Chapter 3

PETRI-HELP - Intelligent Support for Modelling Time-Discrete Distributed Systems with Petri Nets

Authors: C. Möbus, K. Pitschke, O. Schröder, J. Folckers, H. Göhler

3.1 Abstract

An Intelligent Problem Solving Environment (IPSE) is described that supports novices in learning to construct models of time-discrete distributed systems with Condition-Event Petri nets. The system, PETRI-HELP, is based on a theoretical framework recommending that a help system should *offer* help, let the learner use *pre-knowledge*, and support different problem solving *phases*. In PETRI-HELP the learner creates Petri nets for given tasks, tests hypotheses about the solutions (or fragments of them), and receives feedback, completions, and correction proposals.

In PETRI-HELP, tasks are stated to the learner as sets of temporal logic formulas. On request, the system analyzes the Petri net fragment created by the learner and informs the learner about the actual (sub-) set of task formulas fulfilled by the current solution proposal. In addition, the system represents rules that reflect learners' successful steps towards task solutions. These rules are used for offering completions and correction proposals. So the system constantly improves itself by learning from its users.

According to our theoretical framework of help system design as well as empirical work done with PETRI-HELP, the system should also support *planning*, i.e., creating an abstract solution idea and postponing implementation decisions, and the *construction* of

tasks, not only solutions for tasks. Some additional work will be described that is aimed at extending PETRI-HELP into these two directions. PETRI-HELP led to the development of another system: an IPSE for the design of pneumatic circuits. This system will be briefly described in the closing section.

3.2 Introduction

Intelligent help systems and knowledge communication systems are expected to supply the user with information that is sensitive to the actual problem solving situation and to the actual knowledge and intentions of the user. Developing this kind of systems requires a variety of design decisions, like when to supply remedial information, what to supply (what determines "good" help?), and how to present it. The *acceptance* of knowledge communication systems by users critically depends on satisfactory solutions to these problems.

In order to support design decisions for the development of an intelligent help system, a theoretical framework of problem solving and learning is needed. Without such a framework, design decisions will be largely unmotivated and ad hoc, resulting in a system which tends to be inconsistent, difficult to work with, and not accepted by its users. Our ISP-DL Theory (impasse – success – problem solving – driven learning theory) is such a theoretical framework. Figures 3.1 and 3.2 give a brief overview of it. The ISP-DL Theory attempts to integrate impasse-driven learning [22, 23, 28, 38, 42, 43], success-driven learning (e.g., [2, 3, 4, 46, 47]), and phases of problem solving [14, 17]. According to the ISP-DL Theory [26, 27, 24], if the problem solver is faced with a set of possible goals, he or she will construct a solution for a goal, and the knowledge used in this problem solving process will be deductively optimized (success-driven learning) so it will be used more efficiently in future (Figure 3.1).

The problem solving process consists of four phases (Figure 3.2): The problem solver (PS) *deliberates* with the result of choosing or creating a goal to pursue. Then a *plan* to reach the goal is synthesized or transferred from an earlier problem by analoguous reasoning. Next, the plan is *executed*, and the obtained result is *evaluated*. *Impasses* might result at several points in this process: The PS might not be able to choose or to create a goal, or the plan cannot be created, or execution may not be possible, or the obtained result may not be satisfying. The PS reacts to an impasse by entering a new problem solving process, using weak *heuristics* like looking for help, asking, and cheating. As a result, the PS may acquire new knowledge inductively (impasse-driven learning) so he may overcome the impasse and continue with the original problem solving process. But alternatively, the information obtained may not be helpful but confusing, so the learner might encounter a secondary impasse [5].

The ISP-DL Theory leads to several design principles for a knowledge comm 57 unication system (Figure 3.3):

- According to the 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 system should not interrupt the learner but offer help only on request.

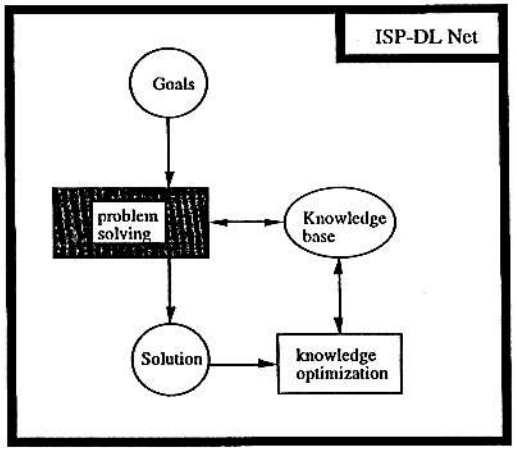

Figure 3.1: ISP-DL Theory: Problem solving and deductive knowledge optimization

- The learner should be prevented from trapping into secondary impasses that may lead away from the original problem solving. So *pre-knowledge* should be usable at impasses as much as possible. Help should be adapted to the knowledge state of the learner, i.e., help should be *user-oriented*. This requires either a learner model or the possibility to test hypotheses about the solution created (see below).
- According to the ISP-DL Theory, help should be provided at different phases of problem solving because impasses may arise in all phases. So a help system should support *deliberating, planning, executing*, and *evaluating* solution proposals. Help should be problem phase oriented.

These three requirements are well met by letting the learner *test hypotheses* about her or his solutions or solution fragments, and get help and proposals from the system [24]. This leaves the activity on the learner's side, the learner is not disturbed by unwanted system comments, but information is *offered* to the learner and may be obtained on request.

Secondly, hypotheses testing means that the system takes account of the user's ideas and intentions, that is, of his pre-knowledge, without leading him away onto different problem solving paths. Thirdly, the hypotheses testing approach takes account of different problem solving phases (Figure 3.2, this will be discussed below). We call systems designed according to these criteria Intelligent Problem Solving Environments (IPSEs, [24]).

PETRI-HELP [25, 30, 39] is an IPSE designed to support novices modelling with Condition-Event Petri nets. According to the ISP-DL Theory, modelling with Petri nets is a problem solving activity consisting of the following sub-activities[1]

[1]Each of these phases can again be considered as consisting of the four subphases. For example, the task of developing a specification of a system may consist of the subphases "*deliberating*" (deciding what to specify), "planning" steps to perform in order to create a specification, "executing" this plan, and "evaluating" the result (for example, finding out whether the specification contains contradictions). So the concept of different phases of problem solving activities has a recursive structure.

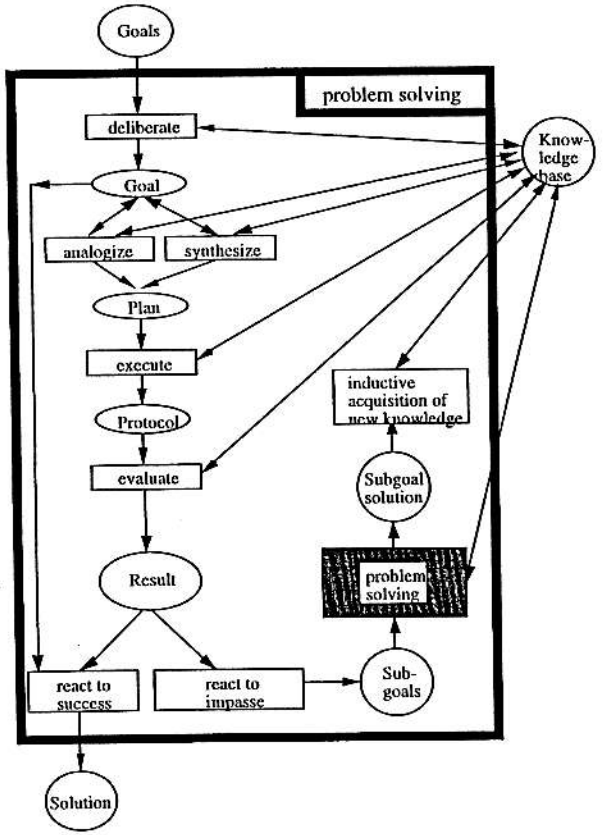

Figure 3.2: ISP-DL Theory: Problem solving phases and inductive knowledge acquisition

- to develop specifications of systems or processes to be modelled ("*deliberating*")
- to plan a Petri net solution for a given specification ("*planning*")
- to actually construct a Petri net ("*executing*")
- to evaluate the resulting net, for example whether it meets the specification ("*evaluating*").

So the skill of modelling with Petri nets may be decomposed into four subskills. The first subskill does not directly refer to the construction of Petri nets but to the development of specifications for them. The other three subskills refer directly to Petri net construction.

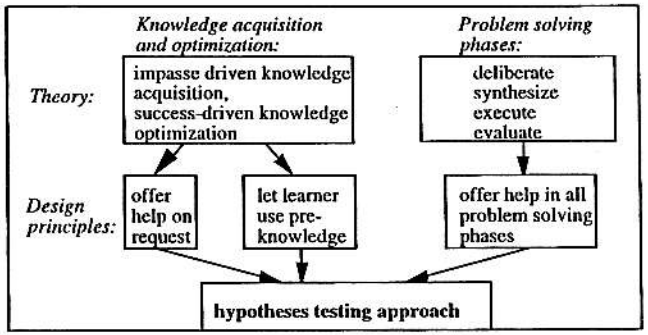

Figure 3.3: Design principles following from the ISP-DL Theory for intelligent knowledge communication systems

In PETRI-HELP, the learner can create Condition Event Petri nets for given tasks, test hypotheses, and ask for help and proposals. In the next section we will describe our preliminary empirical studies which we did in order to investigate the design principles for the development of PETRI-HELP. Then we will describe the main components of PETRI-HELP. After that some empirical work with PETRI-HELP is presented. The results indicate how PETRI-HELP should be extended. The fifth section describes work aimed at these extensions: The incorporation of a user model, explanations, the systematic construction of net solutions, and support of specification development (the first of the four subskills mentioned above). Finally, the cooperations of the PETRI-HELP project and its extension leading to a new system in the domain of pneumatic circuits will be described.

3.3 Empirical Work Supporting the Development of PETRI-HELP

During the initial phase of the development of PETRI-HELP, several questions were raised which had to be solved empirically. Our preliminary empirical investigations were aimed at the following questions:

- What difficulties do especially novices have while constructing Petri nets? Where do they need assistance?
- In order for a system to analyze learners' Petri net solution proposals, a task description or specification is necessary. Temporal logic [21] is a convenient means for specifying time-discrete systems. Furthermore, specifying modelling tasks as sets of temporal logic formulas allows to analyze Petri net solution proposals by model checking (see below in more detail). But before pursuing this approach, we had to investigate empirically whether it is feasible for novices at all to work with temporal logic task specifications, and whether users accept temporal logic task specifications or reject them as incomprehensible, for example.

- What kinds of strategies do subjects pursue in constructing Petri nets?

In these studies we made use of the Petri net editor developed by the MOBY Project Group [12]. Fourteen subjects worked in single-subject sessions. Each temporal logic task formula was written on a card. Figure 3.4 gives an example.

The subjects received a short informal description of the temporal logic operators. The subjects were asked to put each card at a certain place as soon as they considered the formula as being fulfilled by the current state of their Petri net solution. Some of the subjects' interactions with the system were videotaped, so it was possible to find out the subjects' associations of formulas to Petri net fragments.

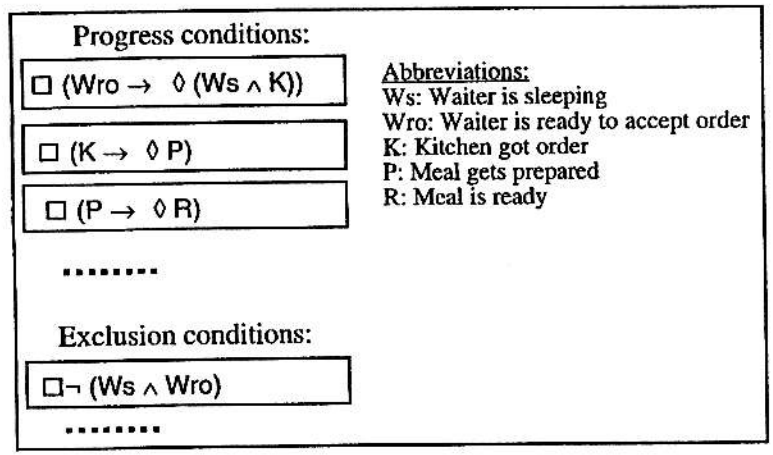

Figure 3.4: Cards with temporal logic formulas as part of the material used in our preliminary empirical studies

As stated earlier, viewed from our ISP-DL Theory the construction of a Petri net to a given task can be described as consisting of synthesizing a plan, executing it, and evaluating the result. Analyses of the video protocols showed that the strategy pursued by novices can be summarized by the following steps:

- Choose a formula
- Synthesize a Petri net fragment that might fulfill the formula
- Execute this plan
- Evaluate the result by net simulation
- If necessary: Change the net
- Repeat the process with another formula

The following kinds of impasses were observed most frequently:

- In the *synthesize* phase, subjects encountered the impasse that the realization of a formula seemed to invalidate another formula already realized earlier, so they had to reconsider the earlier formula(s). Another common impasse was that they did not know how to proceed, especially in the initial phase of problem solving. For example, they did not know how to express a disjunction in a Petri net, and some of them asked for hints in these situations.
- In the *evaluate* phase, subjects tended to lose track of the simulation so they were very uncertain about whether to consider a formula fulfilled or not. A related impasse was that subjects felt uncertain in deciding when the net was complete, i.e., the task finished.

None of our subjects had serious problems with reading the formulas, so we did not consider it necessary to develop additional material for explaining the task specifications.

The novices did not seem to follow a special strategy in selecting and realizing the task formulas. But with more expertise, the subject's net construction processes tended to become more systematical. When becoming "intermediates", the subjects started to follow the design strategy: "First implement the temporal logic implications. Then check the exclusion conditions." (i.e., the co-occurrence of two states is excluded, like $\neg$ (Ws $\wedge$ Wro), see Figure 3.4). Within this design strategy, we were able to identify several *design heuristics* that the subjects seemed to follow. Each design heuristic links a progress condition to a Petri net fragment. Figure 3.5 shows the nine most frequent design heuristics identified. (For example, the design heuristic "merging" says that a progress condition with the pattern "$\Box(x_1 \wedge \cdots \wedge x_m \rightarrow \Diamond y)$" can be implemented in PETRI-HELP as a set of places labeled $x_1, \cdots, x_m$ leading to a transition leading to a state y.) The design heuristics fully explained 40 of 54 solutions created by our subjects.

Another observation was that with the use of design heuristics, subjects tended to encounter less impasses in the evaluation phase. They considered their solution as complete as soon as a net fragment had been constructed for each progress condition. Consequently, they tended to make less use of the net simulation. But the design heuristics worked only as long as after applying them the exclusion conditions did not require the consideration of any additional constraints. One of the tasks requiring special attention to the evaluation conditions was "traffic lights": In addition to applying design heuristics, the subjects had to make sure that one traffic light is always red. With the task "traffic lights", the subjects had serious problems, and based on the formulas there was no way to assist them. They did a lot of trial and error in this situation, making much use of the net simulation facility.

One of our subjects was an expert: a computer scientist who had much experience with Petri nets. Our main observations were:

- The expert also followed the design strategy: "First implement the temporal logic implications. Then check the exclusion conditions."

- As to be expected from ISP-DL Theory, the expert aggregated certain steps. For example, novices and intermediates checked the exclusion conditions one by one, but the expert took all cards containing exclusion conditions at once and checked them "in one glance".
- The expert did not work according to the design heuristics (Figure 3.5) as clearly as the intermediates. For example, he tried to keep the nets as simple as possible. In some cases he skipped progress conditions containing the "$\Diamond$" operator, because he considered these formulas to be fulfilled as a by-product of the realization of other formulas.

So the main results of these preliminary studies with respect to the design of PETRI-HELP were:

- In some situations, novices have problems in expressing logical operators, especially disjunction, as Petri net fragments. So there should be information available about how to do this, i.e., hints about the next few editing steps should help to overcome this impasse.
- Novices have problems in deciding when task formulas are fulfilled, and when the task is finished. So the subjects should be provided with information concerning the evaluation of solution proposals, like information about fulfilled and unfulfilled formulas.

These empirical results are in accordance with our design principles for PETRI-HELP: The first problem novices have could be addressed by providing completion or correction proposals. This takes account of the learner's solution proposal as much as possible, so the learner has to make only minimal changes to his proposal. The second problem could be addressed by testing hypotheses about fulfilled and unfulfilled formulas. In this way, the learner may make use of her / his pre-knowledge because the *learner* specifies what formulas he or she considers fulfilled by the proposal.

3.4 The Main Components of PETRI-HELP

PETRI-HELP consists of the following components:

- A sequence of *modelling tasks*. Each task requires the creation of a Petri net model for a time-discrete distributed system. Each task is specified as a set of temporal logic formulas so Petri net solution proposals can be analyzed by model checking.
- A *net editor* for constructing and simulating Petri nets.
- A *hypotheses test environment* where the PS may state hypotheses about the task formulas fulfilled by the current state of the solution. The system gives feedback about the hypotheses.

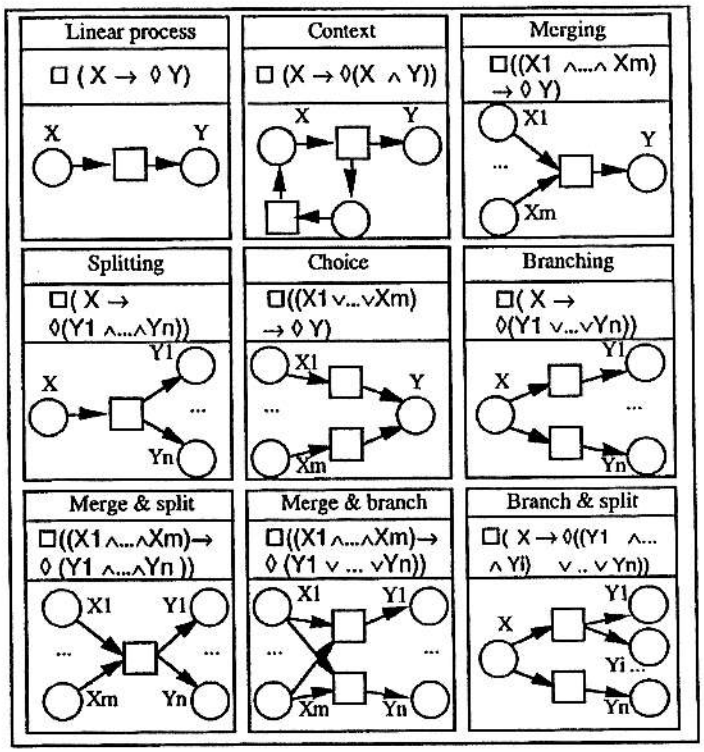

Figure 3.5: Empirically observed design heuristics

- A *completion and correction component* which on the PS's request delivers proposals how to complete or to correct the current state of a Petri net. This component is based on rules learned by the system from the solution proposals of other learners.

We will describe each component in turn.

3.4.1 Modelling Tasks

In order for a system to give help and support to a PS, it must be able to analyze the PS's solution to a given task. So a task specification is necessary. Specifications for Petri net modelling tasks are not very common, even in textbooks (e.g., [35, 36]) Petri nets tend to be presented as solutions to tasks described informally. Exceptions are for example Josko [19] where Petri nets are used to specify the semantics of computer architecture descriptions, and Olderog [29], where Petri nets define the semantics of process terms derived from trace specifications.

PETRI-HELP contains a sequence of ten modelling tasks. (The list of tasks could be extended easily.) The learner may create Petri net solutions to these given tasks. Each task is specified as a set of temporal logic formulas [21]. As mentioned before, this allows to verify Petri net solution proposals by model checking [8, 10, 19]. This is done by interpreting the temporal logic formulas on the case graph of the PS's Petri net proposal [25].

Figure 3.6 (left side) shows the temporal logic specification to the modelling task "Restaurant" [20]. Initially, the waiter is sleeping (starting condition: Ws). □, ◇, ○ are the temporal logic operators. Informally, □ means "always" ("it is always true that ..."), ◇ means "eventually" ("now or at some point in future it will be true that ..."), and ○ means "nexttime ("at the next point in time it will be true that ..."). So for example "$\Box(Ws \rightarrow \Diamond Wro)$" means informally: "It is always true that if the waiter sleeps then he will eventually be ready to accept an order." The window on the right of Figure 3.6 contains the descriptions of the abbreviations used in the formulas.

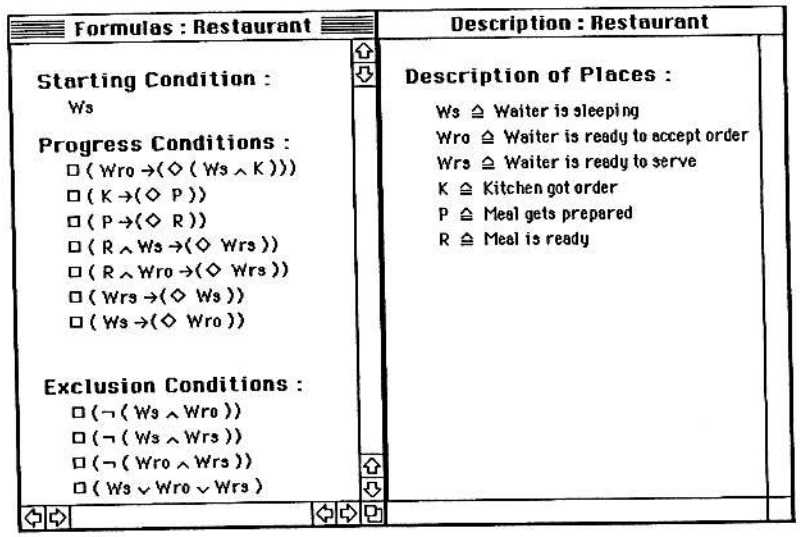

Figure 3.6: Temporal-logic specification of the modelling task "Restaurant" of the task sequence of PETRI-HELP

The tasks in PETRI-HELP are partially ordered according to five modelling goals. This partial order reflects our preliminary view about psychological task complexity. Increasing task complexity should also lead to an increase in task difficulty. Although we did not pursue this empirical hypothesis within the project, it could be investigated with a test theoretic approach (i.e. Rasch model as used by [40], with a cognitive complexity analysis (i.e., [33, 34]), or with a set theoretic approach (i.e., [11]).

The five modellings goals are:

a) "Atomic formulas": to create nets for implications with an atomic formula in premise and conclusio. Example: $\Box(Ws \rightarrow \Diamond Wro)$

b) "Conjunctions": to create nets for implications with conjunctions in premise and conclusio. Examples: $\Box((R \wedge Ws) \rightarrow \Diamond Wrs)$, $\Box((a \wedge b) \rightarrow \bigcirc (c \wedge d \wedge e))$

c) "Context": to create nets for implications with the same atomic formulas in premise and conclusio. Example: $\Box((a \wedge b) \rightarrow \Diamond(a \wedge c))$

d) "Disjunctions": to create nets for implications containing disjunctions in premise and / or conclusio as well. Example: $\Box((a \vee b) \rightarrow \bigcirc (c \wedge e \vee d \vee a \wedge c))$

e) "Additional constraints": to create nets for sets of formulas requiring special attention to the exclusion conditions (example: two traffic lights. In addition to the succession of colors for each traffic light, there is the constraint that one traffic light must always be red.)

The partial order of modelling goals incorporated in the sequence of PETRI-HELP modelling tasks is shown in Figure 3.7. As can be seen, there are tasks for modelling natural systems (four seasons, photosynthesis), technical systems (railroad, thermostat, automatic lock differential, traffic lights), social systems (restaurant, library), and systems containing technical as well as social aspects (telephone, garage). It would be no difficulty to extend the task set.

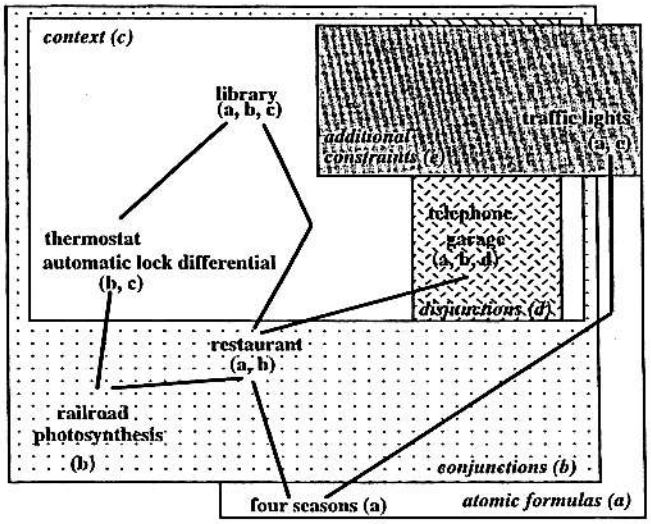

Figure 3.7: Partial order of the PETRI-HELP modelling tasks

3.4.2 Net Editor

Figure 3.8 depicts a snapshot of the PETRI-HELP net editor: a solution proposal of the "Restaurant" task. In a Condition-Event Petri net, circles represent places (conditions, states), and rectangles represent transitions (events). The condition represented by a place is true if the place contains a token. On the left of Figure 3.8 the tools for editing are shown: deleting, moving, naming, creating places, transitions, and arcs, and setting tokens. In addition, there is a tool for simulation.

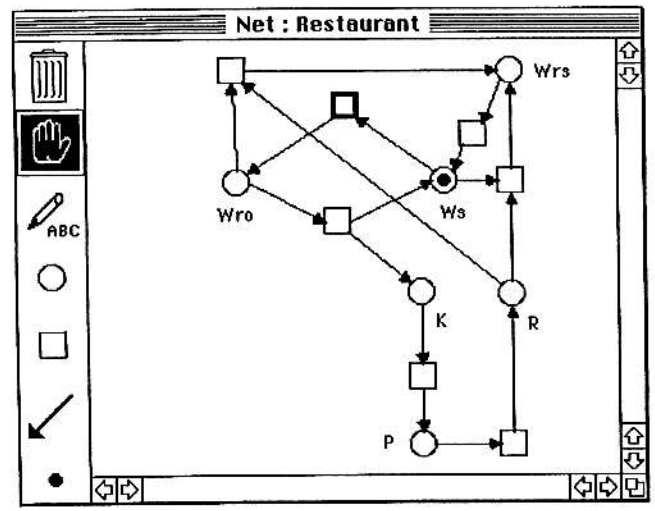

Figure 3.8: The PETRI-HELP net editor

3.4.3 Hypotheses Test Environment

When the PS is constructing a Petri net to a given task, he may state hypotheses about which subset of the formulas is fulfilled by the actual state of the solution. This is done by selecting the respective task formulas (Figure 3.9).

The system then analyzes the task formulas by model checking. As the result, it returns the formulas fulfilled and not fulfilled by the current state of the solution (Figure 3.10), i.e., it partitions the selected formulas into the largest subset of formulas the Petri net solution proposal is a model of, and the set of remaining formulas.

Our temporal logic allows for branching time and makes use of step semantics (cf. [10]). Its semantics is defined in Table 1. The temporal-logic formulas are interpreted on the case graph of the Petri net solution proposal. A part of the case graph of the net of Figure 3.8 is shown in Figure 3.11. Furthermore, since PETRI-HELP is restricted to Condition-Event Petri nets, the models are always finite and usually cyclic.

3.4.4 Completion and Correction Component

Based on the model checking approach, PETRI-HELP can inform the PS about what parts of the task specification are fulfilled by the current state of the solution. The model checking component does not tell the PS how to continue with the proposal or how to correct it, if the PS is caught in an impasse and does not know how to proceed. Therefore in PETRI-HELP there is also a completion / correction component.

When a PS is creating a correct Petri net to a given task, for example, by applying design heuristics like those in Figure 3.5, the sequence of intermediate Petri net solution states occuring is *nonmonotonic* with respect to the set of fulfilled formulas. That is, a formula fulfilled at a certain state may be unfulfilled at a later state. In general, any change of the Petri net proposal will require the whole set of task formulas to be verified again. So if the system would offer completion proposals based on empirical design heuristics, it could happen that previously fulfilled formulas would be unfulfilled again at later stages of solution development.

Def:

- K_i is node #i in the case graph, K_i is a set of atoms
- $S(K_i) := \{K_j \mid K_j$ is immediate successor of node #i in the case graph$\}$
- $\mathrm{Path}(K_i) := \{ (j_1, \ldots, j_n) \mid K_{j_1} = K_i \quad \wedge$
  $K_{j_{i+1}} \in S(K_{j_i}) \quad \wedge$
  $K_{j_{i+1}} \notin \{K_{j_1} \ldots K_{j_i}\} \quad \wedge$
  $(S(K_{j_n}) = \emptyset \quad \vee$
  $\exists\, K_p \in S(K_{j_n}) \mid K_p \in \{K_{j_1} \ldots K_{j_n}\})\}$

$K_i(\mathrm{atom}) = t$	iff	atom $\in K_i$
$K_i(a \wedge b) = t$	iff	$K_i(a) = t \wedge K_i(b) = t$
$K_i(a \vee b) = t$	iff	$K_i(a) = t \vee K_i(b) = t$
$K_i(a \rightarrow b) = t$	iff	$K_i(a) = f \vee K_i(b) = t$
$K_i(\neg a) = t$	iff	$K_i(a) = f$
$K_i(\bigcirc a) = t$ (nexttime a)	iff	if $S(K_i) = \emptyset$ then f else $\forall\, K_j \mid K_j \in S(K_i)$: $K_j(a)$
$K_i(\Diamond a) = t$ (eventually a)	iff	$\forall\, P \in \mathrm{Path}(K_i)\ \exists\, j \in P$: $K_j(a)$

$K_i(\Box a) = t$ (always a)	iff	$\forall\, P \in \mathrm{Path}(K_i)\ \forall\, j \in P$: $K_j(a)$

Table 1: Semantics of the temporal logic used in PETRI-HELP

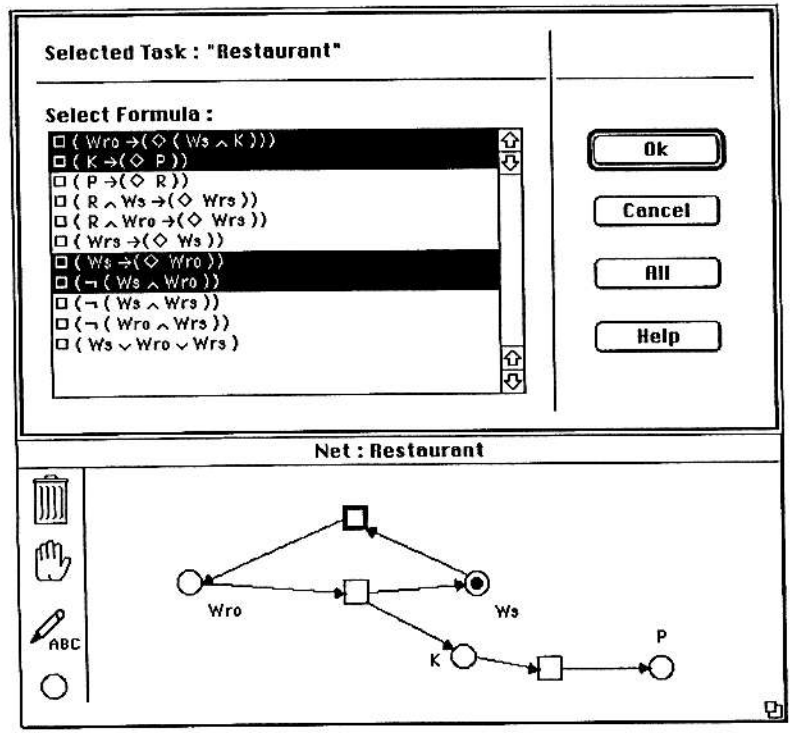

Figure 3.9: Stating a hypothesis: The PS marks the task formulas he considers fulfilled by the current state of the solution proposal

This is undesirable because it will cause secondary impasses and confuse the novice. PETRI-HELP's approach to this problem is to identify solution states that fulfill a superset of the task formulas that were fulfilled by the previous state. When a PS constructs a Petri net, we call a solution state *safe*

- if it is the starting state (the empty net), or
- if the task formulas fulfilled by it are a proper superset of the task formulas fulfilled by the previous safe state created by the PS.

Based on safe states, we can define *design rules* which can be used for completion and correction proposals. There are two kinds of design rules:

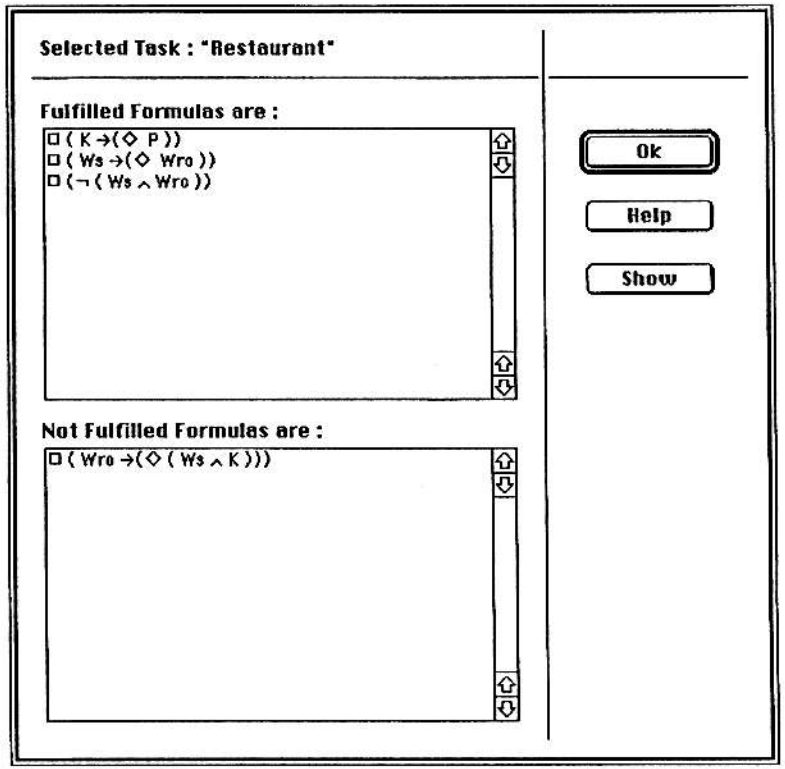

Figure 3.10: Feedback to the hypothesis of Figure 3.9

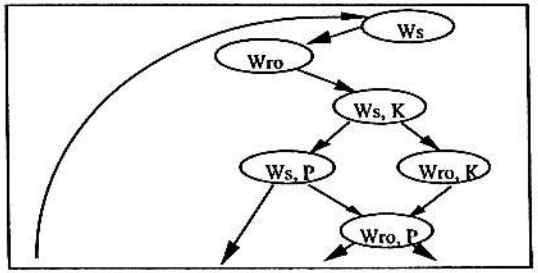

Figure 3.11: Part of the case graph of the Petri net of Figure 3.8

a) $net_i \rightarrow net_j$, where net_i and net_j are safe solution states obtained by the PS. $net_i \models F_i$, $net_j \models F_j$ and $F_j \supset Fi$. F_i, F_j are subsets of the task specification. After reaching net_i, net_j is the very next safe state reached by that PS. That is, the PS did not reach a net_k, $net_k \models F_k$, such that $F_j \supset F_k \supset F_i$.

If the actual solution state of the PS is net_i, then the difference between net_j and net_i can be used to propose a net completion.

b) $net_i \models F_i$.

If the actual solution state of the PS is net_i, and there is a rule "$net_j \models F_j$" and $F_j \supset F_i$, then

- the difference between net_i and net_j (the part of net_i not contained in net_j) can be used for a correction proposal: places, transitions, and arrows that have to be deleted.
- the difference between net_j and net_i (the part of net_j not contained in net_i) can be used to propose a completion: places, transitions, and arrows that have to be added.

These two types of design rules are learned by the system from interactions of PSs with the system while constructing Petri nets. In one mode, PETRI-HELP does model checking after each editing step of the PS. If a new safe solution state is identified, the two types of rules stated above are created

- by associating the last safe state with the current safe state (rule type a), and
- by associating the current safe state with the set of formulas it is a model of (rule type b).

These two kinds of rules learned by the system are the basis of the completion and correction proposals given by the system. By making use of these rules, completion and correction proposals make sure that previously fulfilled formulas remain satisfied, thus avoiding secondary impasses and confusion of the learner.

Figure 3.12 shows a completion proposal generated during the development of the net of Figure 3.8. The recommendation is to add a transition connected to the place P by an arrow leaving from P. The right of Figure 3.13 shows an example for a completion and correction proposal. If the PS asks for information for the net on the left of Figure 3.13, the system recommends

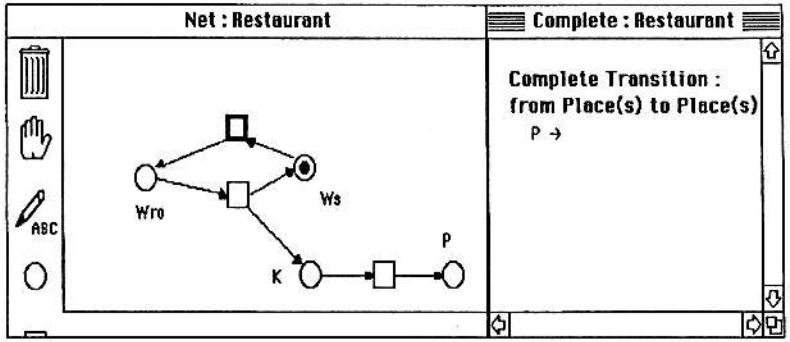

Figure 3.12: Completion proposal

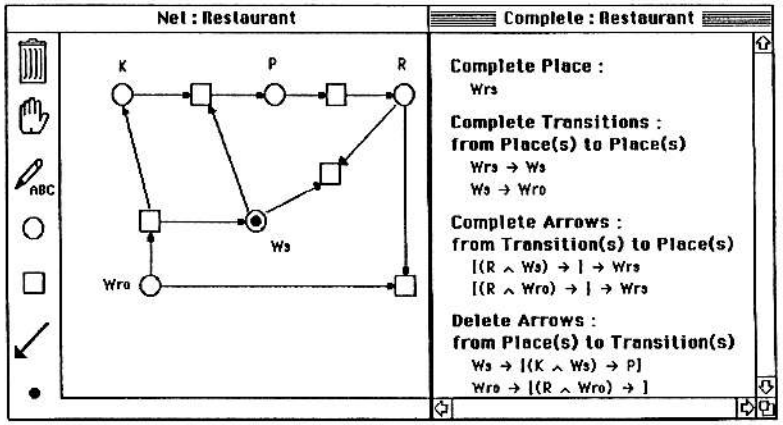

Figure 3.13: Completion and correction proposal

- *completing a place:*
  to add a place Wrs
- *completing transitions:*
  to add a transition with the preset {Wrs} and the postset {Ws} ("Wrs → Ws" on the right of Figure 3.13), and another transition with the preset {Ws} and the postset {Wro} ("Ws → Wro" on the right of Figure 3.13)
- *completing arrows:* to add an arrow from the transition with the preset {R, Ws} and an empty postset (denoted on the right of Figure 3.13 by "[(R ∧ Ws) →]") to the new place Wrs, and another arrow from the transition with the preset {R, Wro} and an empty postset (denoted on the right of Figure 3.13 by "[(R ∧ Wro) →]") to the new place Wrs
- *deleting arrows:* to delete the arrow from the place Ws to the transition with preset

{K, Ws} and postset {P} (denoted on the right of Figure 3.13 by "[(K ∧ Ws) → P]"), and to delete the arrow from the place Wro to the transition with preset R, Wro and an empty postset (denoted on the right of Figure 3.13 by "[(R ∧ Wro) →]").

PETRI-HELP generates completion and correction proposals such that the smallest possible proper superset of the actually fulfilled formulas is fulfilled. The reason for this is that the PS should receive only minimal help information that is sufficient for him to overcome the actual impasse. Figure 3.12 shows a situation where the learner gets only a small amount of information, whereas in Figure 3.13 a lot more information is given. The reason for this is that Figure 3.13 shows a less common situation where only few rules have been learned for by the system. In general, the amount of information delivered to the PS on request depends on the amount of learning of the system: As long as PETRI-HELP has not learned much, the rules tend to bridge large gaps in the development of a Petri net solution. But the more PETRI-HELP learns, the smaller the gaps bridged by the rules tend to get, and so less information tends to be delivered to the PS at a time.

3.5 Empirical Work Supporting the Development of PETRI-HELP

With the first version of the implemented system, we conducted empirical investigations within a computer science postqualification course for teachers, and a practice course for students. With the implemented system, our main questions were to find out

- how the subjects made use of and accepted the possibility to test hypotheses and to get feedback about fulfilled and unfulfilled formulas,
- how the subjects made use of and accepted the completion / correction proposals.

The subjects participating in the practice course filled out an evaluation sheet where they were asked to comment on their impasses and on their use of the various PETRI-HELP features. Most subjects were able to solve the whole sequence of PETRI-HELP tasks within a few hours. We found that testing hypotheses was made use of and accepted widely, even after the subjects acquired some expertise. We propose that the reason for this is that receiving feedback about fulfilled and unfulfilled parts of the task specification directs the subjects' attention to where to proceed with the task, but without forestalling the solution.

In contrast, the completion and correction proposals were less widely used and accepted. The subjects' comments indicated that there seemed to be three reasons for this:

- In several cases, the subjects did not consider the information received as plausible. They had problems to see why the proposal would take them a step further. In some cases they tried to self-explain the information given [7], but sometimes they just

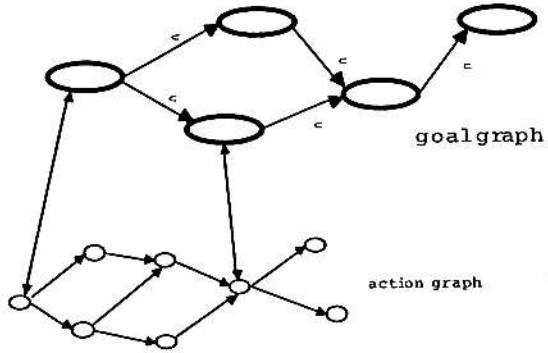

Figure 3.14: Correspondence between the two learned graphs. Each node in the goal graph consists of a subset of the set of specification formulas, each node in the action graph is a Petri net created by the user. The links between these two graphs link goals to nets fulfilling them.

3.6.2 Model Checking Based Explanations

In PETRI-HELP, the following two kinds of situations require explanations:

- Why is an unfulfilled formula of the task specification in fact unfulfilled?
- Why is the actual completion / correction proposal in fact proposed?

The approach implemented in PETRI-HELP is concerned with the first kind of explanations ("Why not" explanations). It would be easy though to extend this approach to the second kind of explanations.

In order to do this, it would be necessary to show

- *what* additional formulas would be fulfilled if the user accepts the completion / correction proposal
- *why* these formulas would in fact be fulfilled after accepting the proposal
- *why* these formulas are *not* fulfilled by the present net.

In our empirical studies, we observed that human tutors tended to explain unfulfilled formulas to the learner by simulating (parts of) the net. That is, the fact that a formual is not satisfied is verified by showing that a marking state of the net can be reached that violates the formula. The model checking based explanation component of PETRI-HELP implements this explanation pattern. Information is given in a stepwise manner (see [32] for detail). This corresponds to our theoretical position that only as much help information should be given as needed to resolve the actual impasse by self explanation.

For any unfulfilled formula of a hypothesis, the learner may ask for an explanation ("Why is this formula not satisfied?"). The explanation component assumes always-quantified formulas.

- On the first level of explanation, the system informs the learner that it is possible to reach a marking state of the net where the formula is not true. This information may already suffice for the learner to see why the formula is in fact unfulfilled, so he might stop asking for further information here.
- On the second level of explanation, the formula is decomposed. In case of an implication, the learner is informed that a marking state of the net might be reached that with respect to the formula is to be interpreted such that the premise of the formula is true, but its comclusio is not. In case of a conjunction, the learner is informed that a state can be reached where at least on conjunct is false. In case of a disjunction, the learner is informed that a state can be reached where all disjuncts are false.
- On the third level of explanation, the learner is informed about the reasons for the situation described on the second level. This level of explanation makes use of three basic concepts: deadlock, circle, and alternative-possible. "Deadlock" means that the reason for the formula being unfulfilled is that the net may reach a state where no transition is able to fire. In case of an implication, this means that a state can be reached that corresponds to the formula's premise being true, it's conclusio being false, and no transition able to fire. "Circle" means that that the reason for the formula being unfulfilled is that a circular sequence of marking states is possible so that certain places may never get tokens. In case of an implication, this means that all states of this sequence correspond to the formula's premise being true but it's conclusio being false. "Alternative possible" only applies for implications containing the "nexttime" operator. It says that the reason for the formula being unfulfilled is that a marking state is possible that corresponds to the formula's premise being true but it's conclusio being false because it may not necessarily true in the next time step (only at some later time step).
- Finally, on the fourth level of explanation, the sequence of marking states leading to the situation decribed at the third level (that is, deadlock, circle, or alternative-possible) is visualized to the learner by net simulation so the learner can see how the critical marking state(s) may be brought about.

3.6.3 Incorporating Viewpoint Centered Planning into PETRI-HELP

As stated, the ISP-DL Theory and empirical studies led to a common implication for PETRI-HELP: Help information should be provided on a level more abstract than places, transitions, and links. This information should support planning and hypotheses testing of plans especially at the early stages of problem solving, and it should support the stepwise transformation of plans into more detailed ones, until a solution is reached.

We investigated an approach developed by Olderog [29] that allows to transform a task description (a specification) into a condition-event Petri net by making use of intermediate specifications as well as "mixed terms" (terms composed of specification fragments and Petri net fragments). Basically, the intention of Olderog is to derive a description of the operational behavior of concurrent processes from a set of logical formulas specifying these processes. A transformation begins with a set of formulas and constructs a process term from it by transformation rules. The process term expresses a possibly concurrent process in an abstract programming language. A Petri net, with an explicit representation of concurrency, defines the semantics of the process term. It can be derived from the term by net construction rules. So a derivation chain can be constructed from the specification to the Petri net.

Based on Olderog, Wedig [44] recently developed a method for the derivation of Petri nets from trace specifications directly without using process terms. Based on this work transformational steps were integrated into PETRI-HELP [15].

3.6.3.1 Viewpoint Centered Specification

The specification of a process is stated in trace logic (i.e., [18]. As a simple example, Table 2 shows a possible trace-logic specification of (a variant of) the "Restaurant" task which we will call "Bavarian Biergarten":

Trace of events in the "Bavarian Biergarten": Ws. @. @. Wro. K. P. @. R. Wrs. Ws. @. Wro. K. @. P ...
Trace logic specification of the "Bavarian Biergarten": trace ↓ {K, P, R} ∈ pref(K.P.R)* ∧ trace ↓ {Ws, Wro, K, R, Wrs} ∈ pref(Ws.Wro.K.R.Wrs)*
View of kitchen: K.P.R.K.P.R.... View of waiter: Ws.Wro.K.R.Wrs.Ws.Wro.K.R.Wrs...

Table 2: Trace and trace logic specification of the "Bavarian Biergarten"

The upper part of Table 2 shows a possible trace. Ws, Wro, K, P, R, Wrs have the same meaning as in Figure 3.4. @ denotes things that may happen but are of no concern here.

From the kitchen's viewpoint, there are only three relevant events: K, P, R. This is expressed by "trace ↓ {K, P, R} ∈ pref(K.P.R)*", where ↓ is the projection operator, and "pref(K.P.R)*" is {ε, K, K.P, K.P.R, K.P.R.K, K.P.R.K.P, K.P.R.K.P.R, ...} (ε is the empty trace). Similarly, the waiter's point of view is a succession of sleeping (Ws), taking orders (Wro), passing them to the kitchen (K), being told that a meal is ready (R), and serving it (Wrs).

From a knowledge acquisition perspective, an attractive feature of trace specifications is that they emphasize the acquisition of the knowledge needed for specifying a system

by looking at it from different viewpoints (like the kitchen's viewpoint and the waiter's viewpoint in our example, see the lower part of Table 2). Thus specifications of complex systems can be constructed by acquiring the knowledge from the agents participating in it.

3.6.3.2 Viewpoint Centered Planning and Implementation

In order to use the transformational approach within PETRI-HELP, several simplifications were made. Figure 3.15 shows our graphical representations of some transformation rules. In these representations, each rule has three parts. The upper part contains the name of the rule, the middle part may contain conditions, and the lower part contains a statement. The parallelism rule has no condition. It visualizes the equivalence of a conjunction of terms S and T (S ∧ T) to two specifications S and T that have to be implemented as parallel nets. On the PETRI-HELP screen, we may create two *goal regions* labeled by S and T, which have yet to be implemented by Petri net fragments. Thus a goal region represents a specification of a task or subtask: the goal to create a Petri net fragment that is equivalent to that specification. The dotted crossing lines between the goal regions in Figure 3.15 mean that these nets will have to be synchronized (which is specified by the net combination rule). For example, applying the parallelism rule to the trace specification of the Bavarian Biergarten in Table 2 leads to the goal regions shown in Figure 3.16.

Now the two nets for the kitchen and for the waiter can be constructed separately and synchronized later. Figure 3.17 shows steps in the construction of the "kitchen net". Figure 3.17a shows its specification as a goal region. "init(S)", the set of next possible events of a process specified by S, might be empty (deadlock, not shown), contain one element (then the prefix rule is applicable), or more than one element (handled by the expansion rule). Since init(trace ↓ {K, P, R} ∈ pref(K.P.R)*) = {K}, the prefix rule applies, generating a place leading to a transition labeled with K leading to a goal region agai 73 n (Figure 3.17b). The new goal region represents the kitchen after having received an order from the waiter: "K.trace ↓ {K, P, R} ∈ pref(K.P.R)*". Figure 3.17b represents a mixed expression. Next, the prefix rule is applicable again, leading to Figure 3.17c. After three applications of the prefix rule (Figure 3.17d), the expression "K.P.R.trace ↓ {K, P, R} ∈ pref(K.P.R)*" will be obtained which is equivalent to the original expression "trace ↓ K, P, R ∈ pref(K.P.R)*". Thus the recursion rule is applicable (with S substituted by "trace ↓ K, P, R ∈ pref(K.P.R)*", and S' substituted by "K.P.R.trace ↓ {K, P, R} ∈ pref(K.P.R)*"). The recursion rule states that if a specification S is equivalent to a mixed expression containing a specification S', and S and S' are equivalent as well, then changing that mixed expression by removing S' and introducing recursion still keeps it equivalent to S. Figure 3.17e shows the result of its application.

trace ↓ {K, P, R} ∈ pref(K.P.R)*	trace ↓ {Ws, Wro, K, R, Wrs} ∈ pref(Ws.Wro.K.R.Wrs)*

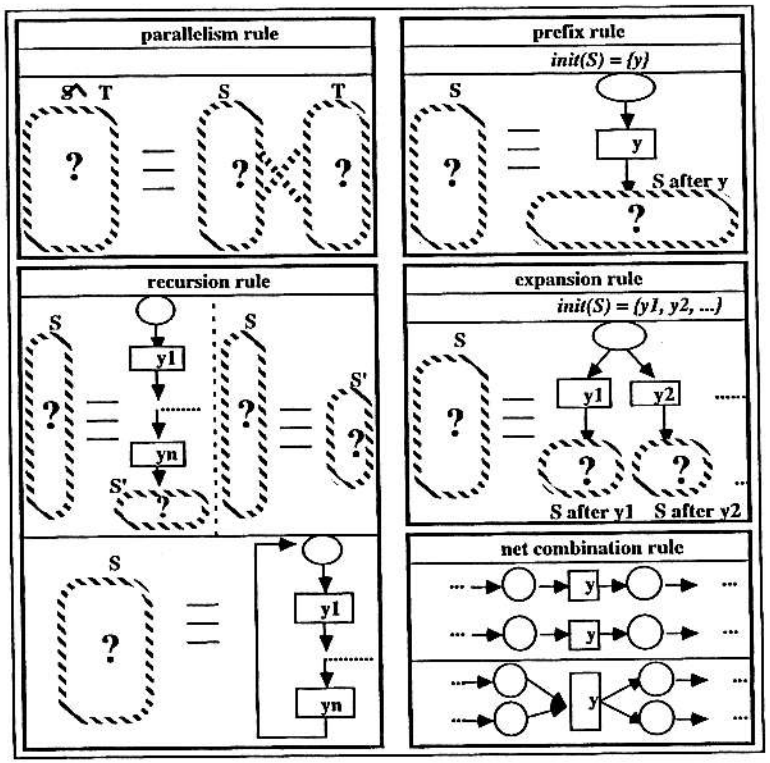

Figure 3.15: Some transformation rules

3.6.3.3 Viewpoint Centered Synchronization

When nets have been created for different viewpoints, they can be glued together to one single net by the net combination rule (Figure 3.15). The net combination rule combines nets for the two views in Table 2 to the net shown in Figure 3.18b. (K and R are the synchronizing transitions.)

Using the transformation approach, different *strategies* are possible, because we can take a look at the two different views of Table 2 simultaneously and thus avoid the parallelism rule. The result is shown in Figure 3.18a. Alternatively, the components of a net can be developed in parallel and combined later (Figure 3.18b). In more complex examples, intermediate strategies between maximal sequentiality and maximal parallelism are possible too.

3.6.3.4 Supporting planning of Petri nets in PETRI-HELP

This section describes how the transformational approach can support learners in constructing Petri nets:

1. If the learner does not know how to proceed and asks for a completion proposal, the system may *offer goal regions* (as illustrated in Figures 3.16 and 3.17 as help). So the learner is provided with descriptions of subtasks yet to be solved, and with recommendations how to decompose the task into subtasks. Thus the system will not be restricted to help on the level of places, transitions, and links.

2. The learner may state *hypotheses about goal regions*, not only about Petri net fragments. So the learner may get information whether he or she is "on the right track" at very early planning stages. For example, the learner may ask the system if it is appropriate to structure the problem of Table 2 into two parallel components (Figure 3.16) without bothering about what the components will exactly look like at this planning stage. So the learner may postpone implementation considerations and work with partial plans and mixed expressions.

3. The learner may receive direct guidance in Petri net construction by using the *transformation rules as help*. While our model checking approach allows for *free*, unconstrained problem solving because every solution proposal can be analyzed by the system, the transformational approach allows the learner to create a Petri net solution by stepwise application of the rules, that is, in a systematical, derivational way. Alternatively, the transformation rules can also be offered as explanations for completion proposals generated by the system.

In order to find out whether it is feasable for novices to use the transformation rules for the derivation of a Petri net, we carried out a single subject study. The subject was a novice concerning Petri nets. Her task was to create the "restaurant" net with paper and pencil, using graphical representations of the transformation rules as shown in Figure 3.15. The subject adopted a maximally parallel strategy. She needed some assistance for applying the parallelism and recursion rule, she did not immediately realize their applicability. But in general, she had no serious problems with this task. This preliminary result suggests that the approach is feasable as a basis for supporting novices in Petri net design.

So the transformation approach is a sound basis for letting the learner express initial ideas, partial plans, test hypotheses about them, and receive proposals from the system at the same level. The learner is enabled to think about specifications (and "mixed terms") without bothering about their implementation from the beginning. The transformational approach as stated here was implemented for the purposes of PETRI-HELP [15] but has not yet been tested empirically.

3.6.4 Supporting the creation of task specifications

In PETRI-HELP, the temporal logic approach allows the analysis of any solution proposal created by a PS by model checking, so it supports free, explorative, unguided problem solving. The trace logic approach allows the derivation of a solution and thus allows

systematic, rule-guided problem solving. Thus according to our theory, both approaches have their merits.

But neither approach addresses the problem of *specification development*. Thus, with respect to our theoretical framework (see Figures 3.2 and 3.3), the problem solving level of *deliberation* remains still uncovered. This means that the learner should be supported by PETRI-HELP in *generating* the specification of some system or process. Then the Petri net solution created by the learner would be checked against this specification. In assisting the learner to create a specification, the system may help the learner and help in a dialog to acquire and to integrate the knowledge needed.

We explored the possibility that the system's assistance in the development of a task specification may be organized as a Socratic dialog. Collins [9] developed a set of dialog rules that we think can be used to govern this process. Applied to the task of creating a task specification, some of Collins' rules can be restated in the following way:

1. Ask the user what agents will be involved in the system to be specified.
2. Ask the user about the states each agent can be in. This will lead to a set of *agent-state-pairs.*
3. Ask the user about which agent-state-pairs are mutually exclusive.
4. Ask the user about what agent-state-pairs, or conjunctions or disjunctions of agent-state-pairs, lead to what consequent agent-state-pairs, or conjunctions or disjunctions of agent-state-pairs.
5. For any agent-state-pair or conjunction or disjunction of agent-state-pairs that has been specified as being a consequence of some other agent-state-pair (s), ask the user whether the consequence is expected to be true in the next time step, or whether it is expected to be true at some later point in time.
6. Ask the user about the conjunct of agent-state-pairs expected to be true in the first time step.

The first two questions are used to establish the atomic formulas of the task specification. The third question delivers information about the exclusion conditions. The answers to questions 4 and 5 are used to establish the progress conditions of the task specification. Question 5 tries to acquire information needed for specifying the temporal logic operators. Finally, question 6 establishes the starting condition.

3.7 Cooperations

From the MOBY Project Group we received a lot of support especially in the earlier phases of the project. They provided us with their Petri net editor so we were able to do our empirical investigations. Especially Hans Fleischhack gave us a lot of support concerning theoretical problems which had to be solved for the design and development of PETRI-HELP.

Furthermore, there was cooperation with Bernhard Josko and the group of Werner Damm, concerning theoretical discussions as well as practical support in the design and implementation of the model checking approach.

There was also cooperation with Ernst-Rüdiger Olderog, concerning the integration of the transformation approach into PETRI-HELP. In several discussions he patiently answered our questions and provided us with a lot of additional material.

We thank Werner Damm, Hans Fleischhack, Bernhard Josko, and Ernst-Rüdiger Olderog for their support and for the very helpful discussions.

Outside the university, there was cooperation with the Competence Center Informatik (CCI), Meppen, concerning the development of help for medical personell being trained in the use of special medical equipment. Based on specifications of medical equipment for surgery we received from the CCI, we developed temporal logic task descriptions so Petri net solution proposals can be constructed, and feedback and completion proposals can be given. In cooperation with the educational institute of the chamber of industry and commerce (Bildungswerk des Deutschen Industrie- und Handelstages), we currently develop PULSE (Pneumatic Learning and Simulating Environment), an Intelligent Problem Solving Environment for the domain of pneumatic circuits, based on the foundations layed in the PETRI-HELP project. The following section will provide a brief description of PULSE.

3.8 PULSE: An Intelligent Problem Solving Environment for the Domain of Pneumatics

Several chambers of industry and commerce (including Oldenburg's) offer a three year training for those being employed as metalworkers leading to a master craftsman's diploma. This course includes a 50 hours section imparting the basic concepts of hydraulics and pneumatics. Due to the fact that only few laboratories exist where the participants can make practical experience in those subjects, we were asked to develop an intelligent problem solving environment for the domain of pneumatic circuit development. The system should offer task descriptions in the way the participants are used to, and support the development of related pneumatic circuits. It should offer the opportunity to test hypotheses about the correctness of the developed pneumatic circuits concerning the task descriptions, and it should offer feedback and help.

In these training courses hydraulics and pneumatics are taught as time discrete and state discrete systems. No differential equations are used to describe the behaviour of the system's components. This level of abstraction fostered the adaption of the methods used in PETRI-HELP for task description and hypotheses testing.

3.8.1 Task Description and Solution Development in the Domain of Pneumatics

In the training section of these courses, task descriptions in the domain of pneumatics are presented twofold. A description of the situation the problem arises in is supported by a formal functional diagram which describes the intended dynamic behaviour of the pneumatic circuit to be developed. The solution is to be developed as a pneumatic circuit including several pneumatic elements like valves, pumps, cylindes, switches, and pipes connecting these elements.

Like PETRI-HELP, PULSE offers several tasks the problem solver may choose from. These tasks are presented in the way the problem solvers are used to. This includes a formal functional specification (as shown in Figure 3.19), and a text describing the situation. For the development of a solution a graphical editor is used which offers tools to insert, delete, manipulate, and compose elements of a pneumatic circuit (see Figure 3.20).

3.8.2 Hypotheses Testing

In accordance with the ISP-DL theory, the problem solver has the opportunity to state a hypothesis about his solution proposal at any time. This is done by marking these parts of the functional diagram which describe the behaviour, the problem solver assumes his circuit proposal should have. As a feedback the hypothesis is returned in a seperate window with the fulfilled behaviour marked in green, and the unfulfilled marked in red.

For checking hypotheses the functional description is transformed to temporal logic formulae and the pneumatic circuit is transformed to a finite state automaton [13]. This is done by composing the finite state automata which describe the possible behavior of the pneumatic elements. After that, model checking takes place checking the validity of the formulas according to the finite state automaton. So it was possible to apply the approach developed in PETRI-HELP to the domain of pneumatic circuits.

3.8.3 State of Development, Experiences, and Further Components

To date, PULSE includes several task descriptions problem solvers may choose and work on. This is supported by an editing environment that offers exactly the pneumatic elements' symbols (DIN ISO 1219) he is used to work with. At any time a hypothesis specifying the intended behaviour of the developed circuit can be stated. Feedback is given about the behaviour already shown by the solution proposal, and the behaviour that could not yet be observed. Furthermore, PULSE has an explanation component that explains the system's feedback and information on a conceptual level [45].

At the end of 1995, a first version of PULSE has been delivered to the DIHT-Bildungsgesellschaft, Bonn. It was distributed among several chambers of industry and commerce where it is tested with participants of the training courses.

At the moment several components are under development that are designed to support the problem solver as well as the lecturers. A simulation tool will support the problem solvers by highlighting the behaviour of the developed pneumatic circuit step by step. This helps detecting problems especially after a hypothesis was rejected. Especially the lecturers indicated their interest in a task editing tool to develop formal task descriptions. Such a tool is currently being developed to support the development of formal functional diagramms.

3.9 Acknowledgement

We would like to express our thanks to the following students: Alexander Grosse, Thomas Herlyn, Oliver Jordan, Frank Möhle, Hagen von Stuckrad, Janine Willms, Stefan Zorn.

3.10 References

[1] Allen, J., Kautz, H., Pelavin, R., Tenenberg, J. (1991). Reasoning about Plans, San Mateo, Ca.: Morgan Kaufman.

[2] Anderson, J.R. (1983). The architecture of cognition. Cambridge: Harvard University Press.

[3] Anderson, J.R. (1986). Knowledge compilation: The general learning mechanism. In R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds): Machine learning, Vol. II. Los Altos: Kaufman, 289-310.

[4] Anderson, J.R. (1989). A theory of the origins of human knowledge. Artificial Intelligence, 40, 313-351.

[5] Brown, J.S., van Lehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. Cognitive Science, 4, 379-426.

[6] Carberry, S. (1989). Plan Recognition and its Use in Understanding Dialog, in A. Kobsa, W. Wahlster (eds), User Modeling in Dialog Systems, Berlin: Springer.

[7] Chi, M.T.H., Bassok, M., Lewis, M., Reimann, P., Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems, Cognitive Science, 13, 145-182.

[8] Clarke, E.M., Emerson, F.A. & Sistla, A.P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. on Programming Languages and Systems, 8(2), 244-263.

[9] Collins, A. (1977). Processes in Acquiring Knowledge, in R.C. Anderson, R.J. Spiro, W.E. Montague (eds), Schooling and the Acquisition of Knowledge, Hillsdale: Erlbaum, 339-374.

[10] Damm, W., Döhmen, G., Gerstner, V., Josko, B. (1990). Modular Verification of Petri Nets. The Temporal Logic Approach, in J.W. de Bakker, W.P. de Roever, G. Rozenberg (eds), Proceedings REX-Workshop on stepwise refinement of distributed systems: models, formalisms, correctness. Berlin: Springer, LNCS 430.

[11] Doignon, J.-P., Falmagne, J.-C. (1985). Spaces for the Assessment of Knowledge, Int. Journal of Man-Machine Studies, 23, 175-196.

[12] Fleischhack, H., Lichtblau, U. (1993). MOBY - A Tool for High Level Petri Nets with Objects. IEEE Int. Conf. on Systems, Man and Cybernetics, Vol. 4, 644-649.

[13] Göhler, H. (1995). Entwicklung eines Hydraulik-Problemlösemonitors. Universität Oldenburg, Fachbereich Informatik, Diplomarbeit.

[14] Gollwitzer, P.M. (1990). Action phases and mind-sets. In E.T. Higgins & R.M. Sorrentino (eds): Handbook of motivation and cognition: Foundations of social behavior, Vol.2, 53-92.

[15] Grosse, A. (1995). Prolog-Implementation der Entwicklung von Petri-Netzen aus Spezifikationen, Universität Oldenburg, Fachbereich Informatik, Studienarbeit.

[16] Grunst, G., Oppermann, R., Thomas, C. (1993). Benutzungsmodellierung bei kontext-sensitiver Hilfe und adaptiver Systemgestaltung, in: A. Kobsa, W. Pohl (Hrsg.): Arbeitspapiere des Workshops 'Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen', Berlin, 13.-15.9.93. WIS-Memo7, AG Wissensbasierte Informationssysteme, Informationswissenschaft, Universität Konstanz.

[17] Heckhausen, H. (1989). Motivation und Handeln, Berlin: Springer (2nd ed.).

[18] Hoare, C.A.R. (1985). Communicating sequential processes. Englewood Cliffs: Prentice Hall.

[19] Josko, B. (1990). Verifying the correctness of AADL modules using model checking. In J.W. de Bakker, W.P. de Roever, G. Rozenberg (eds): Proceedings REX-Workshop on stepwise refinement of distributed systems: models, formalisms, correctness. Berlin: Springer, LNCS 430, 386-400.

[20] Kammermeier, F. (1987). Einführung in Petri-Netze und Ansätze zu ihrer Verwendung bei der Analyse dynamischer Systeme, TEX-B Memo 27-87, Universität Kaiserslautern / Fraunhofer-Gesellschaft Karlsruhe.

[21] Kröger, F. (1987). Temporal Logic of Programs, Berlin: Springer.

[22] Laird, J.E., Rosenbloom, P.S., Newell, A. (1986). Universal subgoaling and chunking. The automatic generation and learning of goal hierarchies. Boston: Kluwer.

[23] Laird, J.E., Rosenbloom, P.S., Newell, A. (1987). SOAR: An architecture for general intelligence. Artificial Intelligence, 33, 1-64.

[24] Möbus, C. (1995). Towards an Epistemology of Intelligent Problem Solving Environments: The Hypotheses Testing Approach. in J. Greer (ed): Proceedings AI-ED 95, World Conference on Artificial Intelligence in Education, Washington, D.C., AACE, 138-145.

[25] Möbus, C., Pitschke, K., Schröder, O. (1992). 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, 294-301.

[26] Möbus, C., Schröder, O., Thole, H.-J. (1991). 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, 137-143.

[27] Möbus, C., Schröder, O., Thole, H.-J. (1992). A model of the acquisition and improvement of domain knowledge for functional programming. Journal of Artificial Intelligence in Education, 3(4), 449-476.

[28] Newell, A. (1990). Unified theories of cognition. Cambridge: Harward University Press.

[29] Olderog, E.-R. (1991). Nets, terms, and formulas. Cambridge: Cambridge University Press.

[30] Pitschke, K. (1994). User Modeling for Domains without Explicit Design Theories, in: Proceedings of the Fourth International Conference on User Modeling (UM94), 15.-19. August 1994, Hyannis, MA, USA, The MITRE Corporation, Bedford, MA

[31] Pitschke, K., Schröder, O., Möbus, C. (1991). Entwurf eines Hilfesystems für Petrinetzmodellierer, in P. Gorny (Hrsg.), Informatik und Schule 1991, Berlin: Springer (Informatik-Fachberichte 292), 299-305.

[32] Pitschke, K., Schröder, O., Möbus, C. (1995). Erklärungsgenerierung in PETRI-HELP, in F. Huber-Wäschle, H. Schauer, P. Widmayer (Hrsg.), GISI 95 - Herausforderungen eines globalen Informationsverbunds für die Informatik. Berlin: Springer, 304-313.

[33] Polson, P.G., Kieras, D.E. (1985). A Quantitative Model of the Learning and Performance of Text Editing Knowledge. CHI 85 Proceedings, 207-212.

[34] Polson, P.G., Muncher, E., Engelbeck, G. (1986). A Test of Common Elements Theory of Transfer, CHI 86 Proceedings, 78-83.

[35] Reisig, W. (1986). Petrinetze - eine Einführung, Berlin: Springer.

[36] Reisig, W. (1992). A primer in Petri net design, Berlin: Springer, 1992.

[37] Rich, E. (1989). Stereotypes and User Modeling, in A. Kobsa, W. Wahlster (eds), User Modeling in Dialog Systems, Berlin: Springer.

[38] Rosenbloom, P.S., Laird, J.E., Newell, A., McCarl, R. (1991). A preliminary analysis of the SOAR architecture as a basis for general intelligence, Artificial Intelligence, 47, 289-305.

[39] Schröder, O., Möbus, C., Pitschke, K. (1993). Design of Help for Viewpoint Centered Planning of Petri Nets, in B. Brna, S. Ohlsson, H. Pain (eds), Proceedings of the 5th International Conference on Artificial Intelligence and Education (AI-ED 93), Association of the Advancement of Computing in Education (AACE), 370 - 377.

[40] Spada, H. (1976). Modelle des Denkens und Lernens. Bern: Huber.

[41] Valiant, L. (1984). A Theory of the Learnable. Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, 436-445.

[42] Van Lehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl, A. Lesgold (eds): Learning issues for intelligent tutoring systems. New York: Springer, 19-41.

[43] Van Lehn, K. (1991). Rule acquisition events in the discovery of problem solving strategies. Cognitive Science, 15, 1-47.

[44] Wedig, A. (1993). Kalküle zur Entwicklung von Petri-Netzen aus Spezifikationen, Diplomarbeit am Fachbereich Informatik der Universität Oldenburg.

[45] Willms, J. (1996). Eine konzeptbasierte Erklärungskomponente für eine intelligente Problemlöseumgebung im Bereich der Pneumatik. Universität Oldenburg, Fachbereich Informatik, Diplomarbeit.

[46] Wolff, J.G. (1987). Cognitive development as optimisation. In L. Bolc (ed): Computational models of learning. Berlin: Springer, 161-205.

[47] Wolff, J.G. (1991). Towards a theory of cognition and computing. Chichester: Ellis Horwood.

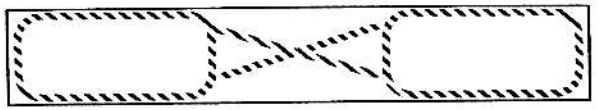

Figure 3.16: Applying the parallelism rule to the trace specification in Table 2

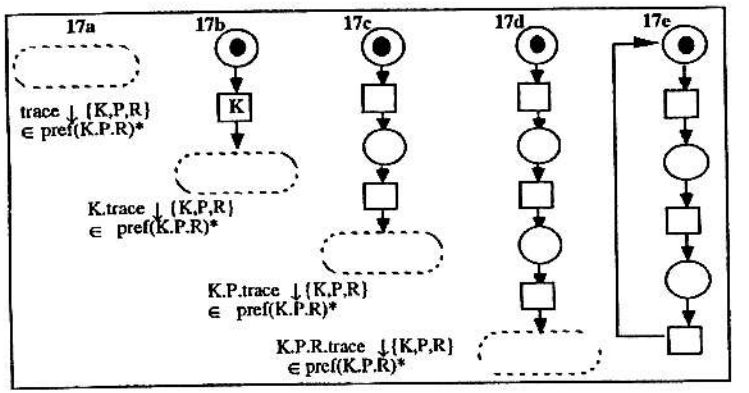

Figure 3.17: Petri net construction, using prefix and recursion rule

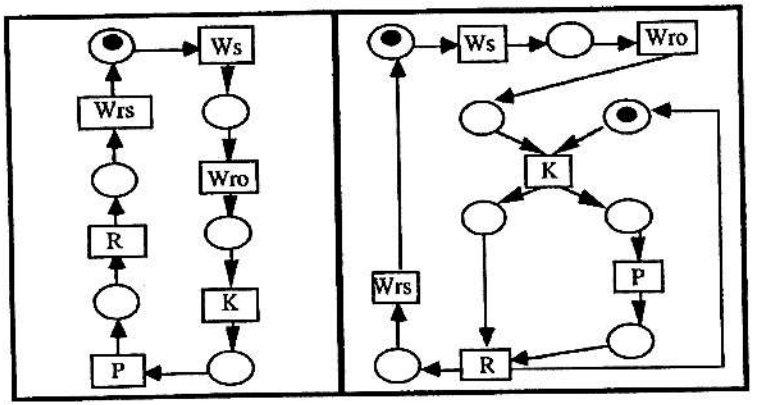

Figure 3.18: Two Petri nets derived from the trace specification in Table 2. Figure 18a (on the left): net derived by a sequential strategy, Figure 18b (on the right): net derived by a parallel strategy

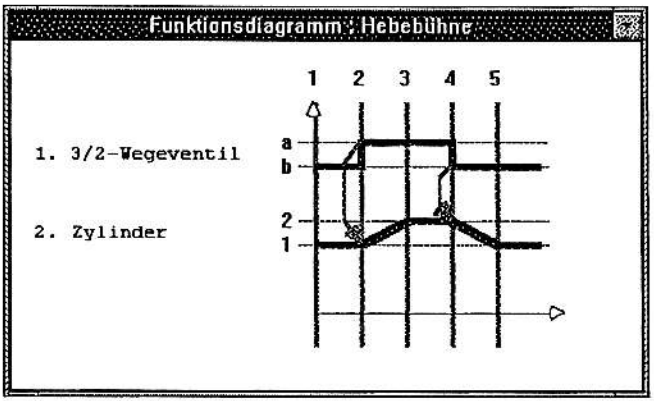

Figure 3.19: Task description as a functional diagram

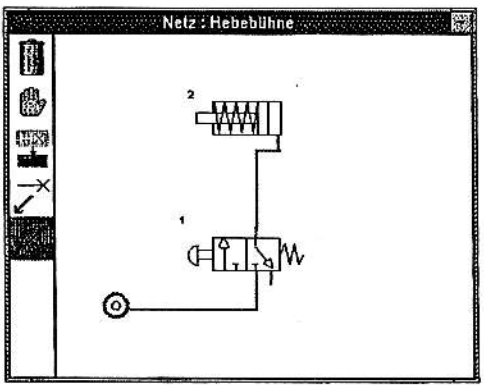

Figure 3.20: The pneumatic circuit editing field including a solution for the task described in Figure 3.19.

AIS --- AIS --- AIS --- AIS

Final Report

November 1996

Arbeitsgruppe Informatik-Systeme

FB Informatik - Universität Oldenburg

Preface

This report gives an overview of the work done by the 'Arbeitsgruppe Informatik-Systeme' (AIS) at the Department of Computer Science of the University of Oldenburg during the years 1990 to 1995. The 'Arbeitsgruppe Informatik-Systeme' has been supported by the 'Niedersächsisches Vorab der Volkswagen-Stiftung' (Az. 210-70631/9-13-14/89).

The AIS was founded to investigate principles, methods and prototypical tool implementations for the development of complex software and hardware systems. Eight main projects have been supported by the AIS:

1. MOBY: Modelling and analysis of office procedures by Petri Nets
   (Prof. Dr. V. Claus, Dr. H. Fleischhack),
2. DNS: Distributed simulation of high-level Petri Nets
   (Prof. Dr. M. Sonnenschein),
3. Petri-Help: Intelligent support for modelling time-discrete distributed systems with Petri Nets
   (Prof. Dr. C. Möbus),
4. COMDES: Specification, verification and simulation of computer architecture design
   (Prof. Dr. W. Damm),
5. Structure and behaviour of finite distributed automata
   (Prof. Dr. V. Claus, Prof. Dr. E. R. Olderog),
6. X-Fantasy: Interface design and implementation for multimedia applications
   (Prof. Dr. H.-J. Appelrath),
7. MUSE II: User interface design with respect to software-ergonomic criteria
   (Prof. Dr. P. Gorny),
8. Integrated ISDN Systems Concepts: Computer supported cooperative work using ISDN technology
   (Prof. Dr. P. Jensch).

Generally spoken, the first five projects work on modelling and analysis of complex systems, while the last three projects orientate towards multimedia applications. Thus, AIS projects dealt with topics like modelling of business processes, computer supported cooperative work, user interface design, distributed computing or hardware design.

The Department of Computer Science at the University of Oldenburg was officially founded in 1988. So, the opportunity of basic research in AIS projects was essential for its development and establishment. As one result of this process, the institute OFFIS was founded at Oldenburg, where many ideas, attempts and results of AIS projects still influence research activities and applications.

The editor wants to give his best thanks to Anja Gronewold who was responsible for the set up and layout of this report.

Oldenburg, November 1996

Michael Sonnenschein

integrated the proposal into their solution without understanding it, and sometimes they were even upset. These effects of feedback and help information have also been expected by our ISP-DL Theory [27].

- Proposing parts of the solution takes away part of the problem solving from the PS. In some cases the information delivered is too detailed. Probably, sometimes a more abstract hint instead of a Petri net fragment would have been sufficient as help information.
- In some cases, more information was given in the proposals than actually needed by the subject. But as mentioned earlier, this is a problem that diminishes when the system learns more rules.

These results suggest the following directions of further development of PETRI-HELP:

- According to the ISP-DL Theory, completion and correction proposals should not cause a big surprise and thereby lead to another impasse. So these proposals should be adapted to the *actual knowledge state* of the PS.
- The system should be able to *explain* its completion and correction proposals.
- Completion and correction proposals should not only be given as parts of the final solution, but also at a *more abstract* "planning" level. This is also what our ISP-DL Theory recommends: Help information should be given at the "synthesize" phase as well (Figure 3.3).

Subsequent work on PETRI-HELP was aimed at these three directions. The next subsection describes work aimed at the design of a user model. After that, the explanation facilities of PETRI-HELP are described, and an approach for supporting abstract planning. Finally, as mentioned before, the task of modelling does not only consist of creating net models for given specifications of time-discretized distributed systems, but also of specifying such systems as well. So we also show how PETRI-HELP supports the development of specifications.

3.6 Extensions of PETRI-HELP

3.6.1 Modelling the User

Up to now, there was no possibility in PETRI-HELP to adapt the system's help facilities to the actual user's needs. The following section describes our approach to system behavior adapted to the user [30].

For this objective is necessary to know what the user is planning to do. A plan is considered to be a sequence of actions that transforms a certain state to a final state [1]. Plan recognition means identifying well known planning operations in a given sequence of actions. Planning as well as plan recognition requires a planning space and knowledge about

the planning domain. Our approach, being designed to detect plans in PETRI-HELP, is applicable to every planning domain where no explicit domain knowledge is available. It only requires an oracle (in the sence of Valiant, 1984) to classify the user's solutions as correct or incorrect. In PETRI-HELP, this is done by model checking. Since in the Petri net domain there is no explicite design theory for developing a net for a specification, our approach is to let the system *learn* the search space of possible goals and solutions (Petri net proposals) from problem solving sessions of users. The idea is to identify a user's solution path in this search space to derive predictions about next steps from the identified path, and to use these predictions for giving help information how to proceed with the net proposal.

Therefore, goals of the user as well as their realizations have to be detected by the system. Every hypothesis the user states about his (sub-)net is interpreted as a subgoal. The hypotheses consist of (sub-)sets of the temporal logic specification formulas. These subgoals form the goal graph (problem space).

Every subgoal the user tests by model checking corresponds to a Petri net developed by the user. This net is expected to solve the tested goal. If the hypothesis holds, the net is associated with the stated subgoal. From these nets, the so called action graph is created. It consists of all nets fulfilling subgoals, and all the nets the user traverses to reach nets fulfilling subgoals.

Thus two search spaces are established (Figure 3.14): one for the goals and one for their realizations. Both are linked by correspondences between nets and the goals they fulfill (some kind of goals-means-relation, [26]. These spaces are built up by observing users, ranging from novices to experts. In these graphs, all the information that can be detected during the problem solving sessions, like all user actions, times the actions require, and the frequency of decisions are stored.

For the purpose of prediction of goals and actions, the actual user is identified within these graphs. The system's prediction is that the goals or actions with the highest probability, will be performed by the user, given the observed actions and goals (or their generalizations) of the user as a precondition. So there is no actual user model with assumptions about the user and conclusions drawn from them [37, 6], but a kind of usage model [16] with user identification.The process of generating help using this model is split into two steps:

- First the actual goal of the user must be identified. The system predicts the current goal by relating the user's history to the goal graph and choosing the goal with the highest probability. The probabilities are derived from the recorded behavior of former users.
- The second step predicts the action that the PS will most likely take for reaching that goal. As on the goal level, the next net with the highest probability is identified. The difference between this net and the user's net is used for generating help information.

Since in this approach help is based on the most probable continuation of the proposal, given the actual state of the solution (that is, the user's path through the goal graph and action graph), the information delivered takes account of the actual problem solving behavior of the user.