Engineers Don't Search

Benno Stein

Paderborn University
Department of Computer Science
D-33095 Paderborn, Germany
stein@upb.de

Abstract. This paper is on the automation of knowledge-intensive tasks in engineering domains; here, the term "task" relates to analysis and synthesis tasks, such as diagnosis and design problems.

In the field of Artificial Intelligence there is a long tradition in automated problem solving of knowledge-intensive tasks, and, especially in the early stages, the search paradigm dictated many approaches. Later, in the modern period, the hopelessness in view of intractable search spaces along with a better problem understanding led to the development of more adequate problem solving techniques.

However, search still constitutes an indispensable part in computer-based diagnosis and design problem solving—albeit human problem solvers often gets by without: "Engineers don´t search" is my hardly ever exaggerated observation from various relevant projects, and I tried to learn lessons from this observation. This paper presents two case studies.

1. Diagnosis problem solving by model compilation. It follows the motto: "Spend search in model construction rather than in model processing."
2. Design problem solving by functional abstraction. It follows the motto: "Construct a poor solution with little search, which then must be repaired."

On second sight it becomes apparent that the success of both mottos is a consequence of untwining logic-oriented reasoning (in the form of search and deduction) and approximation-oriented reasoning (in the form of simulation).

Keywords: Model Construction, Search, Diagnosis, Design Automation

1 Automating Knowledge-Intensive Tasks

"How can knowledge-intensive tasks such as the diagnosis or the design of complex technical systems be solved using a computer?"

A commonly accepted answer to this question is: "By operationalizing expert knowledge!" And in this sense, the next subsection is a hymn to the simple but powerful, associative models in automated problem solving. Engineers don't search,[1] and computer programs that operationalize engineer (expert) knowledge have been proven successful in various complex problem solving tasks.

[1] Which also means: "Experts don't search", or "need less search" (during problem solving).

W. Lenski (Ed.): Logic versus Approximation, LNCS 3075, pp. 120–137, 2004.

A second, also commonly accepted answer to the above posed question is: "By means of search!" This answer reflects the way of thinking of the modern AI pragmatist, who believes in deep models and the coupling of search and simulation. Deep models, or, models that rely on "first principles" have been considered the worthy successor of the simple associative models [6, 9]; they opened the age of the so-called Second Generation Expert Systems [8, 37]. In this sense, Subsection 1.2 formulates diagnosis and design problems as instances of particular search-plus-simulation problems.

Though the search-plus-simulation paradigm can be identified behind state-of-the-art problem solving methodologies [2, 14], many systems deployed in the real world are realized according to simpler associative paradigms (cf. [4, 5, 24, 28, 32, 34], to mention only a few). As a source for this discrepancy we discover the following connection: The coupling of search and simulation is willingly used to make up for missing problem solving knowledge but, as a "side effect", often leads to intractable problems.

Drawing the conclusion "knowledge over search" is obvious on the one hand, but too simple on the other: Among others, the question remains what can be done if the resource "knowledge" is not available or cannot be elicited, or is too expensive, or must tediously be experienced? Clearly, expert knowledge cannot be cooked up—but we learn from human problem solvers where to spend search effort deliberately in order to gain the maximum impact for automated problem solving. The paper in hand gives two such examples: In Subsection 1.3 we introduce the principles of "model compilation" and "functional abstraction" to address behavior-based diagnosis and design problems. These principles are fairly different and specialized when compared to each other; interestingly, common to both is that they develop from the search-plus-simulation paradigm by untwining the roles of search and simulation. In this way they form a synthesis of the aforementioned paradigms.

The Sections 2 and 3 of this paper outline two case studies from the field of fluidic engineering, which illustrate how the proposed principles are put to work.

1.1 Thesis: Knowledge Is Power[2]

Human problem solving expertise is highly effective but of heuristic nature; moreover, it is hard to elicit but rather easy to process [21]. E. g., a simple but approved formalization of diagnosis knowledge are associative connections:

$$obs_1 \wedge \ldots \wedge obs_k \rightarrow d,$$

where the obs_i and d denote certain observations and a diagnosis respectively. Likewise, successful implementations of design algorithms don´t search in a gigantic space of behavior models but operate in a well defined structure space instead, which is spanned by compositional (left) and taxonomic relations (right):

$$c \rightarrow c_1 \wedge \ldots \wedge c_k \qquad\qquad c \rightarrow c_1 \vee \ldots \vee c_k,$$

where the c_i denote components, i. e., the associations describe a decomposition hierarchy in the form of an And-Or-graph. Another class of design algorithms employ the

[2] This famous phrase is often attributed to Edward A. Feigenbaum, though he did not originate the saying.

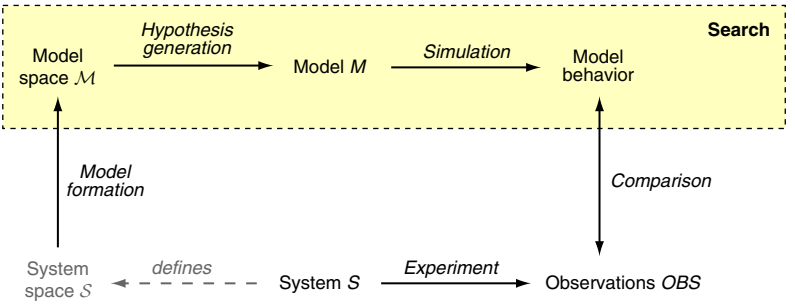

Fig. 1. A generic scheme of model-based diagnosis: Given is the interesting system S, which defines a space $\mathcal{S}$ of faulty systems, and a set of observations OBS. On a computer, $\mathcal{S}$ is represented as a model space, $\mathcal{M}$, wherein a model M^* is searched whose simulation complies with OBS.

case-based reasoning paradigm retrieve-and-adapt, an advancement of the classical AI paradigm generate-and-test [22, 30, 31]:

$$SIM(D_1, D_2) \rightarrow USABILITY(M_1, D_2),$$

which states that the known solution M_1 for a demand set D_1 can be used (adapted) to satisfy a demand set D_2, if D_1 and D_2 are similar.

1.2 Antithesis: Search Does All the Job

Preliminaries. Let S be a system. In accordance with Minsky we call M a model of S, if M can be used to answer questions about S [25]. M may establish a structural, a functional, an associative, or a behavioral model. In this paper the focus is on behavioral models, which give us answers to questions about a system's behavior.

A search problem is characterized by a search space consisting of states and operators. The states are possible complete or partial solutions to the search problem, the operators define the transformation from one state into another. Here, in connection with behavior-based diagnosis and design problems, the search space actually is a model space. It is denoted by $\mathcal{M}$. The model space is only defined implicitly; it comprises all models that could be visited during search.

Diagnosis Problem Solving. Starting point of a diagnosis problem is a system S along with as set of observations OBS. The observations are called symptoms if they do not coincide with the expected behavior of S. Performing diagnosis means to explain symptoms in terms of misbehaving components, that is, to identify a system S^* in a space $\mathcal{S}$ of faulty systems that will exhibit OBS. A model-based diagnosis algorithm performs this search in a model space $\mathcal{M}$, which contains—at the desired level of granularity—models that correspond to faulty systems in $\mathcal{S}$. The objective is to identify a model $M^* \in \mathcal{M}$ whose simulation produces a behavior that complies with OBS. Figure 1 illustrates the connections.

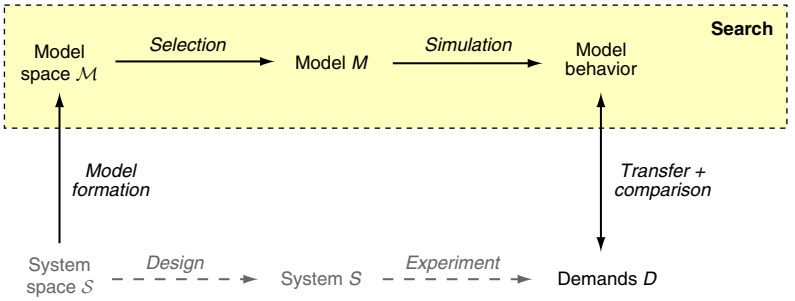

Fig. 2. An generic scheme of design problem solving: Given is a space $\mathcal{S}$ of possible design solutions and a set of demands D. On a computer, $\mathcal{S}$ is represented as a model space, $\mathcal{M}$, wherein a model M^* is searched whose behavior fulfills D.

Several model-based diagnosis approaches, such as the GDE, GDE+, or Sherlock base on such a simulation cycle. Their search in $\mathcal{M}$ is highly informed since they exploit the underlying device topology for hypotheses generation.[3] Adopting the notation of Reiter, model-based diagnosis can be formalized as follows [29]:

$$\alpha_{\Delta} = SD \wedge OBS \wedge \{AB(c) \mid c \in \Delta\} \wedge \{\neg AB(c) \mid c \in COMPS \setminus \Delta\},$$

where SD is a logic-based formulation of $\mathcal{M}$, $COMPS$ denotes a set of symbols that represent the components of the system S, $\Delta \subseteq COMPS$ denotes the broken components, and AB is a special predicate that indicates whether or not a component is working abnormally. Stipulating Occam's razor, a diagnosis algorithm determines a diagnosis Δ as the solution of the following optimization problem:

$$\Delta = \text{argmin}_i\Big(|\Delta| = i \quad \wedge \quad \alpha_{\Delta} \text{ is satisfiable}\Big)$$

Design Problem Solving. Starting point of a design problem is a space $\mathcal{S}$ of possible design solutions along with a set D of demands. Solving a design problem means to determine a system $S^* \in \mathcal{S}$ that fulfills D. Typically, S^* is not found by experimenting in the real world but by operationalizing a virtual search process after having mapped the system space, $\mathcal{S}$, onto a model space, $\mathcal{M}$. It is the job of a design algorithm to efficiently find a model $M^* \in \mathcal{M}$ whose simulation produces a behavior that complies with D and which optimizes a possible goal criterion. Figure 2 illustrates the connections.

Compared to the previous diagnosis scheme, the model space of a design problem is usually orders of magnitude bigger. This is also reflected by the following formalization:

$$\alpha_{CPT} = \underbrace{SD \wedge D \wedge COMPS}_{\text{configuration}} \underbrace{\wedge\, PARAM}_{\text{+ parameterization}} \underbrace{\wedge\, TOP}_{\text{+ structure finding}} \ ,$$

[3] Model-based diagnosis approaches can be further characterized in the way simulation is controlled, fault models are employed, dependencies are recorded, measurement points are chosen, or failure probabilities are utilized [17, 11, 12, 42].

where SD is a logic-based formulation of the behavior of all components in $\mathcal{M}$, $COMPS$ denotes the actually selected components, $PARAMS$ their parameterization, and TOP defines the topology, i. e., how the selected components are connected to each other. The complexity of a design problem depends on the degrees of freedom in the search process: Within a configuration problem merely $COMPS$ is to be determined, whereas in a behavior-based design problem the components need to be parameterized and yet the system structure is to be found such that α_{CPT} is satisfiable.

The outlined diagnosis and design schemes are inviting: Giving a mapping from systems $\mathcal{S}$ to models $\mathcal{M}$—which can be stated straightforwardly in engineering domains—the related analysis or synthesis problem can be solved by the search-plus-simulation paradigm. As already mentioned at the outset, many successful implementations of diagnosis and design systems do not follow this paradigm. They contain an explicit representation of an engineer's problem solving knowledge instead, say, his or her model of expertise. A problem solver that has such knowledge-based models at its disposal spends little effort in search—a fact which makes these models appearing superior to the deep models used in the search-plus-simulation paradigm. On the other hand, several arguments speak for the latter; a compelling one has to do with knowledge acquisition: In many situations it is not feasible for technical or economical reasons to acquire the necessary problem solving knowledge to operationalize tailored models of expertise.[4] Remarkably, de Kleer actually concludes: " Knowledge isn't power. Knowledge is evil." [10].

1.3 A Synthesis: Untwine Search and Simulation

The purpose of this subsection is twofold: It annotates problems of the search-plus-simulation paradigm, and it introduces two advancements of this principle: model compilation and functional abstraction. Here, the former is applied to diagnosis problem solving, while the latter is used to tackle a design problem.[5] Within both principles search as well as simulation still play central roles. However, compared to the search-plus-simulation paradigm, the simulation step is no longer integral part of the search cycle, say, search (logic-oriented reasoning) and simulation (approximation-oriented reasoning) are untwined.

Model Compilation. The search-plus-simulation paradigm in model-based diagnosis enables one to analyze a system for which no diagnosis experience is available or which is operated under new conditions [17]. On the other hand, for complex technical systems model-based diagnosis needs excellent simulation capabilities, because the goal driven reasoning process requires inverse simulation runs (from observations back to causes) to efficiently cover all symptoms [14, 11]. Still more problematic are the following limitations:

[4] Other advantages bound up with this paradigm are: the possibility to explain, to verify, or to document a reasoning process, the possibility to reuse the same models in different contexts, the extendibility to new device topologies, or the independence of human experts.

[5] This correspondence is not obligatory; in [39] a configuration tool of a large telecommunication manufacturer is described, wherein model compilation provides a key technology.

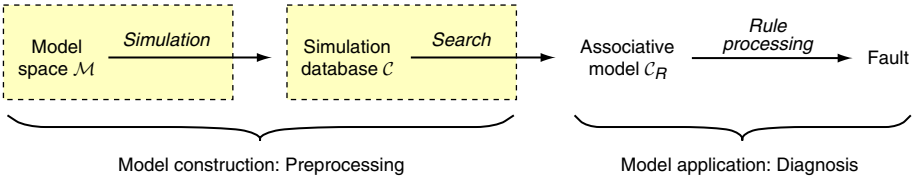

Fig. 3. Model compilation untwines logic-oriented and approximation-oriented reasoning: Simulating the model of a system in various fault modes yields a simulation data base $\mathcal{C}$ from which a rule-based model $\mathcal{C}_R$ is constructed.

1. In domains with continuous quantities the classification of values as symptoms is ambiguous [23].
2. Long interaction paths between variables result in large conflict sets.
3. Many technical systems have a feedback structure; i. e., cause-effect chains, which are the basis for an assumption-based reasoning process, cannot be easily stated.

A promising strategy in this situation is the compilation of an associative model from the given model of first principles, which is achieved as follows: By simulating the model of first principles in various fault modes and over its typical input range a simulation database $\mathcal{C}$ is built up. Within a subsequent search step, a rule-based model $\mathcal{C}_R$ is constructed from $\mathcal{C}$, where long cause-effect chains are replaced with weighted associations and which is optimized for a classification of the fault modes. In this way the simulation and the search step form a preprocessing phase, which is separated from the phase of model application, i. e., the diagnosis phase. Figure 3 illustrates the principle.

Compiled models have a small computational footprint. As well as that, model compilation breaks feedback structures, and, under the assumption that all observations have already been made, an optimum fault isolation strategy can be developed [41]. Section 2 outlines a model compilation application in the fluidic engineering domain.

Functional Abstraction. The search-plus-simulation paradigm has also been suggested as a fundamental problem solving strategy for design tasks.[6] While the role of simulation is like in the diagnosis task above, namely, analyzing a model's behavior, does the reasoning situation raise another difficulty: Applying just search-plus-simulation renders real-world design tasks intractable, because of the mere size of the related model space $\mathcal{M}$. As already indicated, the lack of problem solving knowledge (here in the form of design rules) forces one to resort to the search-plus-simulation paradigm. Again, model compilation could be applied to identify underlying design rules, but, this is tractable only for medium-sized configuration problems [39]. Moreover, the complexity problem, which is caused by the size of $\mathcal{M}$, is not eased but only shifted to a preprocessing phase.

If search cannot be avoided, one should at least ensure that search effort is spent deliberately. In this situation we learn from the problem solving behavior of engineers:

[6] Gero, for example, proposes a cycle that consists of the steps synthesis, analysis, and evaluation. Sinha et al. present a framework to implement simulation-based design processes for mechatronic systems [27, 35].

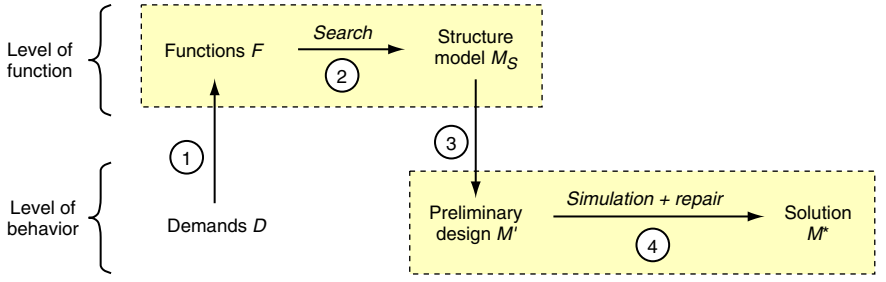

Fig. 4. The paradigm of functional abstraction applied to design problem solving. Observe that logic-oriented reasoning (search) has been decoupled from approximation-oriented reasoning (simulation + repair): The former is used to find a structure model M_S, the latter is used to repair a suboptimum raw design.

1. Engineers solve a design problem rather at the level of function than at the level of behavior, accepting to miss the optimum.
2. Engineers rather adapt a suboptimum solution than trying to develop a solution from scratch, accepting to miss the optimum.
3. Engineers can formulate repair and adaptation knowledge easier than a synthesis theory.

Putting together these observations one obtains the paradigm "Design by Functional Abstraction", which is illustrated in Figure 4.[7] Put it overstated, the paradigm says: At first, we construct a poor solution of a design problem, which then must be repaired.

Key idea of design by functional abstraction is to construct candidate solutions within a very simplified design space, which typically is some structure model space. A candidate solution, M_S, is transformed into a preliminary raw design, M', by *locally* attaching behavior model parts to M_S. The hope is that M' can be repaired with reasonable effort, yielding an acceptable design M^*. Design by functional abstraction makes heuristic simplifications at two places: The original demand set, D, is simplified toward a functional specification F (Step 1), and, M_S is transformed locally into M' (Step 3). Section 3 presents an application of this paradigm.

2 Case Study I. Diagnosis Problem Solving by Model Compilation

The fault detection performance of a diagnosis system depends on the adequateness of the underlying model. Model compilation is one paradigm for constructing adequate models; the model-based diagnosis paradigm, either with or without fault models, provides another. Under the latter, the cycle of simulation and candidate discrimination is executed at runtime, while under the compilation paradigm it is anticipated in a preprocessing phase (see Figures 1 and 3). Reasoning with compiled diagnosis models is similar

[7] The first three steps of this method resemble syntax and semantics (the horseshoe principle) of the problem solving method "Heuristic Classification", which became popular as the diagnosis approach underlying MYCIN [7].

to associative diagnosis; however, the underlying model in an associative system is the result of a substantial model formation process. By contrast, model compilation pursues a data mining strategy and aims at an automatic acquisition of associative knowledge [36].

The idea to derive associative knowledge from deep models has been proposed among others in [37]. Moreover, with respect to fault detection and isolation (FDI), measurement selection, and diagnosability a lot of research has been done. A large part of this work concentrates on dynamic behavior effects, which are not covered here [15, 26]. Nevertheless, since the compilation concept focuses on search space and knowledge identification aspects it can be adapted to existing FDI approaches as well.

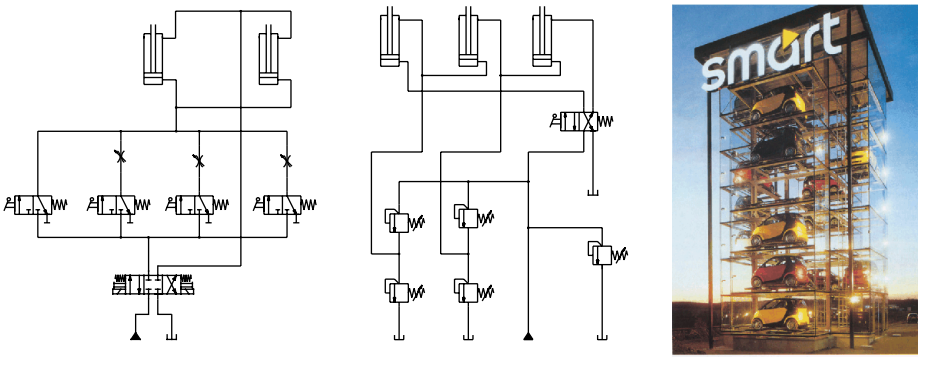

Fig. 5. Diagrams of two medium-sized hydraulic circuits and a photo of the hydraulically operated Smart Tower.

2.1 Hydraulic Systems and Their Components

In this section, as well as in Section 3, the field of hydraulic engineering serves us as application domain. Hydrostatic drives provide advantageous dynamic properties and represent a major driving concept for industrial applications. They consist of several types of hydraulic building blocks: Cylinders, which transform hydraulic energy into mechanical energy, various forms of valves, which control flow and pressure of the hydraulic medium, and service components such as pumps, tanks, and pipes, which provide and distribute the necessary pressure p and flow Q. Figure 5 (left-hand side) shows two medium-sized examples of circuits we are dealing with.

Component Faults and Fault Models. A prerequisite for applying model compilation for diagnosis purposes is that components are defined with respect to both their normal and their faulty behavior. Below, such a fault model is stated exemplary for the check valve. Typical check valve faults include jamming, leaking, or a broken spring. These faults affect the resistance characteristic of the valve in first place (cf. Table 1).

Other fault models relate to slipping cylinders due to interior or exterior leaking, incorrect clearance or sticking throttle valves, directional valves with defect solenoid or

Table 1. Resistance law of a working and a faulty check valve operating in its control range where $\Delta p > p_0$. The deviation coefficient $\varepsilon_{\text{valve}}$ is a state quantity, which is modeled as a continuous random variable.

Normal resistance behavior	Faulty resistance behavior
$R = \dfrac{m^2 \cdot \Delta p}{(\Delta p - p_0)^2}$	$R = \dfrac{m^2 \cdot \Delta p}{(\Delta p - p_0 \cdot (1 + \varepsilon_{\text{valve}}))^2}$

contaminated lands, and pumps showing a decrease in performance. For all fault models, a deviation coefficient ε is modeled as a continuous random variable which defines the distribution of the fault seriousness.

2.2 Construction of a Compiled Model

We construct a compiled model for a system S in five steps. Within the first step a simulation data base $\mathcal{C}$ is built, which is then successively abstracted towards a real-valued symptom data base $\mathcal{C}_\Delta$, a symbolic interval data base $\mathcal{C}_I$, an observer data base $\mathcal{C}_O$, and, finally, a rule set $\mathcal{C}_R$, which represents the heuristic diagnosis model.[8]

Simulation. Behavior models of hydraulic systems are hybrid discrete-event/continuous-time models [3]. The trajectories of the state variables can be considered as piecewise continuous segments, called phases. The discrete state variables such as valve positions, relays, and switches are constant within a phase, and in between the phases one or more of them changes its value, leading to another mode of the system. The continuous variables z_i such as pressures, flows, velocities, or positions are the target of our learning process; they form the set Z. The phase-specific, quasi-stationary values of the variables in Z are in the role of symptoms, since abrupt faults can cause their significant change.

Let S be a system, let D be a set denoting the interesting component faults in S, and let $\mathcal{M}$ be the related space of models. I. e., $\mathcal{M}$ includes the interesting faulty models with respect to D as well as the correct model of S. The result of the simulation step is a data base $\mathcal{C}$, which contains samples of the vector of state trajectories drawn during the simulation of the models in $\mathcal{M}$.

Symptom Identification. For each fault simulation vector in $\mathcal{C}$ the deviations of its state variables to the related faultless simulation vector is computed. The computation is based on a special operator "$\ominus$", which distinguishes between effort variables and flow variables. The former are undirected, and a difference between two values of this type is computed straightforwardly. The latter contain directional information, and their difference computation distinguishes several cases. Result of this step is the symptom data base of $\ominus$-deviations, $\mathcal{C}_\Delta$.

Interval Formation. The symptom vectors in $\mathcal{C}_\Delta$ are generalized by mapping for each $z \in Z$ the deviations $\delta_z^{(1)}, \ldots, \delta_z^{(|\mathcal{C}|)}$, $\delta_z^{(i)} \in \mathbf{R}$, onto p intervals $\mathbf{I}_z^{(1)}, \ldots, \mathbf{I}_z^{(p)}$, $\mathbf{I}_z^{(j)} \subset \mathbf{R}$, with $\bigcup_j \mathbf{I}_z^{(j)} = \mathbf{R}$. This is an optimization task where, on the one hand, the loss of

[8] A detailed description of the compilation procedure can be found in [40].

discrimination information is to be kept minimum (the larger p the better), while on the other hand, constraints of measuring devices are to be obeyed (the smaller p the better). Observe that in this abstraction step the domain of real numbers is replaced by a propositional-logical representation: For each state variable $z \in Z$ a new domain I_z of interval names is introduced, which map in a one-to-one manner onto the real-valued intervals. The symbolic interval database that develops from $\mathcal{C}_\Delta$ by interval formation is denoted by $\mathcal{C}_I$.

Measurement Selection. By means of simulation, values are computed for all variables in Z. In fact, restricted to a handful of measuring devices or sensors, only a small subset $O \subset Z$ can be observed at the system. Measurement selection means to determine the most informative variables in Z—or, speaking technically, to place a set of $|O|$ observers such that as many faults as possible can be classified. O is determined by analyzing for each phase and for each variable $z \in Z$ the correlations between the symbolic intervals I_z and the set of component faults D. Our analysis generalizes the idea of hypothetical measurements, which goes back on the work of Forbus and de Kleer. Let $O \subset Z$ be the set of selected observers according to this analysis; the database that emerges from the symbolic interval database $\mathcal{C}_I$ by eliminating all variables in $Z \setminus O$ is called observer database $\mathcal{C}_O$; it is much smaller than $\mathcal{C}_I$. However, its number of elements is unchanged, i. e., $|\mathcal{C}_O| = |\mathcal{C}|$.

Rule Generation. Within the rule generation step reliable diagnosis rules are extracted from $\mathcal{C}_O$. The rules have a propositional-logical semantics and are of the form

$$\mathbf{r} \;=\; \iota_1 \wedge \ldots \wedge \iota_k \rightarrow d,$$

where the ι_i denote interval names and d denotes a diagnosis. The semantics of $\mathbf{r}$ is defined by means of two truth assignment functions, $\alpha : \bigcup I_z \rightarrow \{0,1\}$ and $\beta : D \rightarrow \{0,1\}$. If $\mathbf{I}$ is the real-valued interval associated with the interval name ι and if δ is a symptom observed at S, then α and β are defined as follows:

$$\alpha(\iota) = \begin{cases} 1 \Leftrightarrow\ \delta \in \mathbf{I} \\ 0 \text{ otherwise.} \end{cases} \qquad \beta(d) = \begin{cases} 1 \Leftrightarrow \text{ the fault in } S \text{ is } d. \\ 0 \text{ otherwise.} \end{cases}$$

Note that the inference direction of these rules is reverse to the cause-effect computations when simulating a behavior model: We now ask for symptoms and deduce faults, and—as opposed to the simulation situation—this inference process must not be definite. To cope with this form of uncertainty each rule $\mathbf{r}$ is characterized by (1) its confidence $c(\mathbf{r})$, which rates the logical quality of the implication, and (2) its relative frequency, called "support" in the association rule jargon [1]. The rule generation step is realized with data mining methods and yields the desired rule model $\mathcal{C}_R$.

2.3 Experimental Analysis: Diagnosis with DÉJÀVU

Diagnosing the system S means to process the rule model $\mathcal{C}_R$ subject to the observed symptoms. For the operational semantics of the rules' confidence and support values we employ the formula below, which computes for each fault $d \in D$ its confidence in

"$\beta(d) = 1$", given a rule model $\mathcal{C}_R$ and a truth assignment α. The formula combines the highest achieved confidence with the average confidence of all matching rules:

$$c(\text{“}\beta(d) = 1\text{”}) = c(\mathbf{r}^*) + \big(1 - c(\mathbf{r}^*)\big) \cdot \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} c(\mathbf{r}),$$

where $\mathcal{R} \subset \mathcal{C}_R$ denotes the matching rules with conclusion d, and $\mathbf{r}^* \in \mathcal{R}$ denotes a rule of maximum confidence.

The outlined model construction process as well as the rule inference have been operationalized within the diagnosis program DÉJÀVU [20]. For simulation purposes, DÉJÀVU employs the FLUIDSIM simulation engine. The approach has been applied to several medium-sized hydraulic circuits (about 20-40 components) with promising results. The table in Figure 6 shows the diagnosis performance depending on the number of observers $|O|$; basis were more than 2000 variations of $|D| \approx 15$ different component faults. The results were achieved with automatically constructed rule models $\mathcal{C}_R$ that have not been manually revised. The right-hand side of Figure 6 shows the average number of rules in $\mathcal{C}_R$ as a function of $|O|$.

Diagnosis accuracy	Number of observers $\|O\|$					
	1	2	3	4	5	6
"exact"	0.40	0.55	0.53	0.52	0.52	0.53
"1 in 3"	0.51	0.72	0.81	0.88	0.92	0.96

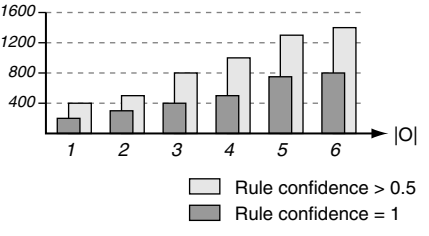

Fig. 6. The table shows the fraction of classified faults depending on the observer number $|O|$; "exact" stands for a unique fault prediction, "1 in 3" indicates a multiple prediction of two or three components including the faulty one. The bar graph on the right shows the number of generated rules, $|\mathcal{C}_R|$, as a function of $|O|$. Dark bars indicate rules with a confidence value of 1, light bars stand for confidence values greater than 0.5.

3 Case Study II. Design Problem Solving by Functional Abstraction

Even for an experienced engineer, the design of a fluidic system is a complex and time-consuming task, that, at the moment, cannot be automated completely. The effort for acquiring the necessary design knowledge exceeds by far the expected payback, and, moreover, the synthesis search space is extremely large and hardly to control—despite the use of knowledge-based techniques.

Two possibilities to counter this situations are "competence partitioning" and "expert critiquing". The idea of competence partitioning is to separate the creative parts of a design process from the routine jobs, and to provide a high level of automation regarding the latter [38]. Expert critiquing, on the other hand, employs expert system technology to assist the human expert rather than to automate a design problem in its entirety [18, 13].

In this respect, design by functional abstraction can be regarded as a particular expert critiquing representative.

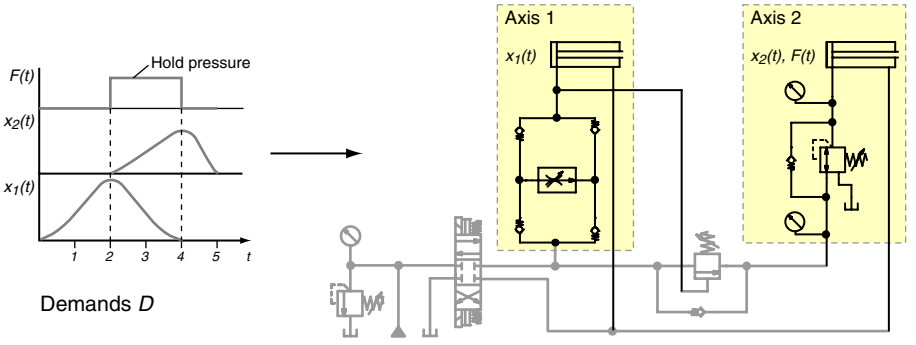

Fig. 7. Hydraulic design means to translate a demand description (left) to a circuit model.

3.1 Design in Fluidic Engineering

Taken the view of configuration, the designer of a fluidic system selects, parameterizes, and connects components like pumps, valves, and cylinders such that the demands D are fulfilled by the emerging circuit.[9] Solving a fluidic design problem at the component level is pretty hopeless. The idea is to perform a configuration process at the level of functions instead, which in turn requires that fluidic functions possess constructional equivalents that can be treated in a building-block-manner. In the fluidic engineering domain this requirement is fairly good fulfilled; the respective building blocks are called "fluidic axes".

Figure 7 shows a demand description D (left) and a design solution M^* in the form of a circuit diagram that fulfills D. The circuit consists of two hydraulic axes that are coupled by a sequential coupling. To automate this design process, so to speak, to automate the mapping $D \longrightarrow M^*$, we apply the paradigm of functional abstraction (cf. Figure 8 and recall Figure 4):

1. The demand specification, D, is abstracted towards a functional specification, F.
2. At this functional level a structure model M_S according to the coupling of the fluidic functions in F is generated.
3. M_S is completed towards a tentative behavior model M' by plugging together locally optimized fluidic axes; here, this step is realized by case-based reasoning.
4. The tentative behavior model M' is repaired, adapted, and optimized globally.

The following subsection describes the basic elements of this design approach, i. e., Step 2, 3, and 4.

[9] The concepts presented in section have been verified in the hydraulic domain in first place; however, they can be applied to the pneumatic domain in a similar way, suggesting us to use preferably the more generic word "fluidic". Again, a detailed description of the approach can be found in [40].

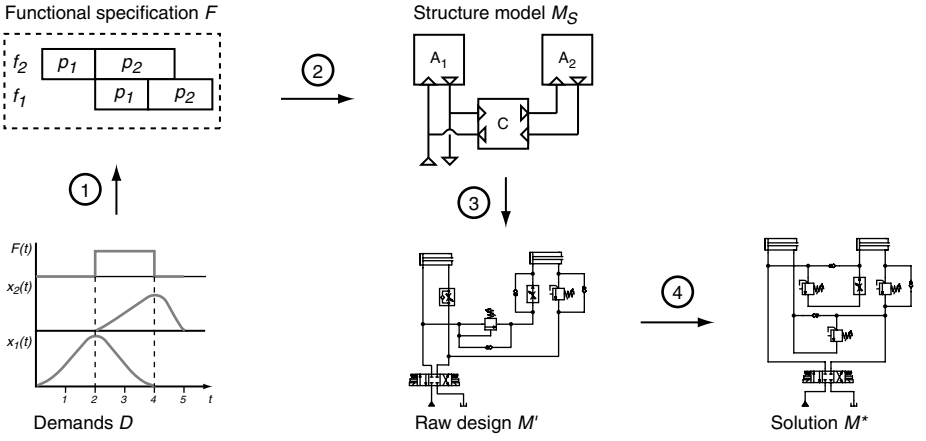

Fig. 8. The functional abstraction paradigm applied to fluidic circuit design.

Remarks. A human designer is capable of working at the component level, *implicitly* creating and combining fluidic axes towards an entire system. His ability to automatically derive function from structure—and vice versa: structure *for* function—allows him to construct a fluidic system without the idea of high-level building blocks.

3.2 Elements of the Design Procedure

Design by functional abstraction rigorously simplifies the underlying domain theory. Here, the tacit assumptions are as follows: (a) Each set of demands, D, can be translated into a set of fluidic functions, F, (b) each function $f \in F$ can be mapped one to one onto a fluidic axis A that operationalizes f, (c) D can be realized by coupling the respective axes for the functions in F, whereas the necessary coupling information can be derived from D.

While the first point goes in accordance with reality, the Points (b) and (c) imply that a function f is neither realized by a combination of several axes nor by constructional side effects. This assumption establishes a significant simplification.

Topology Generation. If the synthesis of fluidic systems is performed at the level of function, the size of the synthesis space is drastically reduced. To be specific, we allow only structure models that can be realized by a recursive application of the three coupling rules shown in Figure 9. The search within this synthesis space is operationalized by means of a design graph grammar [40], which generates reasonable topologies with respect to the functional specification F. The result of this step is a structure model M_S; M_S defines a graph whose nodes correspond to fluidic functions and coupling types.

Case-Based Design of Axes. A structure model M_S is completed towards a behavior model by individually mapping its nodes onto appropriate subcircuits that represent fluidic axes or coupling networks. Figure 10 shows five subcircuits each of which representing a certain fluidic axis. It turned out that the mapping of a fluidic function f onto

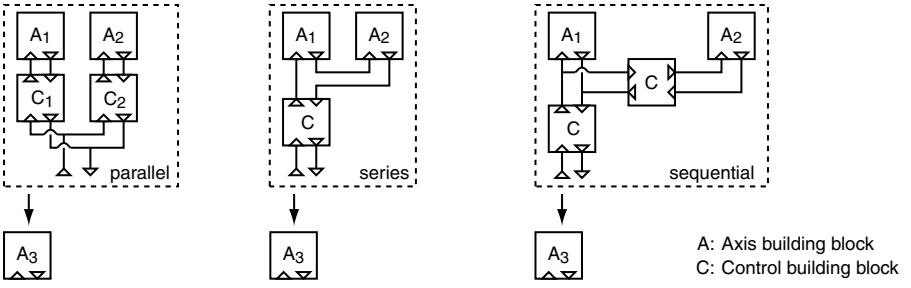

Fig. 9. The three allowed coupling types to realize a circuit's topology.

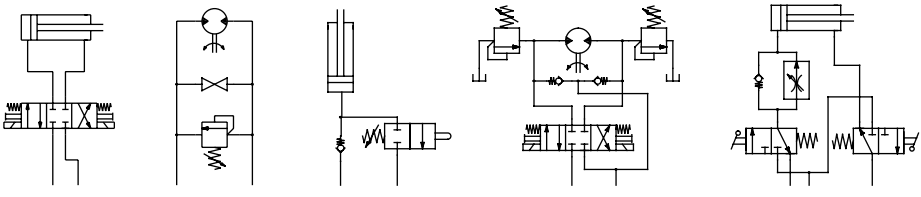

Fig. 10. Five fluidic (hydraulic) axes for different functions and of different complexity.

an axis can be accomplished ideally with case-based reasoning: Domain expert were able to compile a case base $\mathcal{C}$ of about seventy axes that can be used to cover a wide spectrum of fluidic functions.

Speaking technically, an axis is characterized by the unit tasks it can carry out, such as "hold pressure", "fast drive", "hold position", etc. In order to valuate the similarity of fluidic axes and functions, a measure φ was constructed that compares two sequences of unit tasks respecting their types, order, distances, forces, and precision. Moreover, since the axes that are retrieved from $\mathcal{C}$ must be adapted to fulfill a desired f, we also developed a case adaptation scheme that operationalizes engineering know-how in the form of scaling rules. The result of this step, i. e., the composition of adapted fluidic axes according to M_S, yields a preliminary design solution, the raw design M'.

A Design Language for Repair. There is a good chance that a raw design M' has the potential to fulfill D, say, that a sequence of repair steps can be found to transform M' into a behavior model M^*. An example for such a repair measure is the following piece of design knowledge:

"An insufficient damping can be improved by installing a by-pass throttle."

The measure encodes a lot of implicit engineering know-how, among others: (a) A by-pass throttle is connected in parallel, (b) the component to which it is connected is a cylinder, (c) a by-pass throttle is a valve. What is more, the above repair measure can be applied to different contexts in a variety of circuits. To operationalize such kind of knowledge, we developed a prototypic scripting language for fluidic circuit design.[10] In this place we will not delve into language details but refer to [33].

[10] The research was part of the OFT-project, which was supported by DFG grants KL 529/7-1,2,3 and SCHW 120/56-1,2,3.

3.3 Experimental Analysis: Design Automation with ARTDECO-CBD

The outlined concepts have been embedded within the design assistant ARTDECO-CBD, which is linked to a drawing and simulation environment for fluidic systems [19]. The design assistant enables a user to formulate his design requirements as a set of fluidic functions F. For an $f \in F$ a sequence of unit tasks can be defined, where several characteristic parameters such as duration, precision, or maximum values can be stated for each unit task.

Clearly, the crucial question is "How good are the designs of ARTDECO-CBD?" A direct evaluation of the generated models is restricted: an absolute measure that captures the design quality does not exist, and the number of properties that characterizes a design is large and hardly to quantify. On the other hand, the quality of a generated design can be rated *indirectly*, by measuring its "distance" to a design solution created by a human expert. In this connection, the term distance stands for the real modification effort that is necessary to transform the computer solution into the human solution. The experimental results presented in Table 2 report on such a competition.-[11]

Table 2. Runtime and quality results of automatically generated designs. The column "O.K." shows the portion of designs whose simulation fulfills D; only solutions with high similarity values ($\varphi > 0.9$) were considered. The column "Quality" shows the expert evaluation: (+), (o), and (–) indicate a small, an acceptable, and a large modification effort to transform the machine solution into a solution accepted by the human expert.

Number of axes	Time for retrieval	Time for reuse	O.K. (at $\varphi > 0.9$)	Quality		
1	$\ll$ 1s	0.10s	80%	60% (+)	35% (o)	5% (–)
2	$\ll$ 1s	0.63s	75%	50% (+)	45% (o)	5% (–)
3	$\ll$ 1s	0.91s	70%	40% (+)	50% (o)	10% (–)
4	$\ll$ 1s	1.43s	60%	20% (+)	65% (o)	15% (–)
5	$\ll$ 1s	2.00s	20%	5% (+)	80% (o)	15% (–)

Conclusion

The success of a diagnosis or design approach depends on the underlying model space—say, its size, and the way it is explored. A tractable model space is in first place the result of adequate models, which in turn are the result of a skillful selection, combination, and customization of existing construction principles. Engineers don't search because they use adequate models. The challenge is to operationalize such models on a computer.

Especially when expert knowledge is not at hand, the combination of deep models, simulation, and search is inviting because it promises high fidelity. On the other hand, applying a search-plus-simulation paradigm entails the risk to fail completely because of various reasons. This is not inevitable: The principles of model compilation and functional abstraction exemplify how search and simulation can be combined to realize problem solving strategies for complex diagnosis and design problems.

[11] Test environment was a Pentium III system at 450 MHz with 128 MB main memory.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington D. C., May 1993. ACM Press.
2. Erik K. Antonsson and Jonathan Cagan. *Formal Engineering Design Synthesis*. Cambridge University Press, 2001. ISBN 0-521-79247-9.
3. Paul I. Barton. Modeling, Simulation, and Sensitivity Analysis of Hybrid Systems. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, pages 117–122, Anchorage, Alaska, September 2000. ACM Press.
4. David C. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.
5. B. Buchanan and E. Shortliffe. *Rule-Based Expert Systems. The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Massachusetts, 1984.
6. B. Chandrasekaran and Sanjay Mittal. Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving. In David Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence*, pages 349–354, Pittsburgh, PA, August 1982. AAAI Press. ISBN 0-86576-043-8.
7. William J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.
8. Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors. *Second Generation Expert Systems*. Springer, 1992. ISBN 0-387-56192-7.
9. Randall Davis. Expert Systems: Where Are We? And Where Do We Go from Here? *AI Magazine*, 3(2):3–22, 1982.
10. Johan de Kleer. AI Approaches to Troubleshooting. In *Proceedings of the International Conference on Artificial Intelligence in Maintenance*, pages 78–89. Noyes Publications, 1985.
11. Johan de Kleer and Brian C. Williams. Diagnosing Multiple Faults. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 372–388. Morgan Kaufman, 1987.
12. Johan de Kleer and Brian C. Williams. Diagnosis with Behavioral Models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1324–1330, Detroit, Michigan, 1989.
13. Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, and Gerry Stahl. Embedding Critics in Design Environments. *The Knowledge Engineering Review*, 8(4):285–307, December 1993.
14. Kenneth D. Forbus and Johan de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, 1993. ISBN 0-262-06157-0.
15. Paul Frank. Fault diagnosis: A survey and some new results. *Automatica: IFAC Journal*, 26 (3):459–474, 1990.
16. John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.
17. W. Hamscher, L. Console, and Johan de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, 1992.
18. Sture Hägglund. Introducing Expert Critiquing Systems. *The Knowledge Engineering Review*, 8(4):281–284, December 1993.
19. Marcus Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1999.
20. Uwe Husemeyer. *Heuristische Diagnose mit Assoziationsregeln*. Dissertation (to appear), University of Paderborn, Department of Mathematics and Computer Science, 2001.

21. Werner Karbach and Marc Linster. *Wissensakquisition für Expertensysteme*. Carl Hanser Verlag, 1990. ISBN 3-446-15979-7.
22. David B. Leake. Case-Based Reasoning: Issues, Methods, and Technology, 1995.
23. Eric-Jan Manders, Gautam Biswas, Pieter J. Mosterman, Lee A. Barford, and Robert Joel Barnett. Signal Interpretation for Monitoring and Diagnosis, A Cooling System Testbed. In *IEEE Transactions on Instrumentation and Measurement*, volume 49:3, pages 503–508, 2000.
24. John McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*, 19:39–88, 1982.
25. Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.
26. Sriram Narasimhan, Pieter J. Mosterman, and Gautam Biswas. A Systematic Analysis of Measurement Selection Algorithms for Fault Isolation in Dynamic Systems. In *Proc. of the International Workshop on Diagnosis Principles*, pages 94–101, Cape Cod, MA, 1998.
27. Christian J. J. Paredis, A. Diaz-Calderon, Rajarishi Sinha, and Pradeep K. Khosla. Composable Models for Simulation-Based Design. In *Engineering with Computers*, volume 17:2, pages 112–128. Springer, July 2001.
28. Frank Puppe. *Systematic Introduction to Expert Systems, Knowledge Representations and Problem-Solving Methods*. Springer, 1993.
29. Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1): 57–95, April 1987.
30. Michael M. Richter. The Knowledge Contained in Similarity Measures, October 1995. Some remarks on the invited talk given at ICCBR'95 in Sesimbra, Portugal.
31. Michel M. Richter. Introduction to CBR. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Weß, editors, *Case-Based Reasoning Technology. From Foundations to Applications*, Lecture Notes in Artificial Intelligence 1400, pages 1–15. Berlin: Springer-Verlag, 1998.
32. Michael D. Rychener. *Expert Systems for Engineering Design*. Academic Press, 1988. ISBN 0-12-605110-0.
33. Thomas Schlotmann. Formulierung und Verarbeitung von Ingenieurwissen zur Verbesserung hydraulischer Systeme. Diploma thesis, University of Paderborn, Institute of Computer Science, 1998.
34. L. C. Schmidt and J. Cagan. Configuration Design: An Integrated Approach Using Grammars. *ASME Journal of Mechanical Design*, 120(1):2–9, 1998.
35. Rajarishi Sinha, Christian J. J. Paredis, and Pradeep K. Khosla. Behavioral Model Composition in Simulation-Based Design. In *Proceedings of the 35th Annual Simulation Symposium*, pages 309–315, San Diego, California, April 2002.
36. R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In H. V. Jagadish and I. S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Canada, June 1996. ACM Press.
37. Luc Steels. Components of Expertise. *AI Magazine*, 11(2):28–49, 1990.
38. Benno Stein. *Functional Models in Configuration Systems*. Dissertation, University of Paderborn, Institute of Computer Science, 1995.
39. Benno Stein. Generating Heuristics to Control Configuration Processes. *Applied Intelligence, APIN-IEA, Kluwer*, 10(2/3):247–255, March 1999.
40. Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation thesis, University of Paderborn, Institute of Computer Science, 2001.

41. Benno Stein. Model Compilation and Diagnosability of Technical Systems. In *Proceedings of the third IASTED International Conference on Artificial Intelligence and Applications (AIA 03)*, Benalmádena, Spain, 2003.
42. Peter Struß and Oskar Dressler. "Physical Negation"—Integrating Fault Models into the General Diagnostic Engine. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 89)*, volume 2, pages 1318–1323, Detroit, Michigan, USA, 1989.