C. Möbus

Fünfter Workshop

Intelligente Tutorielle Systeme

der Fachgruppe "Intelligente Lernsysteme"
der Gesellschaft für Informatik e.V.

am 2. Oktober 1992
in Karlsruhe

Abstracts

Herausgegeben von:

Rul Gunzenhäuser
Universität Stuttgart
Institut für Informatik

Heinz Mandl
Universität München
Institut füt Empirische Pädagogik
und Pädagogische Psychologie

Vorwort

Die Entwicklung Intelligenter Tutorieller Systeme (ITS) wirft nach wie vor eine Reihe grundlegender Forschungsfragen auf, die eine interdisziplinäre Zusammenarbeit von Informatik, Künstlicher Intelligenz, Kognitionspsychologie, Linguistik und Fachdidaktik notwendig machen.

Die Informatik und die Künstliche Intelligenz beschäftigen sich dabei sowohl mit den methodischen Grundlagen für die Realisierung komplexer rechnerunterstützter Lernumgebungen, Benutzeroberflächen und Hilfesystemen als auch mit neuen Ansätzen der Wissensrepräsentation in Intelligenten Tutoriellen Systemen.

Die Kognitionspsychologie untersucht theoretische Grundlagen der Modellierung und Simulation menschlichen Problemlösens und des Wissenserwerbs. Die Linguistik beschäftigt sich insbesondere mit den theoretischen Grundlagen und den methodischen Realisierungen natürlichsprachlicher Komponenten.

Der vorliegende Band umfaßt die Abstracts der auf dem 5. Workshop der Fachgruppe am 2. Oktober 1992 im Rahmen der GI-Jahrestagung in Karlsruhe gehaltenen Referate. Erwünscht waren Beiträge zu folgenden Themen:

- Neuere Entwicklungen Intelligenter Tutorieller Systeme
- Benutzerschnittstellen und Hilfesysteme für ITS
- Expertensysteme als Lernsysteme
- Wissensrepräsentation und -modellierung für ITS
- Modelle und Anwendungen aus der Wissenspsychologie für ITS
- Linguistische, psychologische und didaktische Aspekte für ITS

Stuttgart / München: September 1992
Rul Gunzenhäuser / Heinz Mandl

Redaktion:
Dipl. Inform. Doris Nitsche-Ruhland
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20-22
7000 Stuttgart 80

Telefon: 0711/7816 404
Telefax: 0711/ 7801045

Inhalt

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, 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.

Introduction

It has been well recognized that in order to develop intelligent help systems which adapt to the learner's current knowledge state, a detailed psychological theoretical framework of knowledge representation and acquisition is necessary which incorporates problem solving and learning. It should be detailed enough to support specific design decisions and to allow specific predictions. At the same time it should be general enough to be applicable to different domains.

We are working 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 (Brown & van Lehn, 1980; Laird, Rosenbloom & Newell, 1986; 1987; van Lehn, 1988; 1990), success driven learning (Anderson, 1983; 1986; 1989; Lewis, 1987; Wolff, 1987), and problem solving phases or action phases (Gollwitzer, 1990; Heckhausen, 1989). 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 to problems. In order to make these design criteria as domain independent as possible, we apply them to two different domains: The ABSYNT problem solving monitor delivers help for learners constructing functional programs. The PETRI HELP system is designed to support Petri net planners. In this paper we will describe ISPDL Theory, the design principles following from it, and how they are or will be realized in ABSYNT and in PETRI HELP.

The ISPDL Theory: Problem Solving, Acquisition and Improvement of Knowledge

The ISPDL Theory (Möbus, Schröder & Thole, 1991) 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* (Gollwitzer, 1990). In the *deliberate* phase the problem solver considers several goals and finally chooses one. In the *plan* phase the problem solver develops a solution plan in order to obtain the goal. Subgoals are created and sequenced. The

solution plan might also be accomplished by analogical reasoning. Then the plan is *executed*. Finally the problem solver *evaluates* the obtained 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 (van Lehn, 1988; 1990; 1991). 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 (Brown & van Lehn, 1980; Ernst & Newell, 1969).

• The *success driven improvement of existing knowledge*. Successfully used knowledge is improved so it can be used more effectively. By rule composition (Anderson, 1986; Lewis, 1987; Neves & Anderson, 1981; Vere, 1977) the number of control decisions and subgoals to be set is reduced.

Principles for Help Design Based on ISPDL Theory

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

1. The help system should not interrupt the learner (see also Winkels & Breuker, 1990) but *offer* information, because according to the theory, information is only helpful at impasse time (van Lehn, 1988). So information is only to be supplied on request by the learner. This principle is somewhat opposed to the principle of immediate feedback (Anderson, 1987; Anderson, Conrad & Corbett, 1989). 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* during problem solving. Thus the system must offer support in each problem solving phase, because different impasses are possible at each point in problem solving.
3. Impasses may arise at *different levels*, so help information should be provided accordingly. It should support the problem solving phases of *planning, implementation*, and *evaluation*.
4. The learner should be enabled to *make use of her/his pre-knowledge* when asking for help as much as possible, so the information provided as help does not suggest different solution plans and thus causes secondary impasses. Rather, the help information should accept the learner´s solution plan and provide the learner with the requested information as precisely as possible.
5. 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 (Chi et al., 1989; van Lehn, Jones & Chi, 1991) of the information obtained, but also to non-understanding and to negative emotions. If the information presupposes 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 necessary which has 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.

6. The *processes of knowledge acquisition and modification, the application of weak heuristics and control processes* should also be modelled. Such a *process* model is an extension of the state model and may make use of additional data, like verbalizations. One of the functions of the process model is to support the development of the state model.

7. It is necessary that *the learner is free in the choice and sequencing of her/his interactions* with the system. This is necessary to construct the state model. It changes continuously by impasse driven knowledge acquisition and success driven knowledge improvement. The more restricted the range of the user's actions is, the less information can be obtained for infering cognitive states and processes.

These seven design principles and the ISPDL Theory were developed in the context of ABSYNT, a visual language and a problem solving monitor with a help system for functional programming. We started to apply them to the design of PETRI HELP as a second domain.

ABSYNT

ABSYNT ("Abstract Syntax Trees", Möbus, 1991; Möbus & Thole, 1989; 1990; Möbus & Schröder, 1990) was developed from ideas stated in a computer science textbook (Bauer & Goos, 1982). It consists of a functional, visual programming language (comparable to pure LISP without the data list structure) and a Problem Solving Monitor which is aimed at supporting programming novices with help and proposals (Sleeman & Hendley, 1982; Kearsley, 1988) while they are acquiring functional programming concepts up to recursion. ABSYNT was designed to encourage explorative but help-guided learning. The ABSYNT system consists of four main parts (Janke & Kohnert, 1989; Möbus & Thole, 1989; 1990; Möbus & Schröder, 1990):

• 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* which makes each computational step of the ABSYNT interpreter visible.

• A *diagnosis-, hypothesis- and help environment* where the learner may state the hypothesis that her/his solution proposal (or part of that proposal) to a given 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 *language level* by synthesizing complete solutions for the given programming tasks, starting from the student hypothesis. If the hypothesis is embeddable in a complete solution, the learner may ask for completion proposals. Figure 1 shows a wrong solution proposal to the "even" task (upper window), a hypothesis (bold parts of upper window), feedback that this hypothesis is embeddable (tiny parts of lower window) within a correct solution, and a completion proposal generated by the system (bold parts of lower window).

A set of diagnostic rules defining a goals-means-relation (an AND-OR-graph with parametrized nodes, Möbus & Thole, 1990) analyzes and synthesizes solution proposals. The rules recognize incomplete proposals and generate complete solutions to the programming tasks.

We are also working on program construction and help generation on the *planning level*. The learner will be able to construct goal trees using goal nodes which are derived from the rules of the goals-means-relation. Each goal node will be equipped with a predicative description. Our intention is that the learner is able to test hypotheses and receive error and completion feedback on this goal level in a similar way as on the language level, as described above.

• A *learner model* ("state model", Möbus, Schröder & Thole, 1991) which controls knowledge diagnosis and help generation. The learner model represents the actual hypothetical state of domain knowledge of the learner. It is continuously updated based on the learner´s programming actions and the times between different programming steps.

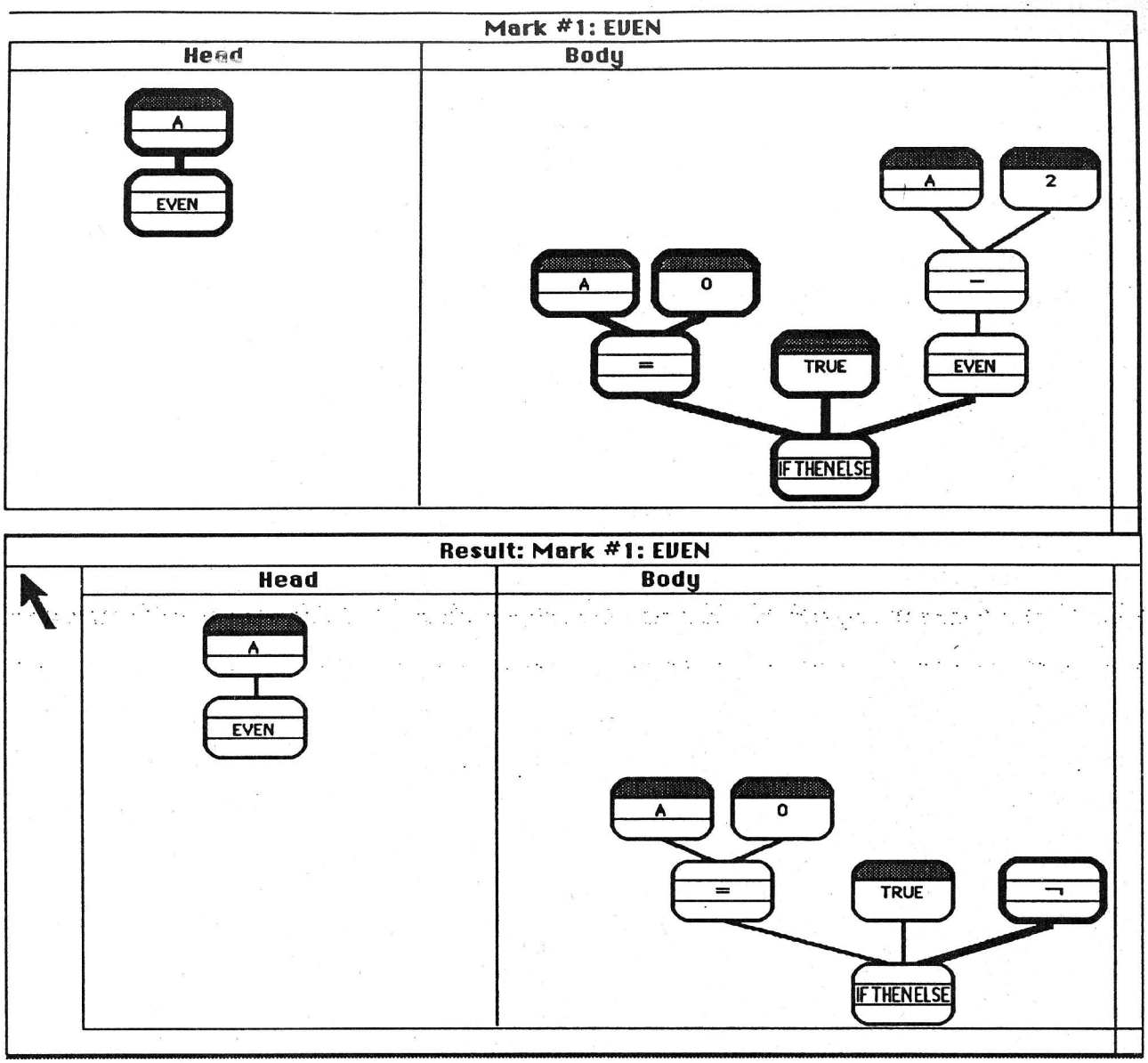

Figure 1: A hypothesis and a completion proposal in the ABSYNT environment

How are the seven 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 allows *detailed support* at impasse time by enabling the learner to test hypotheses and obtain completion proposals.

• Concerning design principle 3: The system allows resp. will allow program construction at the *planning level* (working with goal nodes) and at the *language level* (working with operator nodes, parameters, and constants). Additionally, the system provides help at the *language level* (hypothesis testing, obtaining completion proposals) and will provide similar information at the *planning level.* The hypothesis testing approach is a powerful means for *evaluating* one's solution proposal. Thus the three problem solving phases of planning, executing, and evaluating are systematically supported in ABSYNT.

• Concerning design principle 4: 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 5: The help information is *tailored to the actual knowledge state* of the learner, because help generation is based on the state model which continuously changes dependent on the learner´s programming actions and latency times. In accordance with ISPDL Theory, the state model contains knowledge *acquired* by heuristics (which are not themselves part of the state model) and knowledge *optimized* by composition.
• Concerning design principle 6: There is also a *process model* containig hypothetical processes of knowledge acquisition and modification, impasses, and subsequent problem solving heuristics.
• Concerning design principle 7: Positioning, naming, moving, deleting, connecting and unconnecting nodes are *distinct and freely arrangeable actions* in the visual ABSYNT editor.

PETRI-HELP

In the PETRI-HELP-project, a system is developed for supporting problem solvers in the domain of modelling with condition-event Petri nets. Like in ABSYNT, the system is intended to provide user-centered help sensitive to the actual knowledge state of the user. Although the goal of this project is related to ABSYNT´s, there are differences 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.* In ABSYNT, the programming tasks known by our system are easy to express as tasks for the learner. But for Petri nets the task specification is more complicated. Thus a more precise task specification is necessary, but in the domain of Petri nets it is not usual to present a formal specification of the problem to be solved. An exception is e.g. Josko (1990) who used temporal-logic formulae to verify the correctness of Petri nets which are used to specify the semantics of AADL-programs. We developed task descriptions which are sets of temporal-logic formulae. The advantage of temporal logic task specifications is that they enable the verification of learners´ Petri net proposals ba model checking. We developed temporal logic specifications for 15 tasks partially ordered according to five learning objectives: Petri nets for implications with a) atomic formulae, b) conjunctions, c) same formulae, d) disjunctions in premise and conclusion, and e) with additional constraints. Figure 2 shows the 15 task names partially ordered by the learning objectives. Figure 3 shows the temporal logic specification and a solution proposal for one task, "restaurant".
In empirical studies, our subjects did not have serious difficulties with the temporal logic task descriptions.
• *Analyzing the learners´ solution proposals.* As indicated, we developed a simple model-checker for the diagnosis of 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 (sub-)net. The model checker may be used on every user-created (sub-) net. For this (sub-) net the case graph is computed. The alternate paths in this graph describe the different orders of firing transitions in the net. Figure 4 shows a part of the case graph for the net in Figure 3. The names of the places in the net must correspond to the atoms in the temporal-logic formulae. The user has to specify an initial

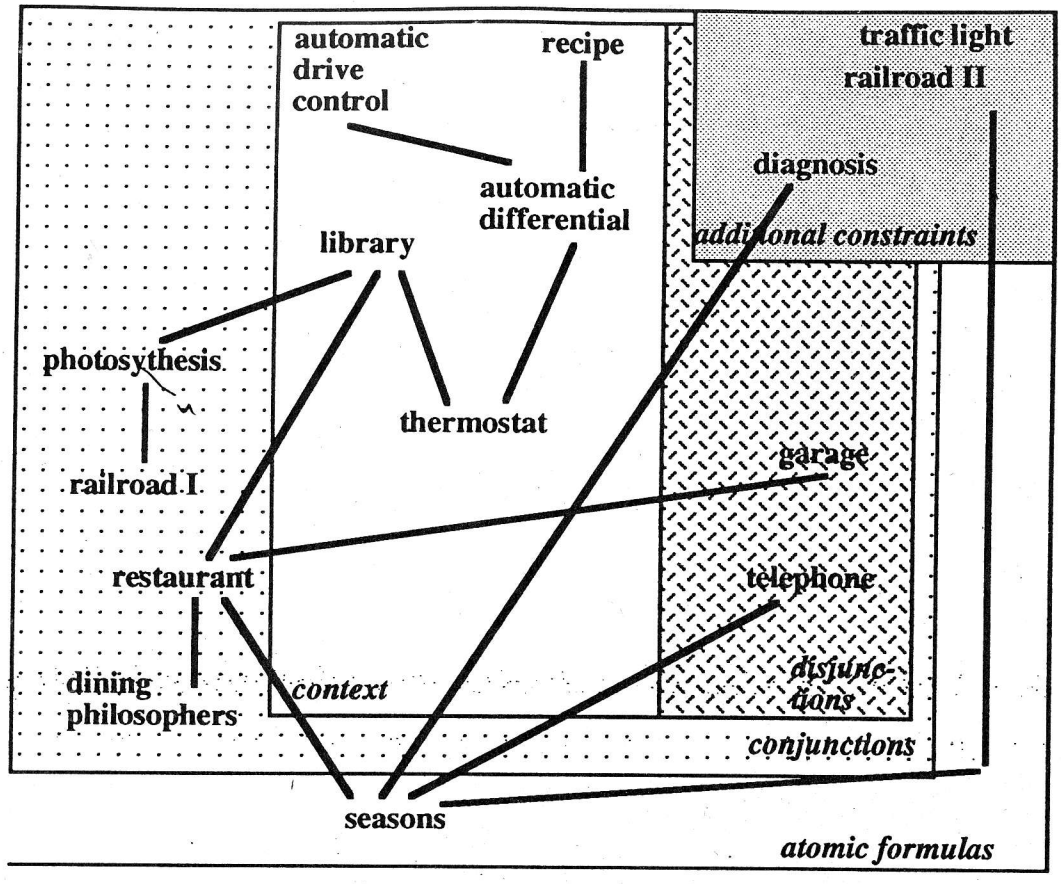

Figure 2: Tasks partially ordered by learning objectives

Ws : Waiter is sleeping
Wro : Waiter is ready to accept an order
Wrs : Waiter is ready to serve
K : Kitchen got the order
P : Preparation
R : Meal is ready

$\Box\ (\neg(Ws \wedge Wro))$
$\Box\ (\neg(Ws \wedge Wrs))$
$\Box\ (\neg(Wro \wedge Wrs))$
$\Box\ (Ws \vee Wro \vee Wrs)$
$\Box\ (Wro \rightarrow \Diamond\ (Ws \wedge K))$
$\Box\ (K \rightarrow \Diamond\ P)$
$\Box\ (P \rightarrow \Diamond\ R)$
$\Box\ (R \wedge Ws \rightarrow \Diamond\ Wrs)$
$\Box\ (R \wedge Wro \rightarrow \Diamond\ Wrs)$
$\Box\ (Wrs \rightarrow \Diamond\ Ws)$
$\Box\ (Ws \rightarrow \Diamond\ Wro)$

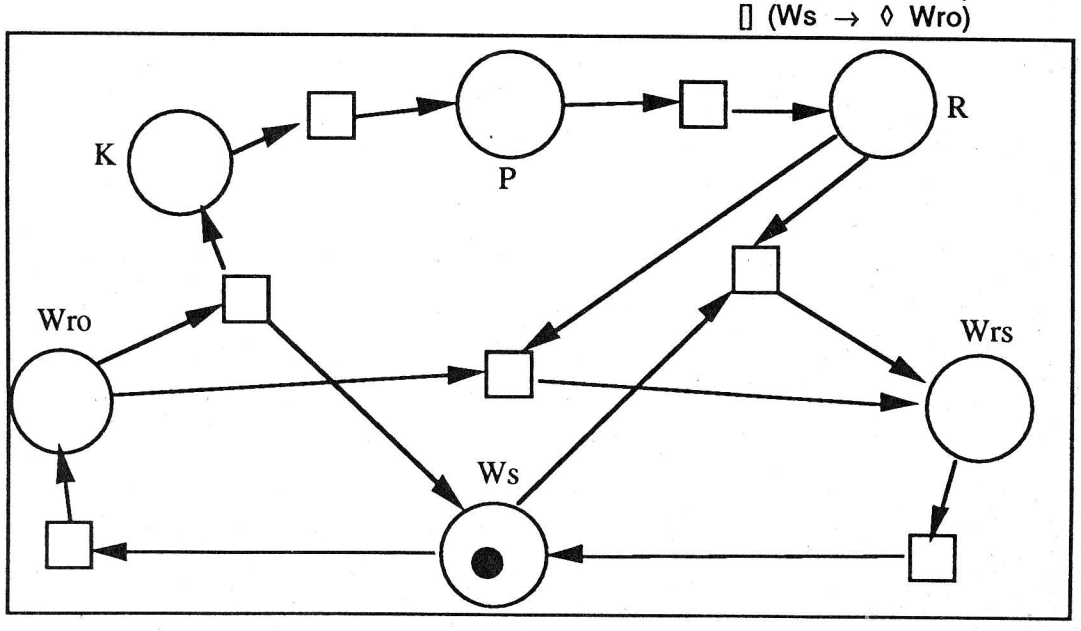

Figure 3: Specification and solution to the task "restaurant"

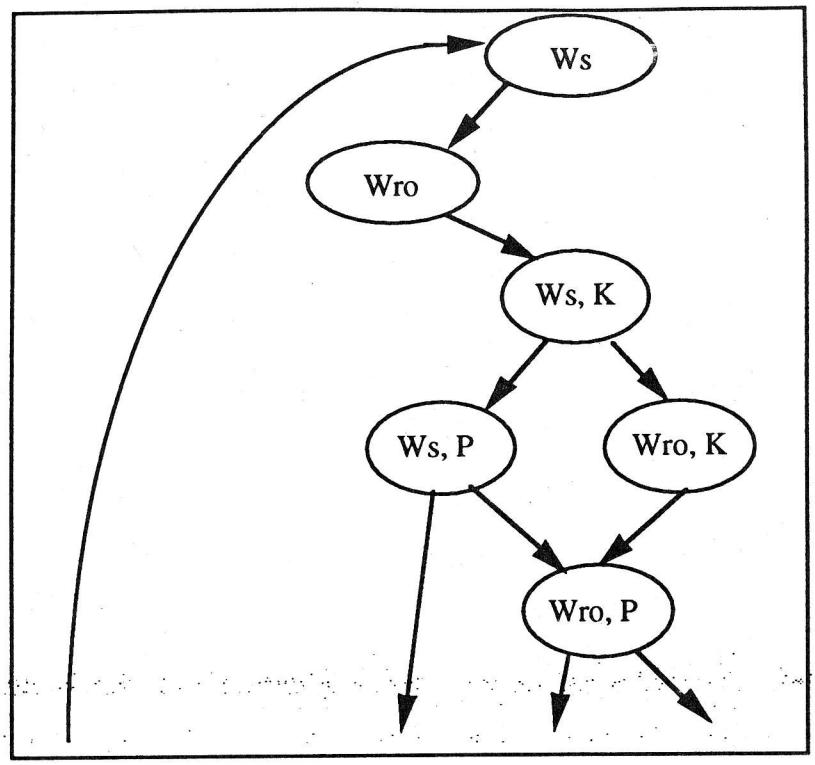

Figure 4: A part of the case graph for the solution of Figure 3

situation in the net, i.e., an initial set of tokens spread over the places. This situation corresponds to the initial state of the case graph, where the interpretation of the logical formulae starts.
If the formula contains no temporal-logic operator (O = nexttime, ◊ = eventually, [] = always), it is a propositional-logic formula and will be evaluated inside the current node of the case-graph:
A conjunction of atoms is true iff every atom is represented in this state of the case-graph (these places contain tokens). A disjunction is true iff the corresponding place of at least one of the atoms contains a token. Implications A -> B are decomposed into not A or B.
If the formula has the shape O F (where O means nexttime, F is a formula), the F must hold in every immediate successor of the current state in the case-graph.
◊ F is true (eventually F) iff in every path leaving the current node F will be true at least in one node.
[] F (always F) holds iff F holds in every state on every path leaving the current state.
The formulae are interpreted recursively on the case graph.

• *Design rules, help information.* Based on the model checker and on empirical studies with Petri net modellers, we developed three kinds of design rules which may be used to support learners. These design rules may be created automatically by the system in response to the learners´s actions:

- Design rules proposing *augmentations to the existing net* such that a superset of the currently fulfilled formulae is fulfilled.
- Design rules proposing a *whole net* fulfilling a superset of the currently fulfilled formulae.
- *Building blocks* which relate formulae to net parts. They have been identified in empirical studies with 14 subjects working in single-subject sessions. The subjects were asked to relate sets of formulae to parts of their solution proposal while constructing Petri nets for our tasks.

Design rules may help the learner in two ways:

- If the learner chooses a design rule then her solution proposal will be completed by the system according the the rule. (In case of Building blocks, the number of formulae fulfilled may be reduced because of the nonmonotonic property of Petri nets with respect to the formulae. Thus, additional model checking is necessary.)
- The learner may let the system check the solution proposal (or part of it) by stating hypotheses. The system then may list the formulae currently satisfied and deliver completion proposals.

This outline of the to-be-developed system will be augmented in three ways:

- A *learner model* representing the actual knowledge state of the learner will control which design rules and completion proposals will actually be offered to the learner.
- A fourth kind of design rules is under investigation which will also incorparate a "goal level". The learner will be enabled to *plan* aspects such as "parallelism", "mutual exclusion" etc.
- The design rules will not be built into the system but learned, based on users´ actions.

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

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

• Concerning design principles 2 and 3: *Detailed feedback* concerning *planning*, *implementation*, and *evaluation* of net proposals will be provided by the model checking of the specification formulae after every construction step of the learner. Again, different levels of help will be possible:

- The set of formulae that hold in the current subnet (by making use of the model checker)
- Design rules suggesting augmentations or revisions to the current subnet
- Building blocks suggesting augmentations (to be verified by model checking)
- Planning rules suggesting intermediate concepts.

• Concerning design principle 4: The selection of design rules, hypothesis testing, and receiving proposals of possible completions are to be under *control of the learner.*

• Concerning design principle 5: The design rules representing the domain-knowledge are to be *learned* during every user´s problem-solving. Thus a *state model* will be constructed.

• Concerning design principle 6: The state model will be extended by a *process model.* It contains information about the evolution of knowledge structures as stated by the state model.

• Concerning design principle 7: There is free choice of net editing actions by the user. The learner may let the system create places, arcs, and transitions by choosing among design rules and building blocks. Alternatively, she may prefer to perform these primitive editing actions by herself.

Conclusions

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

ISPDL Design Principles	ABSYNT	PETRI HELP
1. "Do not interrupt the learner - offer help"	Learner can always get help	Learner can always get help
2. "Provide detailed information the whole time on demand"	Hypothesis feedback and completion	Hypothesis feedback and completion Model checking Design rules, building blocks
3. "Provide information for planning, implementation, and evaluation"	Editing, hypothesis testing, and completion proposals on the goal level and implementation level. Hypotheses as means of evaluation	Editing with design rules, building blocks, and planning rules. Hypotheses and model checking as means of evaluation
4. "Let the learner use her/ his pre-knowledge"	Stating hypotheses	Stating hypotheses Selecting design rules
5. "Tailor information to the knowledge state of the learner"	State model controlling completions to hypotheses	State model controlling completions (constructed by learning rules) and the actual set of design rules offerend to the learner
6. "Back the state model by a process model"	Process model: Impasses, heuristics, knowledge acquisition processes	Process model: Impasses, heuristics, knowledge acquisition processes
7. "Provide freedom in the learner´s actions"	Free arrangement of positioning, naming, moving, deleting, connecting, unconnecting nodes	Free arrangement of positioning, naming, moving, deleting, connecting, unconnecting places and transitions

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., Production Systems, Learning, and Tutoring, in Klahr, D.; Langley, P.; Neches, R. (eds): Production, System Models of Learning and Development. Cambridge: MIT Press, 1987, 437-458

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

Bauer, F.L., Goos, G., Informatik - eine einführende Übersicht, 1. Teil, Berlin: Springer, 1982 (3. Aufl.)

Brown, J.S., van Lehn, K., Repair Theory: A Generative Theory of Bugs in Procedural Skills. Cognitive Science, 4, 1980, 379-426

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R., Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems, Cognitive Science, 1989, 13, 145-182

Ernst, G.W., Newell, A., GPS: A Case Study in Generality and Problem Solving, New York: Academic Press, 1969

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

Heckhausen, H., Motivation und Handeln, Heidelberg: Springer, 1989 (2. Aufl.)

Janke, G., Kohnert, K., Interface Design of a Visual Programming Language: Evaluating Runnable Specifications. 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, 567 - 581

Josko, B., 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, 1990Kearsley, G., Online Help Systems: Design and Implementation, Norwood, N.J., 1988

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

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., 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), 1991, 137-144

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, S. 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, 1991, 137-143

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

Self, J.A., Bypassing the Intractable Problem of Student Modelling, in C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, 107-123

Sleeman, D., Brown, J.S. (eds), Intelligent Tutoring Systems, New York: Academic Press, 1982

Sleeman , D., Hendley, R.J., ACE: A System which Analyzes Complex Explanations, in Sleeman, D., Brown, J.S. (eds), Intelligent Tutoring Systems, New York: Academic Press, 1982, 99-118

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

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

van Lehn, K., Jones, R.M., Chi, M.T.H., Modelling the Self-Explanation Effect with Cascade 3, Learning Research and Development Center, University of Pittsburgh, 1991

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

Wenger, E., Artificial Intelligence and Tutoring Systems, Los Altos: Morgan Kaufman, 1987

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