Towards the Theory-Guided Design of Help Systems for Programming and Modelling Tasks

Claus Möbus, Knut Pitschke, Olaf Schröder

University of Oldenburg, Dept. of Computational Science
Unit on Tutoring and Learning Systems,
D - W2900 Oldenburg, Germany
Claus.Moebus@arbi.informatik.uni-oldenburg.de

Abstract. This paper describes an approach to the design of online help for programming tasks and modelling tasks, based on a theoretical framework of problem solving and learning. The framework leads to several design principles which are important to the problem of when and how to supply help information to a learner who is constructing a solution to a given problem. We will describe two example domains where we apply these design principles: The ABSYNT problem solving monitor supports learners with help and proposals for functional programming. The PETRI-HELP system currently under development is intended to support the learning of modelling with Petri nets.

1. Introduction

It has been well recognized that developing intelligent help systems raises difficult questions, like: How is help and instructional material to be designed? When should remedial information be supplied, and not be supplied? Why might the same information be useless to one person and helpful to another? Existing intelligent tutorial and help systems have not always provided satisfactory answers to such questions. For example, the information delivered to the learner may assume too little or too much knowledge, or the user interaction is too restrictive, or tutoring and help strategies are unprincipled and ad hoc [25]. These shortcomings are basically still true [24, 31]. In order to move forward, a theoretical framework is necessary which incorporates problem solving and learning. It should be detailed enough to enable specific design decisions and predictions. At the same time it should be general enough to be applicable to different domains.
We work on such a framework, ISPDL Theory (*i*mpasse - *s*uccess - *p*roblem solving - *d*riven *l*earning). It is an attempt to integrate the theoretical concepts of *impasse-driven learning* [7, 14, 15, 26, 27], s*uccess driven learning* [1, 2, 4, 16, 33], and *problem solving phases* or action phases [11, 12]. One purpose of ISPDL Theory is to obtain a set of design criteria for intelligent help systems which support problem solvers while planning and constructing solutions. In order to make these design criteria as domain independent as possible, we apply them to two different domains: The ABSYNT problem solving monitor supplies help for functional programming. The PETRI-HELP system is designed to support Petri net planners. In this paper we will describe ISPDL Theory, the design principles implied, and how they are to be realized in ABSYNT and PETRI-HELP.

2. The ISPDL Theory of Problem Solving, Acquisition and Improvement of Knowledge

From empirical investigations we concluded that it is fruitful to describe learning in the domain of functional programming as an interplay of impasse- and success-driven learning. In particular, we developed a model based on these concepts which closely simulates the continuous stream of actions and verbalizations of a single subject while acquiring the

interpreter knowledge about functional programs [23]. Further development led to the ISPDL Theory [20] which is intended to describe the stream of actions and cognitive processes occurring in problem solving situations. ISPDL Theory has three aspects:

• The distinction of *different problem solving phases* [11]. In the *deliberate* phase the problem solver considers several goals and finally chooses one. In the *plan* phase a solution plan is developed to obtain the goal. Subgoals are created and sequenced. Then the plan is executed, or *implemented*. Finally the problem solver *evaluates* the result.

• The *impasse driven acquisition of new knowledge*. In response to an impasse, the problem solver applies weak heuristics, like asking questions and looking for help [26, 27, 28]. Thus the learner obtains new information. As a result of this, the learner may overcome the impasse and acquire new knowledge. Thus impasses trigger the *acquisition* of knowledge. But the new information may cause a secondary problem [7, 10].

• The *success driven improvement of existing knowledge*. Successfully used knowledge is improved. By rule composition [2, 16, 22, 30], which can be based on the resolution method [20], the number of control decisions and subgoals to be set is reduced.

3. Principles for Help Design Based on ISPDL Theory

The ISPDL Theory motivates the following principles for providing help to the learner:

1. The help system should not interrupt the learner (see also [32]) but *offer* information, because according to the theory, information is only helpful at impasse time [26]. So information is only to be supplied on request by the learner. This principle is somewhat opposed to the principle of immediate feedback [3, 5]. But it is implied by the theory, and we think that it is important to let the learner develop her /his own solution ideas even if they seem strange from an expert point of view.

2. The learner must have the opportunity to obtain *detailed feedback and information* at every time impasses may arise in problem solving. Since different impasses are possible at *different levels* of problem solving, the system must offer support in the problem solving phases of *planning*, *implementation*, and *evaluation*.

3. The learner should be enabled to *make use of her/his pre-knowledge* as much as possible when asking for help, so the information provided as help does not suggest different solution plans and thus cause secondary impasses. Rather, help should accept the learner´s solution plan and provide the learner with requested information as precisely as possible.

4. The provided information should be *tailored to the actual knowledge state of the learner*. If the information presupposes too much pre-knowledge, the learner will encounter a secondary impasse. This might lead to self explanation [8, 29] of the information obtained, but also to non-understanding and to negative emotions. If the information assumes too little pre-knowledge, then the learner will get bored by things already known. So whether information is helpful depends on the actual knowledge state of the learner. A *state model* is needed to represent online the actual hypothetical domain knowledge state of the learner. Its two main functions are to *control* the analysis of solution proposals of the learner, and to determine which help information to *choose* in case of several possibilities.

5. The state model should be embedded in a *process model*. The latter models the *processes of knowledge acquisition and modification, the application of weak heuristics and control processes*. It may make use of additional data, like verbalizations. One of the functions of the process model is to support the development of the more restricted state model which must be empirically valid since it is used for diagnosis and help generation.

6. It is necessary that *the learner is free in the choice and sequencing of her/his interactions* with the system. The more restricted the range of the learner´s actions is, the less information can be obtained for inferring cognitive states (modelled by the state model) and processes.

4. Two Examples

The six design principles and the ISPDL Theory were developed in the context of ABSYNT, a visual language and a help system for functional programming. We started to apply them to the design of PETRI-HELP as a second domain.

4.1. ABSYNT

ABSYNT ("Abstract Syntax Trees") [17, 19, 21] was developed from ideas stated in a computer science textbook [6]. It consists of a functional, visual programming language (comparable to pure LISP without the data list structure) and a Problem Solving Monitor supporting programming novices with help and proposals while they acquire functional programming concepts up to recursion. ABSYNT was designed to encourage explorative but help-guided learning. The ABSYNT system consists of four main parts [19, 21]:

• A *visual editor* for constructing programs. ABSYNT programs consist of trees built from connected primitive and self-defined operator nodes, parameters, and constants.

• A *visual trace* makes each computational step of the ABSYNT interpreter visible.

• In a *diagnosis-, hypotheses- and help environment* the learner may state the hypothesis that her/his solution proposal (or part of that proposal) to a programming task is correct. The system then analyzes the part of the solution proposal chosen by the student as a hypothesis. As the result, the system gives help and error feedback on the *implementation level* by synthesizing complete solutions for the given programming tasks, starting from the learner´s hypothesis. If the hypothesis is embeddable within a complete solution, the learner may ask for completion proposals. Figure 1 shows

- a learner´s wrong solution proposal to the "even" task (upper window)
- a hypothesis stated by the learner (bold parts of the proposal in the upper window)
- feedback that this hypothesis is embeddable within a correct solution (copy of the hypothesis in the lower window), and
- a completion proposal generated by the system (bold parts of lower window, i.e., the "¬" operator node in Figure 1). The system generates a complete solution but only one node is shown to the learner in order not to terminate problem solving. The amount and content of completions will depend on the state model (see below).

A set of currently 622 diagnostic rules defining a goals-means-relation [17, 21] analyzes and synthesizes several millions of solution proposals for 40 programming tasks. The rules generate complete solutions, recognize incomplete proposals, and complete them.

One reason for the *hypotheses* approach [17, 20, 21] is that in programs bugs cannot be absolutely localized. Thus we leave the decision which parts of a buggy solution proposal to keep to the learner. Single subject sessions with the ABSYNT Problem Solving Monitor 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. We work on a *process* model to simulate the protocol of a single subject, in order to shed light on the structure of sequences of hypotheses and their role in the knowledge acquisition process.

Currently we extend the ABSYNT language to support program construction and help generation on the *planning level* by introducing goal nodes. The learner will be able to test hypotheses and receive error and completion feedback on this planning level in a similar way as on the implementation level, as described above. Thus the learner may first *plan* a goal tree for the task at hand, test hypotheses about it and debug it, if necessary. Afterwards the learner may *implement* the goals by replacing them with ABSYNT nodes. So *mixed* trees containing goal nodes and ABSYNT nodes will be possible as well.

• A learner model ("*state model*", [20]) controls knowledge diagnosis and help generation. It is implemented but not yet integrated into the system. It represents the hypothetical state of domain knowledge of the learner. It is updated based on the learner´s programming actions and the times between them and gives rise to several empirical predictions [20].

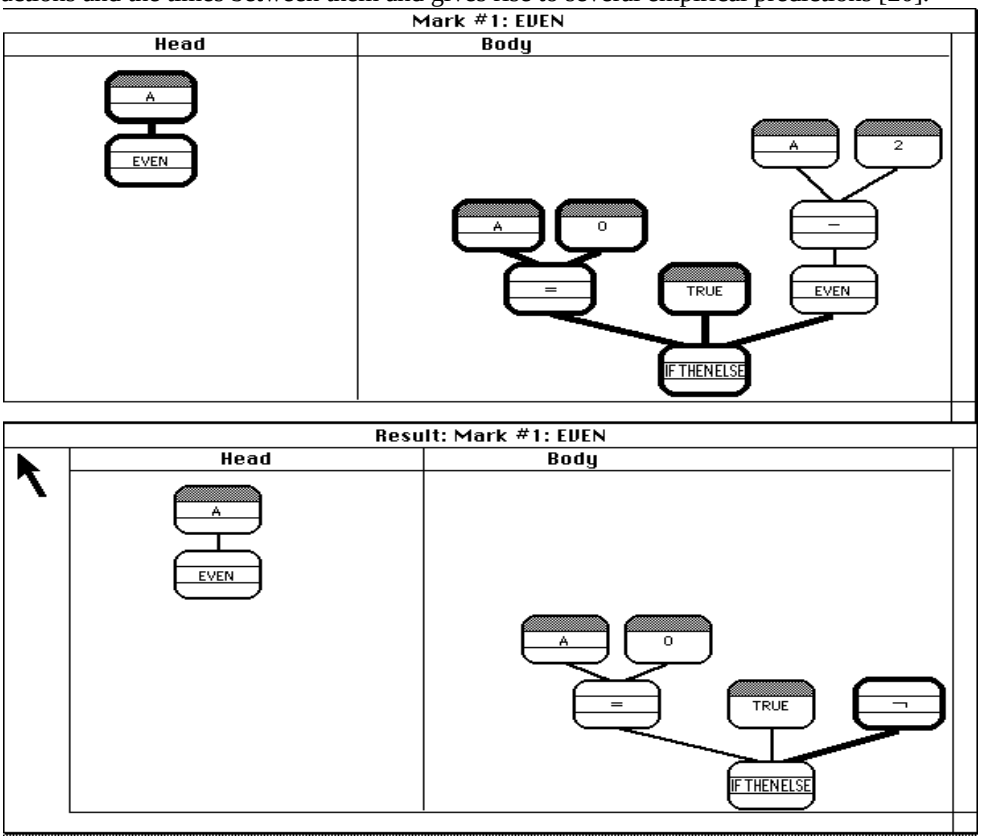

Figure 1: A hypothesis and a completion proposal in the ABSYNT environment

How are the six design principles stated above incorporated into ABSYNT?

• Concerning design principle 1: The system does not interrupt the learner but *offers* help. The learner is free to state hypotheses and ask for help at any time.

• Concerning design principle 2: The system is designed to deliver support at impasses on different problem solving levels. Programming is possible at the *implementation level* with operator nodes, parameters, and constants. It will be possible at the *planning level* with goal nodes. The system provides help at the *implementation level* (hypothesis testing and obtaining completion proposals) and will provide similar help at the *planning level*. Testing hypotheses is also a powerful means for *evaluating* one´s solution proposal, as empirical data have shown. The visual trace also supports evaluation. So ABSYNT supports the three problem solving phases of planning, implementation, and evaluation.

• Concerning design principle 3: By stating hypotheses, the learner is enabled to *make use of her/his pre-knowledge*. The learner, not the system selects the parts of the solution proposal to be retained if corrections are necessary.

• Concerning design principle 4: Help information is *tailored to the actual knowledge state* of the learner since help is based on the state model of the learner´s actual domain

knowledge. In accordance with ISPDL Theory, the state model contains knowledge *acquired* by heuristics (not in the state model) and knowledge *optimized* by composition.
• Concerning design principle 5: We work on a *process model* to simulate two single subjects´ knowledge acquisition, impasses, and subsequent problem solving heuristics.
• Concerning design principle 6: Due to the two dimensional layout of the ABSYNT editor, positioning, naming, moving, deleting, connecting and unconnecting nodes are *freely arrangeable actions* in the visual ABSYNT editor.

4.2. PETRI-HELP

In the PETRI-HELP project [18], a system is developed for supporting problem solvers in the domain of modelling with condition-event Petri nets. Like in ABSYNT, the system will provide help sensitive to the actual knowledge state of the learner. But there are differences to ABSYNT due to the special demands of the Petri net domain:

- specification of the tasks the user is supposed to solve
- the kind of analysis of the learner´s solution proposals
- creation of design rules on which the help information is based

• *Specification of tasks.* For Petri nets the task specification is more complicated than in ABSYNT. Furthermore, in the domain of Petri nets it is unusual to formally specify the problem to be solved. An exception is e.g. [13] where temporal-logic formulae are used to verify computer architecture descriptions, the semantics of which are specified by Petri nets. Temporal logic specifications enable the verification of learners´ Petri net proposals by model checking [9]. So we developed 15 task descriptions by sets of temporal-logic formulae. Figure 2 shows the temporal logic specification and an empirical solution proposal for one task, "Bavarian biergarten".

• *Analyzing the learners´ solution proposals*. We developed a simple model checker for the diagnosis of the user´s solutions in PETRI-HELP. The diagnosis is based on the case graph of the Petri net. In that graph, the temporal-logic formulae of the specification are verified. Thus it is possible to detect the set of formulae which is fulfilled by a user-created net. The model checker may be used after every editing step done by the user in order to determine the set of formulae fulfilled. Figure 3 shows a part of the case graph for the net in Figure 2. The paths in the graph describe the different orders of firing transitions in the net. The user specifies an initial set of tokens spread over the places. This set corresponds to the initial state of the case graph, where interpretation of the formulae starts:
- If the formula contains no temporal-logic operator (O, ◊, [], see Figure 2), then it is a propositional-logic formula and will be evaluated inside the current node of the case graph.
- If the formula has the pattern O F (F is a formula), then F must hold in every immediate successor of the current state in the case-graph.
- ◊ F is true iff in every path leaving the current node F will be true at least in one node.
- Finally, [] F holds iff F holds in every state on every path leaving the current state.

• *Design rules and help information*. We developed two kinds of design rules to support learners. This help will be created *by the system* in response to the learners´ actions:
- Based on the *model checker*, there are design rules proposing completions to the existing net such that a *superset* of the currently fulfilled formulae is fulfilled.
- *Empirical* design rules relate formulae to net parts. They result from empirical studies with 14 subjects working in single-subject sessions on our tasks. For each task, the set of formulae was given to the subjects with the instruction to create a Petri net. The subjects indicated whenever they considered fomulae as fulfilled while constructing the nets.

In the help system the learner will be able to choose from design rules. The actual solution proposal will be completed by the system according to the choice of the learner. A *third* kind of rules will incorporate a *goal* level. The learner will be enabled to plan aspects such as "parallelism", "mutual exclusion" etc. The learner will also be able to state *hypotheses*. The system then lists the satisfied formulae and completes the net. A *learner model* representing the actual knowledge state will control which help is actually offered.

Ws : Waiter is sleeping	[] (Ws → ◊ Wro)	[] (¬(Ws Λ Wro))
Wro : Waiter is ready	[] (R Λ Ws → ◊ Wrs)	[] (¬(Ws ΛWrs))
to accept order	[] (Wro → ◊ (Ws Λ K))	[] (¬(Wro Λ Wrs))
Wrs : Waiter is ready	[] (K → ◊ P)	[] (Ws v Wro v Wrs)
to serve	[] (R Λ Wro → ◊ Wrs)	[] (Wrs → ◊ Ws)
K : Kitchen got order		[] (P → ◊ R)
P : Preparation		
R : Meal is ready	O means "nexttime", ◊ means "eventually", [] means "always"	

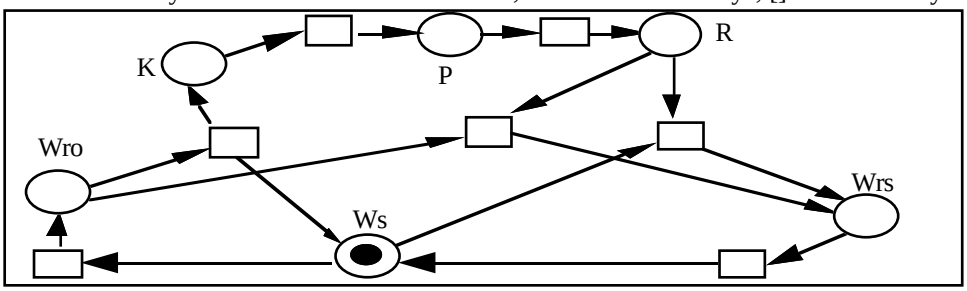

Figure 2: Specification and solution to the task "Bavarian biergarten"

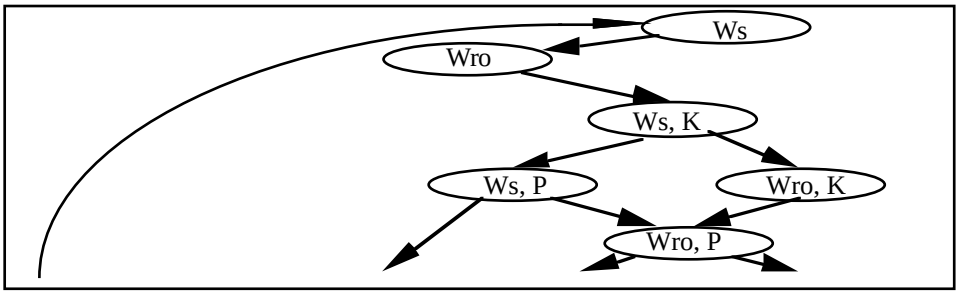

Figure 3: A part of the case graph for the solution of Figure 2

How are the six design principles stated above incorporated into PETRI-HELP?

• Concerning design principle 1: Like ABSYNT, PETRI-HELP is intended to *offer* help.

• Concerning design principle 2: *Detailed feedback* concerning *planning*, *implementation*, and *evaluation* of nets are to be provided after every editing step. Rules for intermediate concepts (i.e., parallelism) address *planning*. Design rules suggesting completions or revisions to the current net address *implementation*. Model checking (to determine the formulae that hold in the current net) and hypotheses testing address *evaluation*.

• Concerning design principle 3: The selection of rules, hypothesis testing, and receiving proposals of possible completions are to be under *control of the learner*.

• Concerning design principles 4 and 5: The system will learn rules from the learners´ actions. Thus a *state model* is constructed in order to determine the actual information to

be offered to the learner. Like in ABSYNT, a *process model* will simulate the evolution of knowledge structures as contained in the state model.
• Concerning design principle 6: Due to the two dimensional layout, the user freely chooses net editing actions. He may let the system create places, arcs, and transitions by choosing among design rules. But he may also perform these editing actions by himself.

5. Conclusions

The table summarizes the design principles following from ISPDL Theory, and the features of the design of ABSYNT and PETRI-HELP corresponding to these principles. We think that ISPDL Theory is a promising approach to the design of help systems.

ISPDL Design Principles:	ABSYNT:	PETRI-HELP:
1. "Do not interrupt the learner - offer help"	Learner can always ask for help	Learner can always ask for help
2. "Provide detailed information for phases where impasses can occur: planning, implementation, and evaluation"	Editing, hypoth. testing, and completion proposals on the planning level and implementation level. Hypotheses testing and visual trace for evaluation	Editing, hypotheses testing, and completion proposals on the planning level (planning rules) and implementation level (design rules). Hypotheses testing and model checking for evaluation
3. "Let the learner use her/his pre-knowledge"	Stating hypotheses	Stating hypotheses Selecting design rules
4. "Tailor information to the knowledge state of the learner"	State model controlled completions to hypotheses	State model controlled completions and design rules offered to the learner
5. "Embed the state model in a process model"	Process model: Impasses, heuristics, knowledge acquisition processes	Process model: Impasses, heuristics, knowledge acquisition processes
6. "Provide freedom in the learner´s actions"	Free arrangement of positioning, naming, moving, deleting, connecting, unconnecting nodes (two dimensional layout)	Free arrangement of positioning, naming, moving, deleting, connecting, unconnecting places and transitions (two dimensional layout)

References

1. J.R. Anderson: The Architecture of Cognition. Cambridge: Harvard University Press, 1983
2. J.R. Anderson: Knowledge Compilation: The General Learning Mechanism. In: R.S. Michalski, J.G. Carbonell, T.M. Mitchell, Machine Learning II. Kaufman, 1986, 289-310
3. J.R. Anderson: Production Systems, Learning, and Tutoring, in D. Klahr, P. Langley, R. Neches (eds): Production System Models of Learning and Development. Cambridge: MIT Press, 1987, 437-458
4. J.R. Anderson: A Theory of the Origins of Human Knowledge, Artificial Intelligence, 1989, 40, 313-351
5. J.R. Anderson, F.G. Conrad, A.T. Corbett: Skill Acquisition and the LISP Tutor, Cognitive Science, 1989, 13, 467-505
6. F.L. Bauer, G. Goos: Informatik, 1. Teil, Berlin: Springer, 1982 (third ed.)
7. J.S. Brown, K. van Lehn: Repair Theory: A Generative Theory of Bugs in Procedural Skills. Cognitive Science, 1980, 4, 379-426
8. M.T.H. Chi, M. Bassok, M.W. Lewis, P. Reimann, R. Glaser: Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems, Cognitive Science, 1989, 13, 145-182

9. E.M. Clarke, F.A. Emerson, A.P. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems, 1986, Vol. 8, No. 2, 244-263
10. G.W. Ernst, A. Newell: GPS: A Case Study in Generality and Problem Solving, New York: Academic Press, 1969
11. P.M. Gollwitzer: Action Phases and Mind-Sets, in: E.T. Higgins, R.M. Sorrentino (eds), Handbook of Motivation and Cognition, 1990, Vol.2, 53-92
12. H. Heckhausen: Motivation und Handeln , Heidelberg: Springer, 1989 (second ed.)
13. B. Josko: Verifying the Correctness of AADL Modules using Model Checking. In: de Bakker, de Roever, Rozenberg (eds): Proceedings REX-Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. Springer LNCS 430, 1990
14. J.E. Laird, P.S. Rosenbloom, A. Newell: Universal Subgoaling and Chunking. The Automatic Generation and Learning of Goal Hierarchies, Boston: Kluwer, 1986
15. J.E. Laird, P.S. Rosenbloom, A. Newell: SOAR: An Architecture for General Intelligence, Artificial Intelligence, 1987, 33, 1-64
16. C. Lewis: Composition of Productions, in D. Klahr, P. Langley, R. Neches (eds), Production System Models of Learning and Development. Cambridge: MIT Press, 1987, 329-358
17. C. Möbus: 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, IFIP TC3, North-Holland, 1991, 137-144
18. C. Möbus, K. Pitschke, O. Schröder: Ein wissensstandsbezogenes Hilfesystem für Petrinetzmodellierer, in: V. Claus, U. Lichtblau (eds): 2. Kolloquium der Arbeitsgruppe Informatiksysteme, Bericht AIS-3, Universität Oldenburg, 1991
19. C. Möbus, O. Schröder: Representing Semantic Knowledge with 2-dimensional Rules in the Domain of Functional Programming, in: P.Gorny, M. Tauber (eds), Visualization in Human-Computer Interaction, Springer, 1990 (LNCS 439), 47-81
20. C. Möbus, O. Schröder, H.-J. Thole: Runtime Modeling the Novice-Expert Shift in Programming Skills on a Rule-Schema-Case Continuum, in: J. Kay; A. Quilici (eds), Proc IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction, 1991, 137-143
21. C. Möbus, H.-J. Thole: 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), Proc. 3rd Int. Conf on Computer-Assisted Learning ICCAL 90, Heidelberg: Springer, 1990 (LNCS 438), 36-49
22. D.M. Neves, J.R. Anderson: Knowledge Compilation: Mechanisms for the Automatization of, Cognitive Skills, in J.R. Anderson (ed), Cognitive Skills and their Acquisition. Hillsdale, Erlbaum, 1981, 57-84
23. O. Schröder: 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), ICCAL 90, Heidelberg: Springer, 1990 (LNCS 438), 142-157
24. J.A. Self: Bypassing the Intractable Problem of Student Modelling, in C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, 107-123
25. D. Sleeman, J.S. Brown (eds), Intelligent Tutoring Systems, New York: Acad Press, 1982
26. K. van Lehn: Toward a Theory of Impasse-Driven Learning, in H. Mandl, A. Lesgold (eds), Learning Issues for Intelligent Tutoring Systems. New York: Springer, 1988, 19-41
27. K. van Lehn: Mind Bugs: The Origins of Procedural Misconceptions, MIT Press, 1990
28. K. van Lehn: Rule Acquisition Events in the Discovery of Problem-Solving Strategies, Cognitive Science, 1991, 15, 1-47
29. K. van Lehn:, R.M. Jones, M.T.H Chi: Modelling the Self-Explanation Effect with Cascade 3, Learning Research and Development Center, University of Pittsburgh, 1991
30. S.A. Vere: Relational Production Systems, Artificial Intelligence, 1977, 8, 47-68
31. E. Wenger: Artificial Intelligence and Tutoring Systems, Los Altos: Kaufman, 1987
32. R. Winkels, J. Breuker: Discourse Planning in Intelligent Help Systems, in: C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, 124-139
33. J.G. Wolff: Cognitive Development as Optimisation, in L. Bolc (ed), Computational Models of Learning. Berlin: Springer, 1987, 161-205