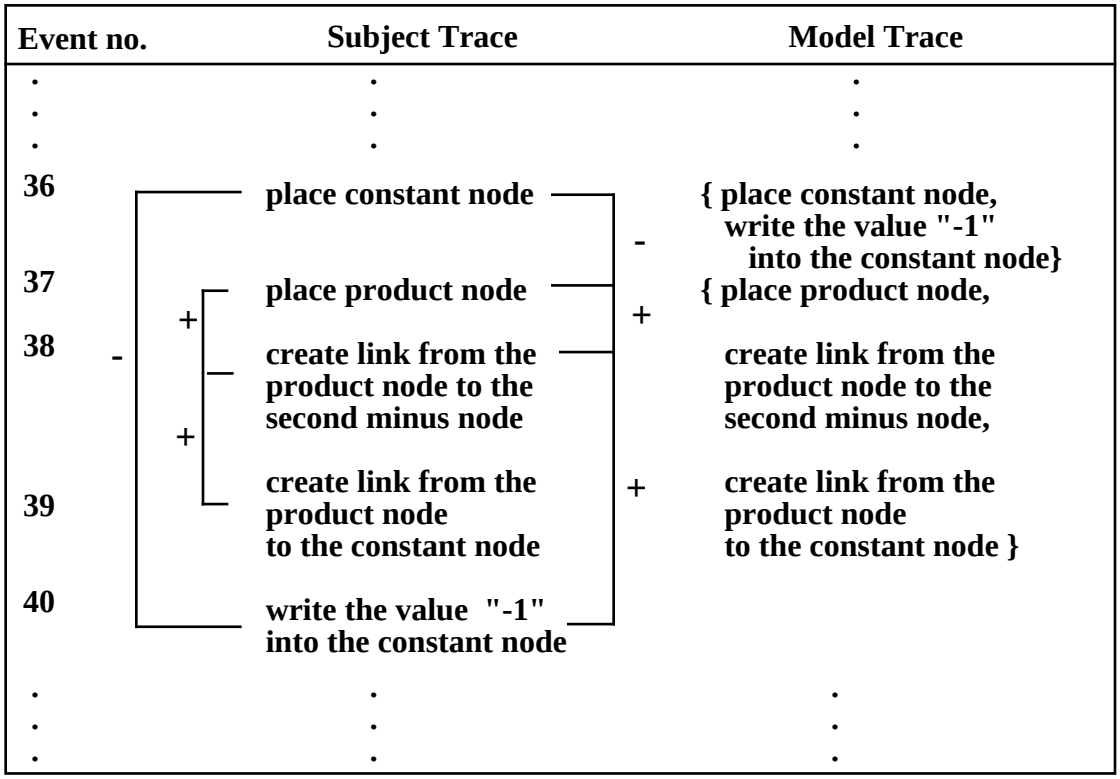

Figure 6: Empirical analysis of the State Model: Subject trace, model trace, correspondencies (+), and discrepancies (-) between them

But the State Model is also capable to identify knowledge gaps - knowledge not yet acquired by the PS but necessary to continue with the actual solution proposal. Based on such a diagnosis, new empirical predictions are possible because the completion proposals selected as help by the system can be varied according to two dimensions: *grain size* and *amount*. If the grain size is *fine*, the completion proposal may rest on a chain of micro rules covering the gap. For example, the completion proposal may consist of an ABSYNT subtree with an explanation of the goal sequence leading to it. If the grain size is *coarse*, then the completion proposal may rest on a single composite. For example, the completion proposal may consist of an ABSYNT subtree without any explanation. Concerning *amount*, the information provided may exactly *fill* the gap, or *too much* information is provided, or *not enough* information so that a part of the gap remains uncovered. Different combinations of grain size and amount lead to different hypotheses. For example:

- If the information is fine-grained and exactly fills the gap, then we would expect that the student considers this information as *helpful*.
- If the information is coarse-grained and exactly fills the gap then the student misses explanations. So s/he might either *passively accept* what is being offered, or engage in *self-explanation* (van Lehn, 1991a).

- If the information is fine-grained but exceeds the knowledge gap, then the student has to"*filter*" the content relevant to the current situation. This might be experienced as *burdensome*.

- If the information leaves a small knowledge gap, then the student might try to induce one new simple rule and thereby cover the rest of the gap. (This situation seems similar to the induction of one subprocedure at a time by van Lehn´s (1987) SIERRA program.)

- Finally, the last case to be considered here is that there is a large gap left, and the information offered is too coarse. The student should experience such an information as very inadequate to his current problem. Thus she or he should feel annoyed or even upset.

There remains much work, of course, to work out these hypotheses and put them to empirical test. But we think we have shown that the State Model is an empirically fruitful approach to knowledge diagnosis and adaptive help generation which is testable and also touches upon further research problems, like motivation and emotion.

The External Process Model

The Process Model is aimed at modelling the knowledge acquisition *processes* which hypothetically lead to the changing knowledge states as described by the State Model. It is conceptualized as a runnable realization of the ISP-DL architecture and tries to give an account of data not computer-assessable with current technology, like the learner´s verbalizations.

The precursor of the Process Model was a model of the acquisition of a cognitive skill in the domain of the operational knowledge (interpreter knowledge) about ABSYNT, based on a theoretical framework of knowledge acquisition (*impasse- and success-driven learning*) which was the precursor of the ISP-DL Theory, and empirical analyses of verbal and action protocols. The starting point for this model consisted of two alternative visual rule sets based on runnable specifications of the ABSYNT interpreter (Möbus & Thole, 1989). The visual rule sets were used as help material for subjects while acquiring the operational knowledge (Möbus & Schröder, 1989; 1990). Rule specific hypotheses concerning memory representation and working memory load were disconfirmed (Schröder et al., 1990). Therefore, a simulation model of the knowledge acquisition process was constructed and compared to a detailed protocol of verbalizations and actions (Schröder, 1990, 1992; Schröder & Kohnert, 1989/90). According to the model, new knowledge is acquired in response to impasses: Impasses trigger problem solving processes which involve the use of the help material, and existing knowledge is optimized if used successfully. The model was compared in detail to a subject trace of actions and verbalizations. It was able to hypothesize, for example, which kinds of situations lead to which kinds of weak heuristics and corresponding actions.

The external Process Model in the domain of constructing ABSYNT programs (see also Schröder & Möbus, 1992) was developed based on the ISP-DL Theory and an empirical analysis of a subject trace (actions and verbalizations) from a single subject session with ABSYNT. The analyzed part of the subject trace contained about four hours of problem solving. Figure 7 depicts the main components of the Process Model as a higher order Petri net (Huber et al., 1990). *Places* (circles / ellipses) represent states (e.g., the content of data memories, mental objects like plans or impasses, or real objects like task descriptions or solutions). *Transitions* (rectangles) represent events or process steps. The model consists of the following steps:

• The description of a programming task is represented as a text graph. The first step is to *understand* the task by creating a propositional task representation (Kintsch & Greeno, 1985; van Dijk & Kintsch, 1983) which is a hierarchy of concept names.

• Based on the task representation, a *plan* for a solution proposal is created by making use of domain knowledge. The domain knowledge is a set of concepts. Each concept has a name, an input-output description that can be calculated with example values, and (possibly) an implementation in ABSYNT. So a concept corresponds to a micro rule, a schema, or a case. But a concept may be incomplete, the ABSYNT implementation may be missing. As a simple example, the concept "multiplication of two numbers" may be known, but its implementation in ABSYNT may be unknown. (So an incomplete concept represents the pre-knowledge before working with ABSYNT.) The plan consists of instantiations of concepts of the domain knowledge.

• After the plan is created, there are several possibilities. The simplest possibility is that the plan is *executed*. Alternatively, an *impasse* might arise. This happens if the plan cannot be executed because of missing ABSYNT implementation knowledge in one or more concept instantiations. Now several *heuristics* are possible. The first one is to look for help. The help material given to subjects consisted of example calculations of the primitive ABSYNT operator nodes. So for each primitive ABSYNT operator, the concept in question is calculated with the same input values than the example calculations. If the result is the same, then the primitive ABSYNT operator is inserted into the concept instantiation, and it is stored with the original uninstantiated concept as well (acquisition of new knowledge). If no such ABSYNT operator is found, then the heuristic has failed. Still the plan cannot be executed. The next heuristic (switch focus) is tried: Another part of the plan is implemented first. Of course this does not solve the impasse, so the impasse will be encountered again. The third heuristic is restructuring. It is tried to replace the concept instantiation by another one that has the same input-output behavior. (This is checked with example values). For example, the plan fragment to change the sign of a number can be replaced by multiplication with -1. If such replacement is created, then it is stored in the knowledge base (acquisition of new knowledge), and the plan is changed accordingly. Otherwise the fourth heuristic, analogy, is tried. The analogical transfer is

basically syntactical, but in addition adjustments are made in the target solution which are again based on calculations with input-output descriptions.

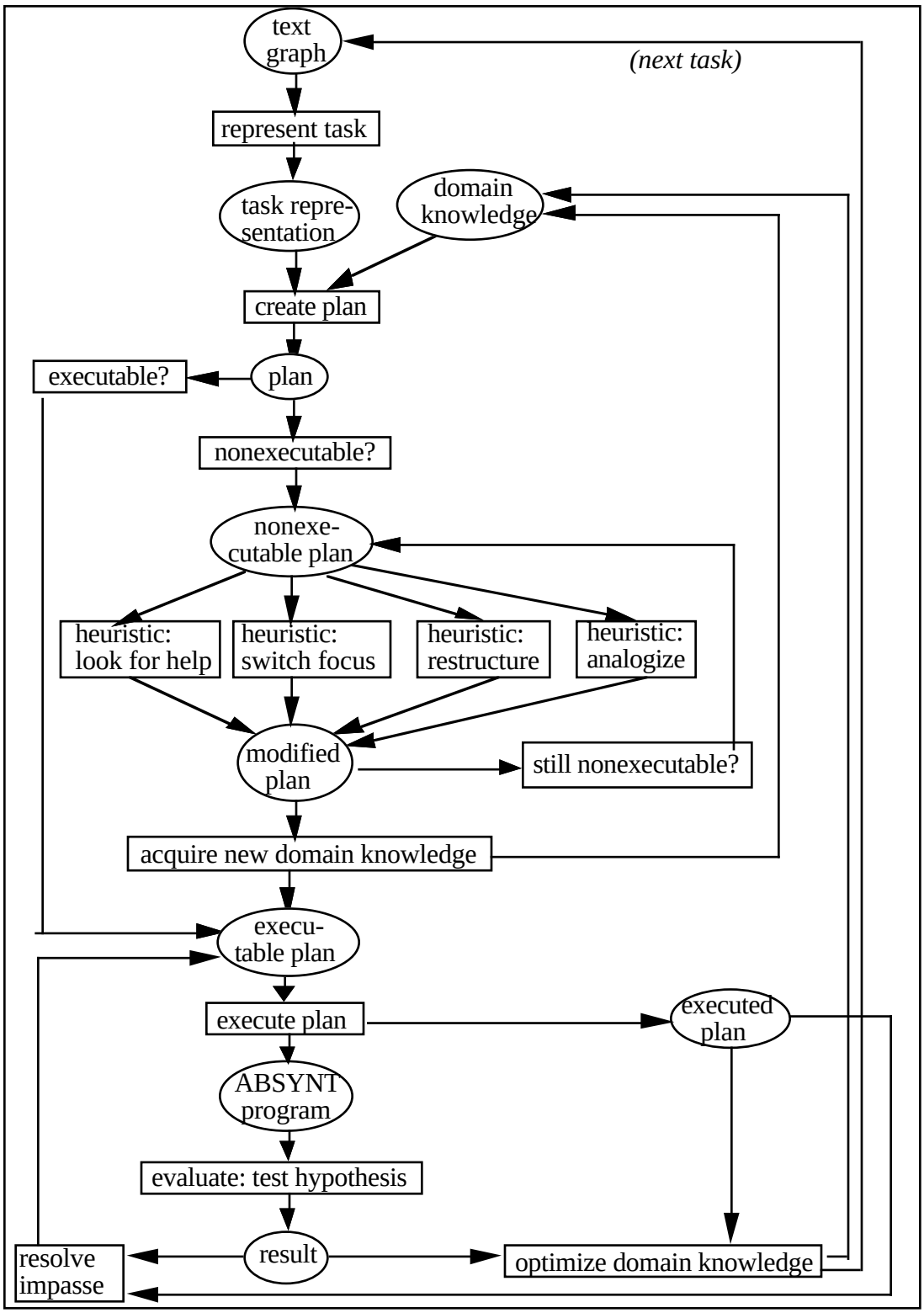

Figure 7: The external Process Model

• After an executable plan has been created and executed, the resulting ABSYNT program is *evaluated* by hypotheses testing. If the feedback is that the whole proposal is correct, then the concepts used with this task are composed, based on the executed plan, and the next task may be tried. If the feedback is that the proposal cannot be recognized by

hypothesis testing, then the executed plan is debugged (= resolve impasse). Strategies of hypotheses testing and debugging are not yet part of the model.

A trace of the model was stepwise compared to the subject trace of solving five programming tasks. One result was that 65% (about 200) protocol categories of the subject trace were reproduced by the model. What is the significance of the external Process Model for the ISP-DL Theory, the internal State Model, and for help design?

• Concerning the *ISP-DL Theory*, the Process Model specifies types of impasses, like missing implementation knowledge and negative feedback, and hypothetical weak heuristics and the conditions for their application. These hypotheses can be used to make the ISP-DL Theory more detailed and empirically testable.

• Concerning the *State Model*, the Process Model specifies when which new knowledge is acquired and optimized. So the Process Model gives hypotheses about how the knowledge contained in the State Model might be generated. The Process Model might also help to find out how to modify the State Model if its predictions are disconfirmed.

• Concerning help design, hypotheses of the Process Model about weak heuristics can be used to design domain-*un*specific, "strategical" help. For example, with the model it should be possible to recommend to the learner to look for analogies in certain situations, or to think about reformulating goals or to ask for their preconditions in other situations.

Conclusions

We described an approach to diagnose and model knowledge and knowledge acquisition processes, and to apply these results to user-adapted help. Our appraoch has three levels:

• a theoretical framework, the ISP-DL Theory, of problem solving and knowledge modification

• an internal State Model for online diagnosis and generation of user-adapted help

• an external Process Model for offline generation of hypotheses about impasses, weak heuristics, knowledge acquisition and optimization.

The three levels are intended to be mutually consistent but serve different purposes. The ABSYNT PSM designed to support the acquisition of basic functional programming skills in a visual language is a concrete realization of this approach. In line with the ISP-DL Theory, it *offers* help to the learner to make use of at impasses, it tries to be maximally consistent with the learner´s knowledge state by hypotheses testing, and it supports the problem solving states of planning, execution, and evaluation. Planning in ABSYNT is based on a transformation approach. Execution leads to runnable ABSYNT programs, and evaluation is based on the hypotheses testing approach.

The internal State Model represents knowledge states consisting of newly acquired and optimized knowledge, as required by the ISP-DL Theory. It is continuously updated based on the online analysis of small action steps. It allows a large amount of predictions concerning the interleaving of action subsequences, times, and rearrangements like moving and repositioning program fragments. Some of these predictions were empirically analyzed. The external Process Model is designed to bridge the gap between the State Model and the ISP-DL Theory by providing hypothetical reasons (impasses, weak heuristics, ...) for the knowledge state changes described by the State Model.

Further work will be directed to integrate the State Model and the Process Model into the ABSYNT PSM, to investigate the empirical predictions, and to generate *explanations* of the system for positive and negative feedback in response to hypotheses testing, and to completion proposals now provided by the ABSYNT PSM.

Our three level approach has been developed in the context of functional programming but is not constrained to it. In a related project, we develop a help system, PETRI-HELP, which is designed to support modelling concurrent or distributed processes with Petri nets (Möbus, Pitschke, Schröder, 1992). A research goal is to study the question how much of the theory and models is domain independent.

References

Anderson, J.R., The Architecture of Cognition. Cambridge: Harvard University Press, 1983

Anderson, J.R., Knowledge Compilation: The General Learning Mechanism, in Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M.(eds), Machine Learning, Vol. II. Los Altos: Kaufman, 1986, 289-310

Anderson, J.R., A Theory of the Origins of Human Knowledge. Artificial Intelligence, 40, 1989, 313-351

Anderson, J.R., Conrad, F.G., Corbett, A.T., Skill Acquisition and the LISP Tutor, Cognitive Science, 1989, 13, 467-505

Anderson, J.R., Swarecki, E., The Automated Tutoring of Introductory Computer Programming, Communications of the ACM, 1986, 29, 842-849

Bauer, F.L., Ehler, H., Horsch, A., Möller, B., Partsch, H., Paukner, O., Pepper, P., The Munich Project CIP, Vol. II: The Program Transformation System CIP-S, Berlin: Springer (LNCS 292), 1987

Bauer, F.L., Goos, G., Informatik (Vol. 1), Berlin: Springer, 1982 (3rd ed.)

Brown, J.S., Burton, R.R., Diagnosing Bugs in a Simple Procedural Skill, in D. Sleeman, J.S. Brown, Intelligent Tutoring Systems, New York: Academic Press, 1982, 157-183

Chang, S.K. (ed), Principles of Visual Programming Systems, Englewood Cliffs: Prentice Hall, 1990

Colonius, H., Frank, K.D., Janke, G., Kohnert, K., Möbus, C., Schröder, O., Thole, H.-J., Stand des DFG-Projekts "Entwicklung einer Wissensdiagnostik- und Fehlererklärungskomponente beim Erwerb von Programmierwissen für ABSYNT", in: R. Gunzenhäuser & H. Mandl (Hrsgb), "Intelligente Lernsysteme", 80 - 90, 1987, Institut für Informatik der Universität Stuttgart & Deutsches Institut für Fernstudien an der Universität Tübingen

Corbett, A.T., Anderson, J.R., Patterson, E.J., Problem Compilation and Tutoring Flexibility in the LISP Tutor, Procedings of the First Int. Conf. on Intelligent Tutoring Systems ITS-88, Montreal, 423-429

Corbett, A.T., Anderson, J.R., Student Modeling and Mastery Learning in a Computer-Based Programming Tutor, in C. Frasson, G. Gauthier, G.I. McCalla (eds), Intelligent Tutoring Systems (Proceedings ITS 92), Berlin: Springer, 1992 (LNCS 608), 413-420

Glinert, E.P., Nontextual Programming Environments, in Chang (ed), Principles of Visual Programming Systems, Englewood Cliffs: Prentice Hall, 1990, 144-230

Gollwitzer, P.M., Action Phases and Mind-Sets, in: E.T. Higgins & R.M. Sorrentino (eds), Handbook of Motivation and Cognition: Foundations of Social Behavior, 1990, Vol.2, 53-92

Gollwitzer, P.M., Abwägen und Planen, Göttingen, Toronto: Verlag für Psychologie, 1991

Greer, J., Granularity and Context in Learning, University of Saskatchewan, Saskatoon, Canada, 1992

Greer, J., McCalla, G.I., Mark, M.A., Incorporating Granularity-Based Recognition into SCENT, Proceedings 4th Int. Conference on Artificial Intelligence and Education, Amsterdam, 1989

Hogger, Ch.J., Essentials of Logic Programming, Oxford University Press, 1990

Huber, P., Jensen, K., Shapiro, R.M., Hierarchies in Coloured Petri Nets, in G. Rozenberg (ed.), Advances in Petri Nets 1990, LNCS, Heidelberg: Springer

Kintsch, W., Greeno, J. G., Understanding and Solving Word Arithmetic Problems. Psych. Review, 92, 1, 1985, 109-129

Kohnert, K., Janke, G., The Environments of ABSYNT: A Problem Solving Monitor for a Functional Visual Programming Language. ABSYNT-Report 4/88, Oldenburg, 1988 (revised 1991)

Kowalski, R., Logic for Problem Solving, Amsterdam: Elsevier Science Publ., 1979

Laird, J.E., Rosenbloom, P.S., Newell, A., Universal Subgoaling and Chunking. The Automatic Generation and Learning of Goal Hierarchies, Boston: Kluwer, 1986

Laird, J.E., Rosenbloom, P.S., Newell, A., SOAR: An Architecture for General Intelligence, Artificial Intelligence, 1987, 33, 1-64

Larkin, J.H., Simon, H.A., Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive, Science, 11, 1987, 65-99

Levi G., Sirovich, F., Generalized And/Or-Graphs. Artificial Intelligence, 1976, 7, 243-259

Lewis, C., Composition of Productions, in Klahr, D., Langley, P., Neches, R. (eds), Production, System Models of Learning and Development. Cambridge: MIT Press, 1987, 329-358

Möbus, C., Toward the Design of Adaptive Instructions and Helps for Knowledge Communication with the Problem Solving Monitor ABSYNT. in: V. Marik, O. Stepankova & Z. Zdrahal (eds): Artificial Intelligence in Higher Education, Proceedings of the CEPES UNESCO International Symposium Prague, CSFR, October 23 - 25, 1989, Berlin - Heidelberg - New York: Springer, Lecture Notes in Artificial Intelligence, Vol.451, 1990, 138 - 145

Möbus, C., The Relevance of Computational Models of Knowledge Acquisition for the Design of Helps in the Problem Solving Monitor ABSYNT, in R.Lewis & S.Otsuki (eds), Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education Tokyo, Japan, 18-20 July, 1990, Elsevier Science Publishers B.V. (North-Holland), 1991a, 137-144

Möbus, C., Wissenserwerb mit kooperativen Systemen, in: P. Gorny (Hrsgb), Informatik und Schule 1991, Informatik: Wege zur Vielfalt beim Lehren und Lernen, GI-Fachtagung, Oldenburg, Oktober 1991, Proceedings, Berlin: Springer (Informatik-Fachberichte 292), 1991b, 288 - 298

Möbus, C., Pitschke, K., Schröder, O., Towards the Theory-Guided Design of Help Systems for Programming and Modelling Tasks, in C. Frasson, G. Gauthier, G.I. McCalla (eds), Intelligent Tutoring Systems, Proceedings ITS 92, Berlin: Springer (LNCS 608), 1992, 294-301

Möbus, C., Schröder, O., Knowledge Specification and Instructions for a Visual Computer Language, in: F. Klix, N.A. Streitz, Y. Waern & N. Wandke (eds), MACINTER-II Man-Computer-Interaction Research, Proceedings of the Second Network Seminar of MACINTER held in Berlin/GDR, March 21 - 25, 1988, Amsterdam: North Holland, 1989, 535 - 565

Möbus, C., Schröder, O., Representing Semantic Knowledge with 2-dimensional Rules in the Domain of Functional Programming, in: P.Gorny & M. Tauber (eds), Visualization in Human-Computer Interaction, 7th Interdisciplinary Workshop in Informatics and Psychology, Schärding. Austria, May 1988; Lecture Notes in Computer Science, Vol. 439, Berlin-Heidelberg-NewYork: Springer, 1990, 47-81

Möbus, C., Schröder, O., Thole, H.-J., Runtime Modeling the Novice-Expert Shift in Programming Skills on a Rule-Schema-Case Continuum, in: J. Kay; A. Quilici (eds), Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction, 12th Int. Joint Conf. on Artificial Intelligence, Darling Harbour, Sydney, Australia, 24-30 August 1991, 137-143

Möbus, C., Schröder, O., Thole, H.-J., Diagnosing and Evaluating the Acquisition Process of Problem Solving Schemata in the Domain of Functional Programming, to appear in G. McCalla (ed), Student Modelling: The Key to Individualized Knowledge-Based Instruction, in press

Möbus, C., Thole, H.-J., Tutors, Instructions and Helps, in: Christaller, Th. (ed), Künstliche Intelligenz KIFS 1987, Informatik-Fachberichte 202, Heidelberg: Springer, 1989, 336 - 385

Möbus, C., Thole, H.-J., Interactive Support for Planning Visual Programs in the Problem Solving Monitor ABSYNT: Giving Feedback to User Hypotheses on the Basis of a Goals-Means-Relation, in: D.H. Norrie, H.-W. Six (eds), Computer Assisted Learning. Proceedings of the 3rd International Conference on Computer-Assisted Learning ICCAL 90, Hagen, F.R.Germany, Lecture Notes in Computer Science, Vol. 438, Heidelberg: Springer, 1990, 36-49

Neves, D.M., Anderson, J.R., Knowledge Compilation: Mechanisms for the Automatization of, Cognitive Skills, in Anderson, J.R. (ed), Cognitive Skills and their Acquisition. Hillsdale, Erlbaum, 1981, 57-84

Newell, A., The Knowledge Level. Artificial Intelligence, 1982, 18, 87-127

Newell, A., Unified Theories of Cognition, Cambridge: Harvard University Press, 1990

Nilsson, N.J., Principles of Artificial Intelligence. Palo Alto: Tioga Publ. Co., 1980

Partsch, H.A., Specification and Transformation of Programs: A Formal Approach to Software Development, Berlin: Springer, 1990

Pomerantz, J.R., Perceptual Organization in Information Processing, in Aitkenhead, A.M., Slack, J.M. (eds), Issues in Cognitive Modeling. Hillsdale: Erlbaum, 1985, 127-158

Rosenbloom, P.S., Laird, J.E., Newell, A., McCarl, R., A Preliminary Analysis of the SOAR Architecture as a Basis for General Intelligence, Artificial Intelligence, 1991, 47, 289-305

Schröder, O., A Model of the Acquisition of Rule Knowledge with Visual Helps: The Operational Knowledge for a Functional, Visual Programming Language, in: D.H. Norrie, H.-W. Six (eds), Computer Assisted

Learning. Proceedings of the 3rd International Conference on Computer-Assisted Learning ICCAL 90, Hagen, F.R.Germany, Lecture Notes in Computer Science, Vol. 438, Heidelberg: Springer, 1990, 142-157

Schröder, O., Erwerb von Regelwissen mit visuellen Hilfen: Das Semantikwissen für eine graphische, funktionale Programmiersprache. Frankfurt: Lang, 1992

Schröder, O., Frank, K.D., Colonius, H., Gedächtnisrepräsentation funktionaler, graphischer Programme, ABSYNT-Report 1/87, Oldenburg, 1987

Schröder, O., Frank, K.D., Kohnert, K., Möbus, C., Rauterberg, M., Instruction-Based Knowledge Acquisition and Modification: The Operational Knowledge for a Functional, Visual Programming Language, Computers in Human Behavior, Vol. 6, 1990, 31-49

Schröder, O., Kohnert, K., Toward a Model of Instruction-Based Knowledge Acquisition: The Operational Knowledge for a Functional, Visual Programming Language. Journal of Artificial Intelligence and Education, Vol. 1, No. 2, 1989/90, 105-128

Schröder, O., Möbus, C., Zur Modellierung des hilfegeleiteten Wissenserwerbs beim Problemlösen, in K. Reiss, M. Reiss, H. Spandl (Hrsg), Maschinelles Lernen - Modellierung von Lernen mit Maschinen, Berlin, Heidelberg: Springer, 1992, 23-62

Van Dijk, T.A., Kintsch, W., Strategies of Discourse Comprehension. New York: Academic Press, 1983

Van Lehn, K., Learning One Subprocedure per Lesson, Artificial Intelligence, 1987, 31, 1-40

Van Lehn, K., Toward a Theory of Impasse-Driven Learning. In: Mandl, H.; Lesgold, A. (eds): Learning Issues for Intelligent Tutoring Systems. New York: Springer, 1988, 19-41

Van Lehn, K., Learning Events in the Acquisition of Three Skills, Proc. 11th Conf. Cognitive Science Society, Ann Arbor, Michigan, 1989, 434-441

Van Lehn, K., Mind Bugs: The Origins of Procedural Misconceptions, Cambridge: MIT Press, 1990

Van Lehn, K., Two Pseudo-Students: Applications of Machine Learning to Formative Evaluation, in R. Lewis, S. Otsuki (eds), Advanced Research on Computers in Education ARCE 90, Elsevier IFIP, 1991a, 17-25

Van Lehn, K., Rule Acquisition Events in the Discovery of Problem Solving Strategies, Cognitive Science, 1991b, 15, 1-47

Vere, S.A., Relational Production Systems, Artificial Intelligence, 1977, 8, 47-68

Weber, G., Cognitive Diagnosis and Episodic Modeling in an Intelligent LISP Tutor, Proceedings Intelligent Tutoring Systems ITS 88, 207-214

Weber, G., Automatische kognitive Diagnose in einem Programmier-Tutor, in D. Metzing (ed), Künstliche Intelligenz GWAI 89, Berlin: Springer, 1989, 331-336

Weber, G., Analogien in einem fallbasierten Lernmodell, in K. Reiss, M. Reiss, H. Spandl (Hrsg), Maschinelles Lernen - Modellierung von Lernen mit Maschinen, Berlin, Heidelberg: Springer, 1992, 143-175

Winkels, R., Breuker, J., Discourse Planning in Intelligent Help Systems, in C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, 124-139

Wolff, J.G., Cognitive Development as Optimisation, in Bolc, L. (ed), Computational Models of Learning. Berlin: Springer, 1987, 161-205

Wolff, J.G., Towards a Theory of Cognition and Computing, Chichester: Ellis Horwood, 1991