

Integrating model-based diagnosis techniques into current work processes – three case studies from the INDIA project

Heiko Milde^{a,*}, Thomas Guckenbiehl^b,
Andreas Malik^{c,1}, Bernd Neumann^a
and Peter Struss^d

^a *Laboratory for Artificial Intelligence, University of Hamburg, Vogt-Koelln-Str. 30, 22527 Hamburg, Germany*

E-mail: {milde, neumann}@informatik.

uni-hamburg.de

^b *Fraunhofer-Institut IITB, Fraunhoferstr. 1, 76131 Karlsruhe, Germany*

E-mail: guc@iitb.fhg.de

^c *ESG Elektroniksystem- und Logistik-GmbH, Germany*

E-mail: malik@esg-gmbh.de

^d *Technical University of Munich, Department of Computer Science, Orleansstr. 34, 81667 Munich, Germany*

E-mail: struss@in.tum.de

Although the area of model-based diagnosis has developed a number of prototypes with impressive features that promised economic impact and, hence, caught industrial interest, the number of actual industrial applications is still close to zero. One of the reasons is that the successful techniques have not yet been turned into tools that reflect and support the current diagnostic work processes and their existing tools. The INDIA project joined eight German partners (research groups, software suppliers, and end users) in an attempt to take a major step in the transfer of model-based diagnosis techniques into industrial applications. This paper describes part of the work carried out in this project. Rather than presenting the theoretical foundations of the techniques in depth, we focus on the aspect of how model-based diagnostic techniques can be related to established tools and systems in order to provide some leverage for today's work processes and to change them gradually, as opposed to postulating a radical change in current practice and organizational structures.

*Corresponding author.

¹Also affiliated with Robert Bosch GmbH during the project.

From this perspective, we discuss the utilization of model-based techniques for the generation of fault trees for on-line testing and diagnosis of fork lifters, generation of test plans for an intelligent authoring system for car diagnosis manuals, and the exploitation of existing state-chart process descriptions for post-mortem diagnosis of processes in a dyeing plant.

1. Introduction

Research on model-based diagnosis has claimed for some time that its methods promise applications with attractive problem-solving capabilities and significant economical advantages. Reviewing the attractive features of model-based diagnosis, the main benefits are connected with the compositionality and transparency of the model, from which diagnosis knowledge can be generated. Compositionality bears the potential for re-using components, building component libraries and inheritance hierarchies alleviating version control and easing modifications. The transparency of component-based behavior descriptions may add further benefits, including complexity management, exploitation of information from the design phase, and a large degree of compatibility with other life-cycle product data including documentation. Hence, by exploiting modeling techniques important benefits can be gained from model-based diagnosis technology.

In fact, the field has not only made these claims, but has recently provided some evidence for their correctness by producing prototypes for significant application areas [5,26]. Despite all this, transfer of the technology encounters a number of difficulties and is not progressing as fast as expected or hoped for. There are a number of reasons for this experience. One of them is certainly the fact that applying model-based techniques requires significant initial efforts, mainly for producing the appropriate models. Furthermore, in order to

promote the application of the technology, one has to consider the current practice and role of diagnosis in industry.

First of all, many producers of technical systems provide only limited diagnosis support to their products. There are only few large market segments where producers develop sophisticated diagnosis support (e.g., the automotive and aircraft industry). This is changing, however, as devices grow more complex, the cost of maintenance personnel becomes more important and improved service is required to increase customer satisfaction and to remain competitive. Often, there is even enough computational power on-board the devices (such as cars, photo copiers, and even simpler consumer products) or easily available to human trouble shooters. Also expert knowledge exists, as principled knowledge about the devices in the heads of designers and developers and as diagnostic expertise of maintenance personal. What is needed, is computer support for the effective and efficient generation of diagnostics, be it for automated on-board or workshop diagnosis, training or instructions for technicians, etc. This is exactly what model-based diagnosis is claiming to offer. However, to really provide some leverage, its techniques have to deliver effective and efficient support to the **current** work processes!

So far, many prototype systems have demonstrated that there are principled solutions to the diagnostic problems in certain industrial areas. However, most of them remained isolated programs that do not reflect and integrate into the actual work processes and the tools and techniques applied. Now, it is time to address these issues.

To present examples, there exist written diagnosis instructions and authoring tools to create and maintain them, and many of today's computer tools capture diagnostic knowledge in the form of decision trees or fault trees. These techniques have been developed from a maintenance rather than from a design perspective. Traditionally, diagnosis matters are a concern only to the service division of a company, not to the design division. However, as technical systems become larger and more complicated, the design of decision trees becomes more demanding and problems arise. Furthermore, frequent product changes cause excessive costs for the maintenance of such diagnosis equipment. Hence there is growing awareness of the need for re-usability of diagnostic information. Diagnosis equipment should be developed in close connection with design, and construction data and FMEA data should be exploited. This illustrates that, ultimately, model-based

techniques will lead to significant changes in the work processes. Indeed, representing and sharing engineering knowledge in computer-based model libraries will result in a major change, and many benefits will stem exactly from such changes.

However, industry usually prefers to perform such changes in small steps. The introduction of model-based reasoning for complete automation of diagnosis is often perceived as too different from the traditional ways of doing diagnosis. Existing know-how would become worthless and new know-how would have to be acquired. As noted above, the organizational structure would be affected, with diagnosis tasks shifting from the service to the design division.

Any realistic strategy for the deployment of the technology has to be aware of how it can be related to existing knowledge and organization of diagnosis tasks and design solutions that complement existing tools in a seamless fashion. Steps in this direction were undertaken in the German joint research project INDIA (Intelligent Diagnosis in Industrial Applications). In this project, three teams,¹ each consisting of a research institute, a software supplier, and an industrial production company, joined to apply model-based diagnosis technology to real-life diagnosis problems and pave the way for successful applications elsewhere. The particular diagnosis problems provided by the industrial partners represented an interesting subset of industrial diagnosis applications. One application area is diagnosis of automotive equipment, another one is maintenance support for transport vehicles (forklifts), the third one deals with operator assistance in post mortem diagnosis of machinery in a dye house.

Although in INDIA also solutions to automated diagnosis were developed, this paper focuses on work that attempted to exploit present forms of representing and exploiting diagnostic knowledge and produce model-based solutions as add-ons to existing technologies. The next section presents a method for automatically generating fault-tree diagnosis systems for on-line diagnosis of forklifts from design data. The key idea is to use modeling techniques of model-based diagnosis for an exhaustive computation of faulty behavior. Based on these data, fault trees can be gener-

¹Robert Bosch GmbH, STILL GmbH, THEN GmbH, ServiceXpert GmbH, R.O.S.E. Informatik GmbH, Artificial Intelligence Laboratory of the University of Hamburg, Fraunhofer Institute for Information Technology and Data Processing, Model-based Systems and Qualitative Reasoning Group of the Technical University of Munich.

ated automatically and edited preserving correctness and completeness properties.

Section 3 discusses work on a new generation authoring system for diagnosis manuals for car subsystems. While maintaining a text-oriented view on the manuals, it provides an explicit representation of the underlying knowledge. This allows to import plans for diagnosis, testing, and repair which are generated automatically by a model-based system.

The third case study is on post mortem diagnosis of the chemical distributor (CHD), a system to dose and distribute chemicals and colors in a dye house automatically. The solution is guided by the goal of linking to existing engineering knowledge and to re-use by importing state charts from other engineering tools into the diagnostic system.

2. Generating fault trees

2.1. Application scenario and challenges

More than 100.000 forklifts made by the German company STILL GmbH Hamburg are in daily use all over Europe. In order to reduce forklift downtimes, approximately 1100 STILL service workshop trucks utilize fault tree-based computer diagnosis systems for workshop diagnosis. In case of a malfunction a diagnosis system is attached to a forklift performing automated testing. The diagnosis system also instructs service technicians to carry out manual tests. Test sequences are specified by fault trees which are diagnostic decision trees allowing fault identification.

Due to the complexity of the electrical circuits employed in forklifts, fault trees may consist of more than 5000 nodes. When forklift model ranges are modified or new model ranges are released, fault trees are manually generated or adapted by service engineers who apply detailed expert knowledge concerning faults and their effects. Obviously, this practice is costly and quality management is difficult. Furthermore, fault trees are not optimized and fault identification cost is unnecessarily high. Hence, there is a need for computer methods to systematically support the design, modification, and optimization of fault trees. The introduction of new diagnosis techniques, however, raises challenges.

Dealing with this application scenario, the Artificial Intelligence Laboratory of the University of Hamburg considers model-based techniques to be advantageous because they provide a systematic way for design, modification and optimization of diagnosis equipment.

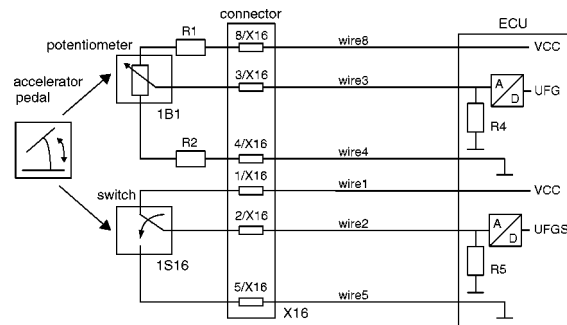


Fig. 1. Wiring diagram of a forklift accelerator pedal circuit.

Since for STILL, completely replacing fault trees is not an immediate option, automatically generating fault trees from device models is promising provided that the complexity of the device modeling process is minimized for economical benefit. Thus, extensive re-use of diagnosis models and fault trees as well as integration of available resources such as design data, service expertise, and expert knowledge from the design process is essential.

In our application, model-based approaches have to deal with electrical circuits of the transport vehicle domain. As an example Fig. 1 shows the wiring diagram of a STILL forklift accelerator pedal circuit which enables an electronic control unit (ECU) to measure the accelerator pedal position. The pedal determines a potentiometer position which controls the value of voltage UFG. In addition, the pedal is connected to a switch controlling voltage UFGS. The ECU provides a supply voltage VCC and it measures UFG and UFGS. Thus, the pedal position is supplied redundantly to secure reliability of the measurements. Beside the quantitative value of UFG, the ECU provides two error flags UFG_HIGH and UFG_LOW with values *OK* and *NOT_OK* which are utilized for fault identification. These flags indicate that UFG exceeds or falls below certain threshold values. The quantitative value of UFGS is also automatically mapped to values *OPEN* and *CLOSED*. In addition to these automated measurements, for fault identification, service technicians can manually measure the voltage drop from wire 8 at connector X16 to ground.

Fault trees of the current diagnosis system allow to identify the following faults: Wires located between the ECU and connector X16 may break due to mechanical stress. The mechanical connection between the accelerator pedal and the switch 1S16 may also break. In this case, the switch is independent from the actual pedal position and it connects either wire 1 and wire 2

or wire 2 and wire 5. The voltage VCC may be supplied incorrectly, i.e., the voltage is not between 9.8 V and 10.1 V. Additionally, the mechanical connection between accelerator pedal and potentiometer 1B1, may not be adjusted correctly or may even be broken.

This example demonstrates that, in our application, electrical circuits usually consist of components that show a variety of different behavior types, such as analog, digital, static, dynamic, linear, nonlinear and software-controlled behavior. Faults may slightly modify component behavior or may even change circuit structures. Hence, a wide range of different symptoms such as slight deviations of parameter values and total loss of functionality may occur. In principle, model-based techniques provide a systematic way for predicting the behavior of electrical circuits, including faulty behavior. However, in our application, adequate modeling of heterogeneous circuits is essential to secure high quality of generated fault trees. Thus, accurate device modeling was a major focus in our work. The following subsections report on the progress which has been achieved.

2.2. Model-based fault tree generation

In our application, nodes of fault trees represent fault sets. Edges are labeled by the tests (involving measurements, observations, display values and error codes) which must be carried out to verify the corresponding child node. Although the basic concepts of model-based fault tree generation are already described in [5,11], for the reader's convenience, we briefly outline the main ideas of the approach in the following.

The first step to model-based fault tree generation is to model a device. This step is supported by component libraries and a device model archive (see Fig. 2). Design data and knowledge from the design process (knowledge concerning intended device behavior, expected faults, available measurements) are integrated into the device modeling process. In a second step, correct and faulty device behavior is predicted automati-

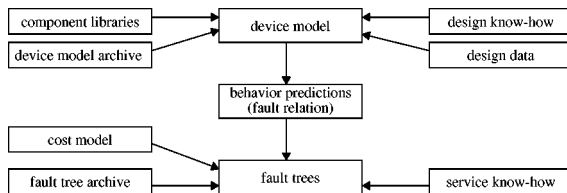


Fig. 2. Basic concepts of model-based fault tree generation.

cally from the device model. Behavior predictions are stored in the so-called fault relation. The third step is to build fault trees from the fault relation. This step is supported by a fault tree archive and a cost model for the tests which can be performed. Fault tree generation can be performed automatically or guided by service know-how, i.e., knowledge concerning cost of measurements and preferable fault tree topologies. In order to realize these concepts, we implemented the MAD system (Modeling, Analyzing, and Diagnosing) that is described in the following.

2.3. Device modeling and behavior prediction

For model-based fault tree generation, behavior predictions are computed taking fault models into account which represent faulty component behavior. Each fault model of the device model is explicitly represented in generated fault trees. Thus, computation of behavior predictions is uncomplex and fault trees are clear and manageable if the device model shows a small number of fault models. Since qualitative models rather represent fundamental system behavior than sharp quantitative parameter values, a single qualitative fault model can cover a wide range of faulty component behavior and, thus, a limited number of fault models can suffice for extensive faulty behavior representation. Therefore, in principle, qualitative circuit modeling is useful for model-based fault tree generation. Moreover, in our application, qualitative electrical device models are adequate because, in STILL fault trees, component faults and symptoms are described qualitatively. Additionally, qualitative models are advantageous because, in principle, dealing with product variants is possible. In the following, MAD's qualitative network analysis is presented. As known from electrical engineering, MAD represents electrical circuits by equivalent networks consisting of standard component models.

2.3.1. Standard component models

Standard component models show no internal structure but they show well-defined and idealized behavior. MAD provides four different standard component models: idealized voltage sources, consumers, conductors and insulators. The behavior of idealized voltage sources is well-known from electrical engineering. Consumers are passive and their current/voltage characteristic is monotonic. Idealized conductors do not allow any voltage drop while idealized insulators do not allow any current. Controlled versions of these standard component models exist. Standard compo-

ment models can be connected in combinations of series, parallel, star and delta (triangular) groupings. This simple internal representation of electrical circuits is sufficient for the following reasons.

- In STILL service workshops, only steady-state diagnosis of electrical circuits is performed. Therefore, only steady-state behavior of physical components has to be represented in component models. In particular, an explicit representation of temporal dependencies is not necessary.
- A small number of qualitative standard component models suffices, because, often, different physical components show similar electrical behavior, i.e., their current/voltage characteristics differ only slightly. Qualitative versions of these current/voltage characteristics are frequently identical.
- MAD's standard component models are deliberately selected so that important behavior classes of the application domain can be adequately represented.

Due to analogies between electrics, mechanics and hydraulics, MAD's internal circuit models are, in principle, also adequate for other technical domains.

2.3.2. Qualitative parameter representation

In general, any parameter value that is different from the expected value can be a fault symptom. Thus, representing actual parameter values as well as deviations from reference values is helpful to characterize faults and symptoms adequately. However, in order to increase the accuracy of automated behavior predictions MAD's qualitative parameter representation consists of *three* attributes, i.e., actual value, deviation value and reference value. In [21] we motivate MAD's three-fold qualitative parameter representation in detail.

Qualitative parameter values stand for intervals and landmarks. Only signs of parameter *deviations* are represented, i.e., '−', '0', and '+' are MAD's qualitative deviation values. As an example, Fig. 3 shows MAD's qualitative *absolute* voltage values and their semantics. These values are utilized to characterize *actual* and *reference* values of voltage. In MAD's internal circuit models, there are no quantitative parameter values represented but the value 0. Note that, *negative_landmark* = − *positive_landmark* holds. The meaning of these landmarks is further specified by their utilization in the modeling process. For instance, the value of the supply voltage VCC of the accelerator pedal circuit is modeled by *positive_landmark*. Thus, for all voltage in the

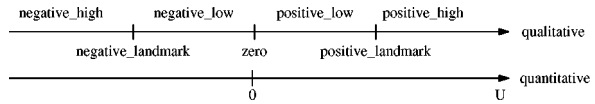


Fig. 3. Qualitative and quantitative absolute voltage values.

device model, *positive_landmark* represents the value of VCC.

Since MAD provides qualitative voltage values *positive_landmark* and *negative_landmark*, in principle, dealing with logical circuits is possible. For instance, logical values (*true*, *false*) can be mapped to MAD's voltage values *zero* and *positive_landmark*. Dealing with logical values is the basis for dealing with hybrid systems consisting of both analog and digital subsystems which are usually found in our application.

2.3.3. Computation of qualitative values

In order to compute qualitative current and voltage values, local propagation methods have been investigated [30]. Since detailed studies proved that local propagation in electrical networks is inappropriate, we follow a different approach first presented by Mauss and Neumann in [20]. Networks are transformed into trees representing the network structure. In particular, series, parallel, star and delta groupings are represented explicitly. Exploiting these structure trees, qualitative behavior can in fact be computed by local propagation.

In general, if interval-based qualitative values are locally propagated, spurious solutions occur [30]. That is, some solutions are not consistent with a global analysis of the device model and, thus, provided the model is accurate, these solutions do not describe electrical circuit behavior. Obviously, fault trees grounded on spurious behavior predictions are useless for fault identification. In order to secure high quality of generated fault trees we developed a new qualitative calculus which shows certain features to improve the accuracy of circuit behavior prediction. In the following, these features are briefly summarized. In [21] they are described in detail.

- First of all, rather than relying on qualitative versions of basic arithmetics, MAD computes qualitative values for currents and voltages by a set of qualitative operators which are qualitative versions of complex quantitative equations. In effect, these equations describe behavior of series, parallel, star and delta groupings. For example, utilizing the well-known equation $R_p = (R1 * R2)/(R1 + R2)$, the compensation resistance R_p

of a parallel grouping of two resistors $R1$ and $R2$ can be computed provided that values of $R1$ and $R2$ are given. MAD's qualitative operators are defined by applying the corresponding quantitative equation to the interval boundaries which represent actual values and reference values of input parameters. Resulting boundaries represent the corresponding qualitative values of output parameters. Qualitative deviation values are computed from actual and reference values. Additionally, output deviation values are inferred from input deviation values if parameter dependencies are monotonous.

To demonstrate that complex qualitative operators are essential, a parallel grouping of two resistors $R1$ and $R2$ is considered. Obviously, $Rp = 0$ holds if $R1 = \textit{infinite}$ and $R2 = 0$ is valid. This result cannot be computed applying qualitative basic operators because qualitative multiplication of $R1$ and $R2$ is undefined if $R1 = \textit{infinite}$ and $R2 = 0$ holds. MAD provides a qualitative operator based on the equation $Rp = (R1 * R2)/(R1 + R2)$ such that, for $R1 = \textit{infinite}$ and $R2 = 0$, $Rp = 0$ is derived.

In principle, for network analysis, a limited number of operators suffices because MAD's internal representation of electrical circuits offers a limited number of standard component models and elementary network structures. Operators are represented by a set of tables comprising more than 30.000 entries which had to be generated by computer in order to secure reliability.

- Second, for computation of qualitative values, MAD utilizes a set of equations which would be redundant if quantitative values were used. It can be shown that if qualitative values are computed, MAD's set of equations is not redundant but avoids some spurious behavior predictions.

To demonstrate that MAD utilizes a 'redundant' set of equations, we again consider a parallel grouping of two resistors $R1$ and $R2$ with corresponding voltage drops and currents $U1$, $I1$, $U2$, and $I2$. Up is the voltage drop across the parallel grouping. Ip is the current through the parallel grouping. If values of $R1$, $R2$, and Ip are given, the value of $I1$ can be computed applying the well-known current divider equation $I1 = R2/(R1 + R2) * Ip$. MAD additionally calculates the value of $I1$ by, first, applying $Up = (R1 * R2)/(R1 + R2) * Ip$ and, second, utilizing $I1 = Up / R1$. If quantitative values were used,

both computations would lead to the same result because, obviously, the set of three equations is redundant. It can be shown that utilization of all three equations avoids some spurious solutions if qualitative values are computed.

- Third, in addition to local propagation of qualitative values, MAD globally analyses network structures and structure trees in order to eliminate (some) spurious predictions for circuits which cannot be structured into series and parallel groupings of standard components. For instance, a global analysis of the network structure allows to determine current directions. Knowledge about current directions can be used to eliminate certain qualitative current values. Therefore, global network analysis may prevent spurious current predictions.

Due to the properties of this qualitative calculus, spurious solutions do not occur at all if the network can be structured into series and parallel groupings of standard components.

2.3.4. Automated behavior prediction

Behavior predictions are performed for all operating modes, faults and fault combinations for which diagnosis support is required. For each operating mode and fault assumption, all symptoms (measurements, observations, error codes, display values) are computed which are in principle available for diagnosis. The output of the prediction step is model-based diagnosis knowledge in form of fault-symptom associations stored the fault relation. This fault relation is the basis for fault tree generation. In Fig. 6, parts of the fault relation of the accelerator circuit example are presented.

2.3.5. COMEDI (COmponent Modeling EDItor)

To improve acceptance among engineers, MAD provides a user interface called COMEDI which is similar to a CAD tool (see Fig. 5). Figure 4 describes the modeling process utilizing predefined models from three different libraries. Providing library models is fundamental because utilization of libraries massively reduces the complexity of the modeling process which is essential for the acceptance of MAD in our application. In the following, the libraries and the modeling process are briefly described. A more detailed presentation can be found in [22].

Due to MAD's internal representation of electrical circuits, it is possible to distinguish 728 different generic component class models. For application component modeling, all these models are provided by the

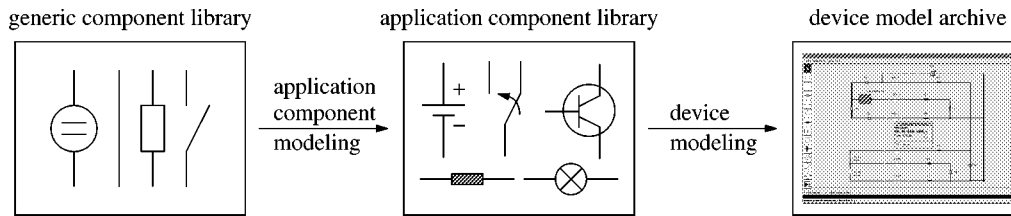


Fig. 4. COMEDI libraries and the device modeling process.

generic component library which cannot be extended. Generic component class models show both correct and faulty behavior modes. Internally, these modes are represented by MAD's threefold qualitative parameter representation. To COMEDI users, behavior modes of generic component class models are described in colloquial language similar to engineers thinking of how components work. By this means, MAD's threefold parameter representation is hidden from users to facilitate the modeling process. For example, a certain generic component class model shows two behavior modes. To COMEDI users, these behavior modes are simply described by *ok_consumer* and *fault_insulator*. Internally, these modes are represented by two different threefold qualitative resistance values. That is, either ($R_{act} = positive, R_{ref} = positive, R_{dev} = 0$) or ($R_{act} = infinite, R_{ref} = positive, R_{dev} = +$) holds.

The application component library contains models of component and subcircuit classes occurring in a certain application. Application component class models can be interactively generated and stored in the library. The ability to extend the application component library is essential for dealing with real world applications because a limited number of predefined application component models cannot cope with a permanently increasing number of different electrical and electronic components utilized for electrical device design. To build a certain application component model, structure and behavior of the model have to be determined. The model structure is generated by assembling generic component class models on the screen. Behavior of application component class models is implicitly given by the model structure and the behavior of generic component class models. Additionally, causal parameter dependencies can be represented in behavior tables based on user-defined qualitative parameter values. By this means, generation of abstract qualitative models for complex components or subsystems such as amplifying circuits, logical circuits, and software controlled components is facilitated. To demonstrate ap-

plication component class modeling, we consider light bulbs which could be blown as a simple example.

The generic component class model described in the previous paragraph is a reasonable application component class model for light bulbs because if a light bulb is blown it shows insulator behavior, otherwise it is a consumer of electrical energy. This generic component class model completely determines structure and behavior of a light bulb application component class model. There are no additional causal parameter dependencies defined. For accelerator pedal circuit modeling, only 11 different application component class models are required. Since we also modeled other circuits, up to now, the application component library consists of about 50 models.

The device model archive allows systematic reuse and modification of device models that were created during former modeling sessions. These models are assemblies of application component models. That is, for device modeling only a structure description is created on the screen.

2.3.6. Modeling and analyzing the accelerator pedal circuit

In Fig. 5, the COMEDI device model of the accelerator pedal circuit is presented. The device operating mode STANDARD is modeled which means that the switch 1S16 is in the position shown in Fig. 1. Correct and faulty behavior of wire 4 is presented in the small window in the center of Fig. 5. To model the error flags UFG_LOW, UFG_HIGH, and UFGS, we utilize functional labeling [25], i.e., strings such as 'OPEN', 'CLOSED', 'OK', and 'NOT_OK' can be attached to MAD's qualitative parameter values. Note that, these strings occur in the fault relation shown in Fig. 6. The manual voltage measurement UG described in Section 2.1 is modeled by a special multimeter component model. Automated behavior predictions are performed for all possible component behavior (correct and faulty) and all device operating modes considered in the accelerator pedal circuit device model. The results are stored in the fault relation. Figure 6 shows fault symptom associations which hold in the device operating mode STANDARD.

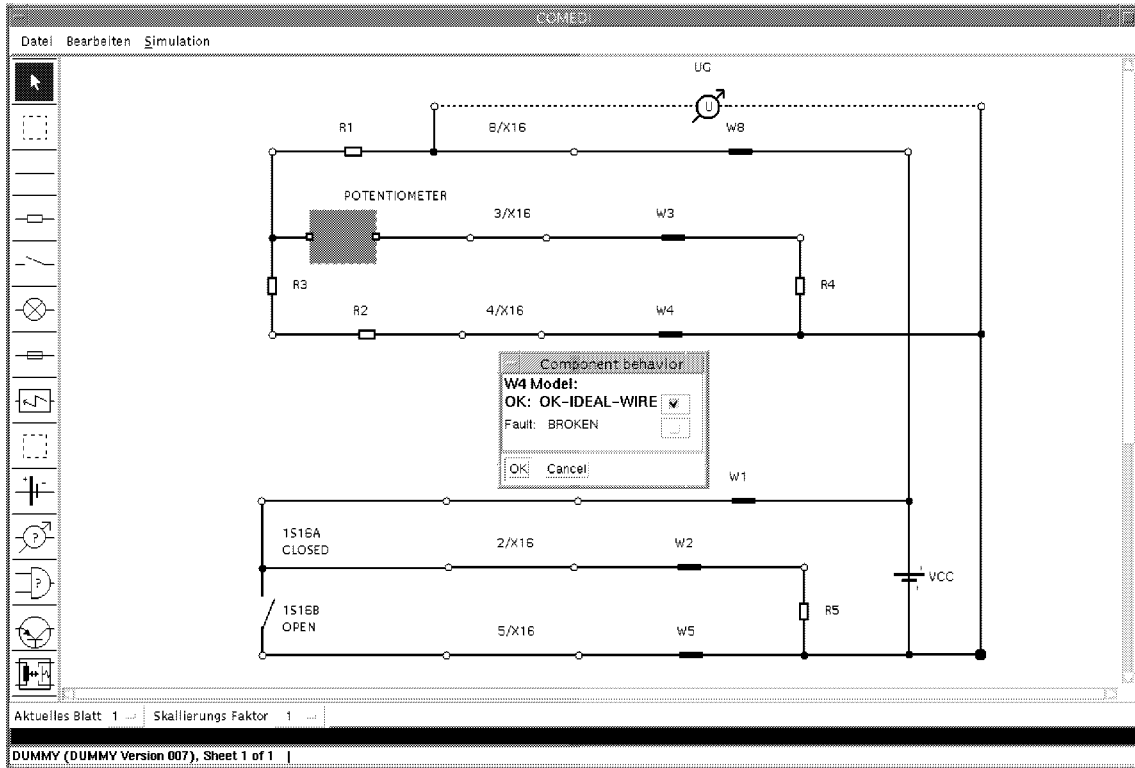


Fig. 5. COMEDI model of accelerator pedal circuit.

Fault Relation Frame						
Behavior modes	Tests:	UFG_LOW_STANDARD	UG_STANDARD	UFG_STANDARD	UFG_HIGH_STANDARD	UFGS_STANDARD
1S16A-is_STUCK-AT-OPEN	OK	N_PMAX_US7	N_PB_US6	OK	CLOSED	
W5-is_BROKEN	OK	N_PMAX_US7	N_PB_US6	OK	OPEN	
W2-is_BROKEN	OK	N_PMAX_US7	N_PB_US6	OK	CLOSED	
W1-is_BROKEN	OK	N_PMAX_US7	N_PB_US6	OK	CLOSED	
W8-is_BROKEN	NOT_OK	L_0_US7	L_0_US6	OK	OPEN	
W4-is_BROKEN	OK	N_PMAX_US7	H_PB_US6	NOT_OK	OPEN	
W3-is_BROKEN	NOT_OK	N_PMAX_US7	L_0_US6	OK	OPEN	
PEDAL-is_DEFECT	NOT_OK	N_PMAX_US7	L_0_US6	OK	OPEN	
PEDAL-is_NOT-ADJUSTED	NOT_OK	N_PMAX_US7	L_PB_US6	OK	OPEN	
VCC-is_LOW-VOLTAGE	NOT_OK	L_PB_US7	L_PB_US6	OK	OPEN	
VCC-is_HIGH-VOLTAGE	OK	H_PIMP_US7	H_PB_US6	NOT_OK	OPEN	
VCC-is_NO-VOLTAGE	NOT_OK	L_0_US7	L_0_US6	OK	CLOSED	
OK-CASE	OK	N_PMAX_US7	N_PB_US6	OK	OPEN	

Fig. 6. Parts of fault relation of the accelerator pedal circuit.

2.4. Fault tree generation

MAD offers three different possibilities to generate fault trees. First, based on fault relations, fault trees

can be created automatically. Second, fault trees from archives can be reused. Third, in order to permit manual adoption and modification of fault trees, MAD offers basic editing operations, such as moving a certain

fault from one fault set to another and recomputing the corresponding tests. In the following, automated fault tree generation is presented in more detail. One can choose from the following criteria to guide fault tree generation.

- **Grouping by observations, error codes, display values.** Fault trees are generated such that subsets of faults correspond to a prespecified symptom. For instance, all faults are grouped together which lead to an unexpected value of the accelerator pedal voltage UFG.
- **Grouping by aggregate structure.** If the aggregate structure of the device is known, fault trees can be generated such that subsets of faults correspond to the same physical component. For instance, faults occurring in the ECU may be grouped together.
- **Minimization of average diagnosis cost.** Automated fault tree generation uses the well-known A*-algorithm to select the tests which minimize the average fault identification cost according to a cost model.

Since minimization of average diagnosis cost is essential for the acceptance of MAD in our application, in the following, we describe MAD's utilization of the A*-algorithm for optimized fault tree generation. We do not take fault probabilities into account because, in our application, these probabilities are not available. A promising approach for fault tree generation considering fault probabilities is presented in [11].

Cost optimized fault tree generation is based on the A*-algorithm which performs a search in a state space. A state contains a set of fault sets which are the current leaves of a growing fault tree. The start state consists of one fault set containing all faults of the fault relation. The goal state consists of fault sets containing faults which cannot be discriminated. A successor of a state is generated by partitioning one leaf fault set into at least two subsets by selecting a partitioning test T . All successors of a state are generated by applying each partitioning test to each leaf. Each path in the state space represents a possible fault tree. A test shows corresponding costs and a set of different possible test results, i.e., a test domain. In general, tests are not exclusive. That is, if a test partitions a fault set into subsets, certain faults can occur in more than one subset. Each state is evaluated by the functions g and h whose definitions are presented in Fig. 7. g is defined as the sum of the diagnostic effort for each fault f . The diagnostic effort of a fault f is the sum of all test cost $C(T)$

on the path between the current leaf fault set containing f and the root fault set. To guide the search, the heuristic function h estimates cost of fault identification assuming that, in the fault identification process, a certain state is already reached. For details of the A*-algorithm see [27].

To demonstrate that h never overestimates real cost of fault identification in a cost-optimized tree, a certain leaf b is considered. There is a set of available tests T_i not yet used on the path between b and the root. These test allow the generation of a cost-optimized subtree below b . Available tests show costs c_i and domains. k_{mx} is the maximum size of these test domains. The following two properties guarantee that the heuristic function h underestimates fault identification cost in a cost-optimized subtree. First, h considers an impossible subtree in which fault identification is cheaper than in a cost-optimized subtree. Second, rather than precisely computing fault identification cost in the impossible subtree h underestimates diagnosis cost.

The impossible subtree and the cost-optimized subtree show the same faults but different tests. The following characteristics secure that fault identification in the impossible subtree is cheaper than in the cost-optimized subtree. First, in the impossible subtree, all available test are exclusive. That is, if a test partitions a fault set, each fault occurs in only one subset. Second, all available tests split fault sets into k_{mx} subsets. Due to these two properties, in the impossible subtree, the discriminating power of available tests is higher than in the cost optimized subtree. Third, in the impossible subtree, the test first performed for fault identification, is as expensive as the cheapest available test of the cost-optimized subtree. All tests performed in the second level of the impossible subtree are as expensive as the second cheapest test of the cost-optimized subtree, and so on. Fourth, in the impossible subtree, if a fault set is partitioned into subsets, all of these subsets contain the same number of faults, i.e., the impossible subtree is balanced. Provided the first three properties hold, a balanced tree yields to lowest fault identification cost.

For the estimation of fault identification cost in the impossible subtree, it is assumed that the depth of the subtree is $\lfloor \log_{k_{mx}}(|b|) \rfloor$, ($\lfloor \dots \rfloor$ is the floor operation) which, in general, is an underestimation. Based on this assumption, for each fault in b , cost of fault identification in the impossible subtree can be underestimated as:

$$g(\text{state}) = \sum_{i=1}^n de(f_i), \quad \text{with} \quad de(f_i) = \sum_{T \in \text{path}(f_i)} C(T)$$

$$h(\text{state}) = \sum_{b \in \text{leaves}(\text{state})} h(b), \quad \text{with} \quad h(b) = |b| \cdot \sum_{i=1}^{\lfloor \log_{k_{\max}}(|b|) \rfloor} C(T_i),$$

with T_i is i -th cheapest unused Test T

Fig. 7. Cost function g and heuristic function h for computing an optimal fault tree with n faults f_i . de denotes the diagnostic effort.

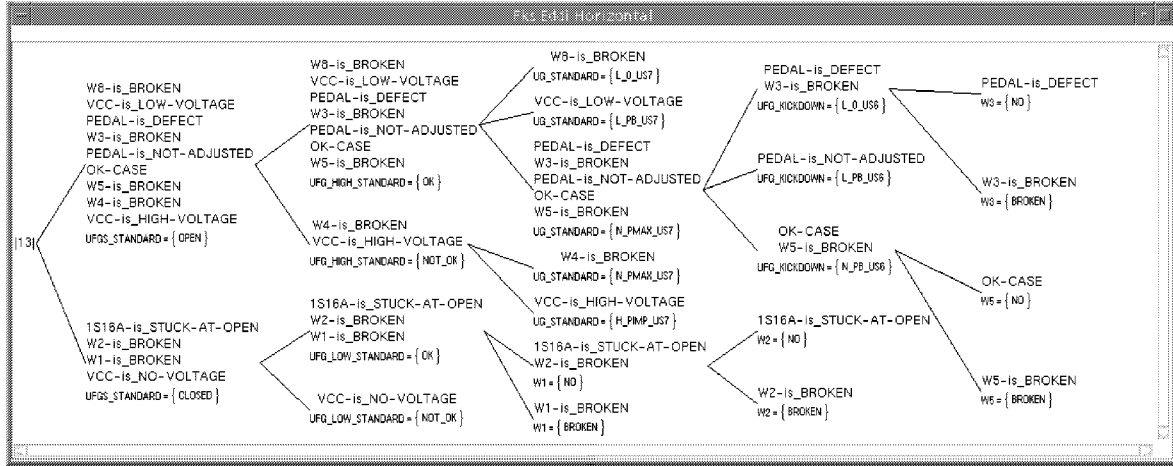


Fig. 8. Cost optimized fault tree of accelerator pedal circuit.

$$\sum_{i=1}^{\lfloor \log_{k_{\max}}(|b|) \rfloor} C(T_i), \quad \text{with } T_i \text{ is}$$

i -th cheapest unused Test T .

Figure 8 shows a cost optimized fault tree of the accelerator pedal circuit. Note that, in the figure, the final row of a fault set is the corresponding test. The root contains 13 faults which are not explicitly shown. Note that, in MAD fault trees, faults are sometimes discriminated by so-called direct fault identifications such as $w3 = \{BROKEN\}$. Direct fault identifications are described in [22].

2.5. Evaluation

The prototypical implementation of MAD allows model-based behavior prediction and automatic generation as well as manual modification of fault trees. All

faults considered in the device model occur in the generated fault tree, and tests are selected correctly to discriminate fault sets. This holds even when fault trees are modified manually. Furthermore, average diagnosis cost is minimal within the constraints imposed by a prespecified fault tree structure. Fault trees have been successfully integrated into existing STILL diagnosis systems.

MAD has been evaluated in the STILL application scenario and it has been found that using the modeling techniques of MAD with some extensions regarding the implementation of certain network analysis concepts, more than 90% of the faults of the current hand-crafted diagnosis system can be handled successfully. In some cases, since component-dependent parameter threshold values are not explicitly represented in MAD models, in fault trees, correct and faulty behavior cannot be completely distinguished. Using MAD, an accelerator pedal fault tree was automatically generated from the circuit model and imported into the STILL di-

agnosis system. STILL service experts found that this fault tree can be used for fault identification.

3. Model-based support to diagnosis and fault analysis in the automotive industry

3.1. The applications

In collaboration with Robert Bosch GmbH as a major supplier of mechatronic car subsystems, the Model-based Systems and Qualitative Modeling Group (MQM) at the Technical University of Munich worked on three prototypes that support different tasks related to fault analysis during the life cycle of a product:

- **Failure-Mode-and-Effects Analysis (FMEA).** This task is performed during the design phase of a device. Its goal is to analyze the effects of component failures in a system implemented according to the respective design. The focus is on assessment of the criticality of such effects, i.e., how severe or dangerous the resulting disturbance of the functionality is in objective or subjective terms (e.g., inconvenience for the driver, environmental impact, risk of hazards). In addition, the probability of the fault and its detectability is considered. Based on this analysis, revisions of the design may be suggested. FMEA is performed in a growing number of areas. Particularly for vehicles, due to safety and environmental aspects, it has to be as ‘complete’ as possible, not only w.r.t. the faults considered, but also in the sense that all possible effects under all relevant circumstances (driving conditions of a vehicle, states of interacting subsystems, etc.) have to be detected.
- **Workshop diagnosis.** The diagnostic task in the repair workshop starts from a set of initial symptoms which are either customer complaints or trouble codes generated and stored on-board by one of the electronic control units responsible for various subsystems of the vehicle. Except for the obvious cases, some investigations, tests, and measurements have to be carried out in order to localize and remove the fault, usually by replacement of components.
- **Generation of diagnosis manuals.** The mechanic in the repair workshop is educated or guided by diagnosis manuals produced and distributed by the corporate service department (on paper, CD-ROM, or via a network). Here, engineers compile various kinds of information (tables, figures, text)

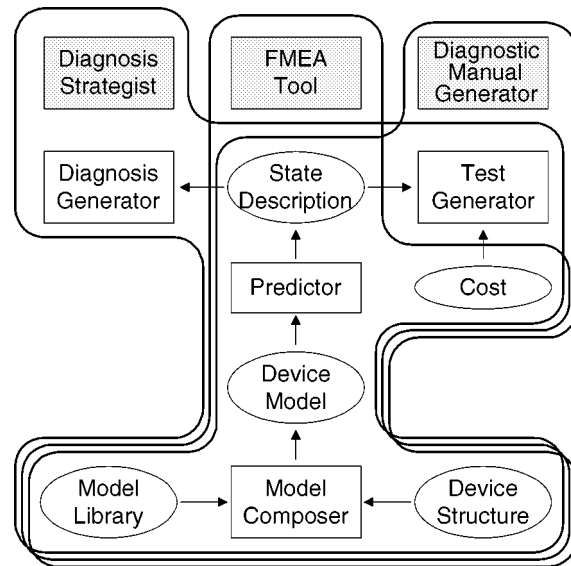


Fig. 9. The three tools and their software components.

which is required or useful for carrying out the diagnosis. Such documents have to be produced for each variant of the various subsystems and specific to a particular make, type, and special equipment of a vehicle, a broad set of issues for a supplier. The core of each document is a set of test plans that guide fault localization starting from classified customer complaints or trouble codes of the ECU.

In practice, these tasks are usually not extremely difficult to perform by an expert. However, they can be time-consuming since they have to be carried out for each specific instance of a general device type which includes collection of all the information specific to this instance. This situation of routine work applied to a large set of variants justifies computer support to be developed. And because knowledge about physical devices is the key to solving the tasks, model-based systems offer a perspective.

In the INDIA project, we built software prototypes for the above tasks by developing model libraries for the domain and by using software components for model-based reasoning of the commercial RAZ'R system. Figure 9 indicates the components used for the various systems and shows that the degree of re-use is fairly high. In this paper, we discuss the task of the generation of diagnosis manuals with a focus on the generation of test plans.

3.2. Automated test generation

Testing a system means manipulating it by changing its structure and/or providing a certain stimulus (input) to it in order to obtain observations that help to gain information about its mode of behavior. If a system is composed of individual components that are assembled in a fixed structure and if component faults are considered as the only origins of malfunctions, testing aims at distinguishing different behavior modes of its components. For instance, testing a newly manufactured device aims to check whether all components function correctly or whether one of them has a fault (usually done by trying to rule out all, or the most likely, single faults). Testing performed to serve diagnosis tries to identify the fault (or class of faults) that is present. Since it is impossible for principled reasons to identify a fault (class) by directly confirming it through observations, it can be done only by looking for tests that help to refute all other (classes of) faults. Therefore, test generation and testing, in contrast to diagnosis, relies on knowledge about the (enumerable) set of faults that are considered.

Model-based methods offer themselves as a basis for supporting and automating these tasks, because behavior models can capture knowledge about the correct and the faulty behavior of the components involved. We have developed the theory underlying model-based (component-oriented) test generation some time ago ([31,32]). Here, we refrain from a formal presentation and only try to convey the intuition. In the theory and the respective algorithms, the behavior models of the system to be tested (e.g., the modes established by all single faults) are represented as relations in the space of its variables, state variables, and parameters. Figure 10a shows two abstract behavior relations as Venn diagrams (ignoring the temporal aspects of the behav-

ior). Testing, i.e., mode discrimination, is the task of detecting the differences between these sets, i.e., tuples that are not part of all relations. More precisely, we are looking for some input to the system (fixing the values of some ‘causal’ variables, i.e., the ones that can be influenced), e.g., $\text{var}_i = \text{val}_{i,1}$ on the vertical axis in Fig. 10a, such that the possible observations under the different modes are differing as much as possible. The possible resulting observations in var_{obs} , indicated on the horizontal axis in Fig. 10a, b, are given by the projections of the intersections of $\text{var}_i = \text{val}_{i,1}$ with the different behavior relations to the observable variables var_{obs} . The example suggests that $\text{var}_i = \text{val}_{i,1}$ is a bad choice because these projections have significant overlap, while $\text{var}_i = \text{val}_{i,2}$ is certain to distinguish mode₁ from mode₂. If probabilities of the behavior modes are available (and/or if behavior descriptions are probability distributions over the relations), it is possible to use the minimization of entropy as a criterion for selecting the test with maximum information gain (see [32] for the details).

Obviously, for continuous systems, computing the respective projections for the possible inputs ranges from very complex to unfeasible. It can be made feasible by applying qualitative instead of numerical models, symbolized in Fig. 10b by moving to the finite granularity of the rectangular models. When we assure that the qualitative behavior relations cover the more fine-grained ones, we can apply the same computational steps as above, which are now operations on finite sets: the number of qualitative distinctions is finite, and so is the number of tuples. Furthermore, the set of faults becomes finite, because the qualitative abstraction aggregates continuous sets of faults (e.g., leakages of different size) into one fault model.

When applied to electrical circuits, qualitative models do not relate numerical values of current and volt-

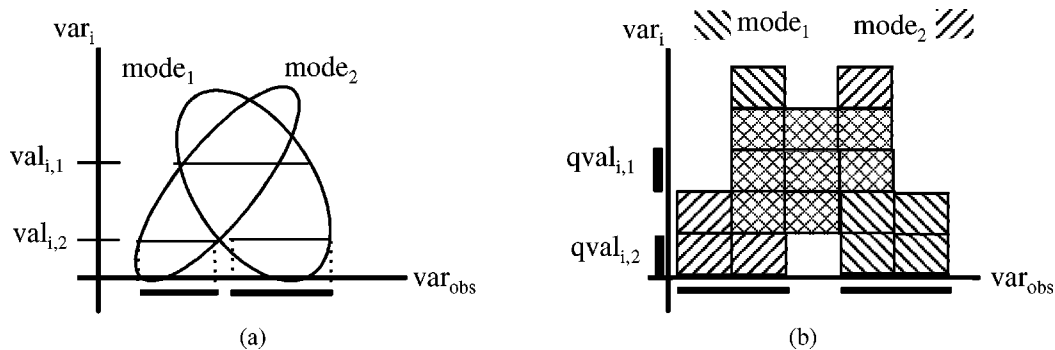


Fig. 10. Quantitative (a) and qualitative (b) abstract behavior relations.

age, but, for instance, capture only the direction of current, represented by values negative ('neg'), positive ('pos'), and zero ('0'). The domain of the voltage variables could consist of values 'gnd' (for ground), 'bat' (representing the battery voltage), and 'btw' (the interval between). With these domains, a model of the correct behavior of a temperature sensor (or any resistor) is given by the relation shown in the left-hand table in Fig. 11, where the value pairs along the vertical axis represent the combinations of voltages on either side of the sensor, and the horizontal dimension represents current. An open circuit fault is characterized by a zero current for all voltages which yields the relation on the right-hand side. From a comparison of the two relations, one can directly identify (and compute by a straightforward algorithm) which inputs (qualitative voltage pairs) lead possibly or deterministically to different observed values for the current.

Thus, qualitative modeling allowed us to implement test generation algorithms and apply them to relay circuits, a problem extracted from a real application of programming test automata for circuits of a train station control system ([18,19]). Although the implemented system produces provably correct sets of tests (and identifies behavior modes that cannot be discriminated under the given sets of causal and observable variables) for examples of fifty or so components, for which the task of test generation is already beyond an exhaustive treatment by humans. However, it is still far from an application system or usable tool. There are two fundamental limitations. The solution treats test generation as a very abstract task and ignores the practical aspects of testing (except for the restriction on causal and observable variables): the equipment and cost involved, sequencing of tests to reduce these costs,

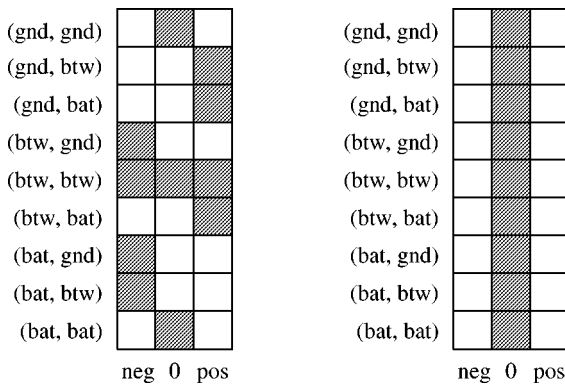


Fig. 11. Qualitative behavior relations for a correct sensor/resistor (left) and one showing an open circuit fault (right).

the fact that certain tests are risky or impossible unless some faults have been ruled out, etc.

Secondly, even if the model-based test generation is extended appropriately to include such aspects, becoming a useful tool requires to reflect the actual work processes that deal with the generation or execution of test plans. In our domain, this is mostly done in the service department. Here, a significant amount of work is dedicated to creating, checking, adapting, updating, and translating diagnosis manuals which are distributed as guidelines to the service stations (in our case on CD-ROM). The core of these documents is formed by the textual description of plans for performing testing in the workshops with the goal of fault localization. Hence, one natural way to introduce test generation is obviously to support the authors in the service departments in their task of producing such manuals. We started this by specifying and implementing a new type of authoring system, called Genesis.

3.3. Generation of diagnosis manuals

Genesis is a software tool that combines test generation with the authoring of repair manuals. It combines model reuse with text reuse and provides a knowledge representation level for device data.

We analyzed the conventional authoring process of a repair manual in order to couple test generation and repair planning. Traditional authoring systems for repair manuals are essentially word processors or publishing systems. They partially support the publishing process, like document release and translation, but do not exploit the strong structuring of repair manuals nor do they assist the author in keeping the repair manual consistent. Furthermore, these text processors cannot export documents using semantically tagged document formats like SGML or XML because they have no notion of semantics.

Our aim was to improve upon a conventional text processing system in three stages:

1. Provide a stronger and more detailed structuring of repair manuals. Associate words, phrases or sections of a document with their semantics.
2. Supply a knowledge representation level that relates text fragments from the repair manuals with a representations of the real world system structure.
3. Enhance the represented system structure with a library of behavioral models for the device and its components as a basis for the integration and exploitation of model-based techniques, such as automated test generation.

The first stage of Genesis aims at a sophisticated reuse of textual fragments from existing repair manuals. This addresses translation issues, which are one of the main expense factors involved in the distribution of repair manuals. There are hundreds of vehicle variants, but the repair manuals commonly build upon only a handful of different types of tests and actions. For the diagnosis of electrical systems, which constitutes the main part of the manuals, these are steps like turning on the ignition, pulling plugs, measuring voltage and resistance, reading signal values, and comparing them with nominal values (refer to Fig. 12 for an example). Hence, the same verbalizations occur, with slight variations, again and again. In Genesis, this issue is exploited by splitting the text of a repair manual into small coherent phrases, which are translated individually. The translation of a manual assembled from these text fragments does not require any additional effort. As a side effect, the repair manuals become more uniform, even if they are written by different authors, because they are all composed from the same repository of verbal expressions. Each text fragment serves a specific semantic role in the final repair manual. The fragments are organized according to this role, which facilitates their retrieval. Moreover, these semantics enable a transformation of the repair manual into a semantically tagged document format like SGML or XML.

Figure 12 is a screen-shot of this first stage of the system. The left part of the screen displays the struc-

ture of the repair manual, by which the author can navigate through the chapters and pages of the document (in this case a manual for a fuel injection system). When a page is selected, its contents is displayed in the second window. This window is also used for editing text. Words and phrases that are connected to physical properties of the vehicle are displayed in bold – components are underlined, parameters are not. The third view verifies that the page is actually composed from semantic text fragments. It illustrates that text fragments can themselves be composed from other text fragments, for example a *TestStep* consists of a list of *SetupActions* and a *Test*. Some text fragments are not visible because the tree has not been fully expanded in this display.

The second stage of Genesis extends the semantic representation of text with a model-based knowledge representation of the vehicle. This vehicle model contains a hierarchical representation of parts and components, a description of the vehicle topology and data sheets with device parameters. The knowledge representation level associates vehicle specific text fragments with their physical correspondent in the model. A main advantage for the author is that he can apply changes to the vehicle model without having to go through the whole repair manual. In fact, if the vehicle data are available in electronic form (e.g., as CAD data), they could be imported via an interface. Genesis

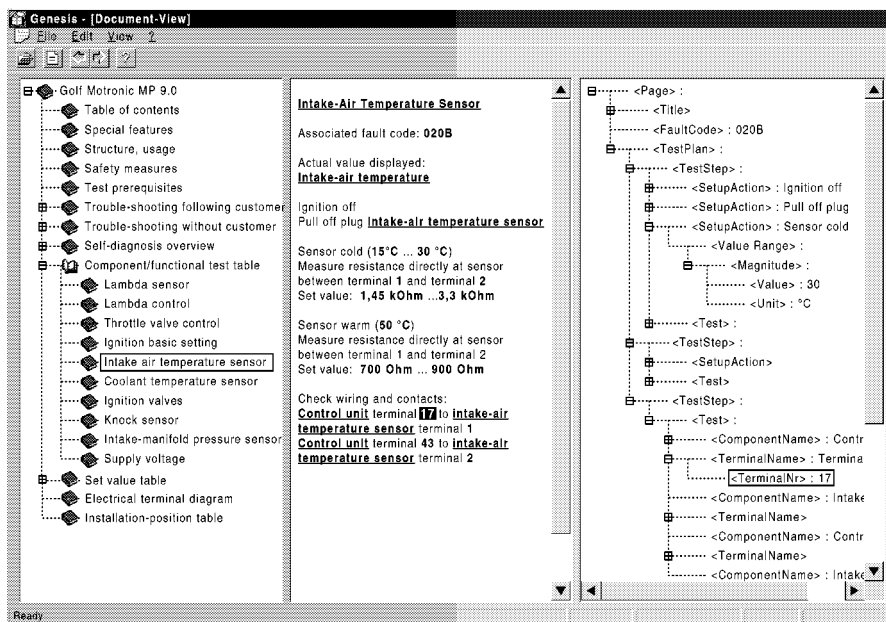


Fig. 12. First stage of Genesis.

will automatically update all references to the changed data. This leaves the arrangement of tests and actions to repair plans unaffected. If the structure of the repair plan itself requires modifications (e.g. because a sensor plug is no longer accessible in the vehicle variant), with the current tools, these modifications have to be carried out by hand.

Figure 13 shows a snap-shot of the additional views in this second program stage. The top left window displays a hierarchical view of the device structure. The window below presents the data sheet of the selected component, in this case the intake-air temperature sensor. The right windows contains a diagram for the selected component (if available). The author can create references to components, terminals or parameters by selecting them in either of the three views.

The third and final stage of Genesis employs techniques of model-based reasoning. Stage two already encourages the author to explicitly represent the structure and characteristics of the vehicle subsystems. The third stage combines this information with models of component behavior, as they are used in model-based diagnosis or design. The resulting behavior model of

the vehicle can be employed to perform automated reasoning tasks, such as the generation of complete repair plans.

3.4. Model-based test plan generation for genesis

Further development of the automatic test generation outlined in Section 3.2 into a useful add-on to Genesis required to include the practical aspects of the diagnosis and testing task, such as the actions and cost involved. In accordance with [13], we discriminate between generic and pragmatic aspects:

Generic models are not task specific. They describe the general physical principles underlying the **behavior** of a type of component or device. An integrated model-based system will tend to share generic models for testing with other reasoning tasks like design or quality analysis. The generic models have been created using the commercial model-based diagnosis environment RAZ'R ([23]). This environment provides facilities for modeling generic behavior of components, as well as for defining parameters, states and the structure of devices. RAZ'R includes algorithms for model com-

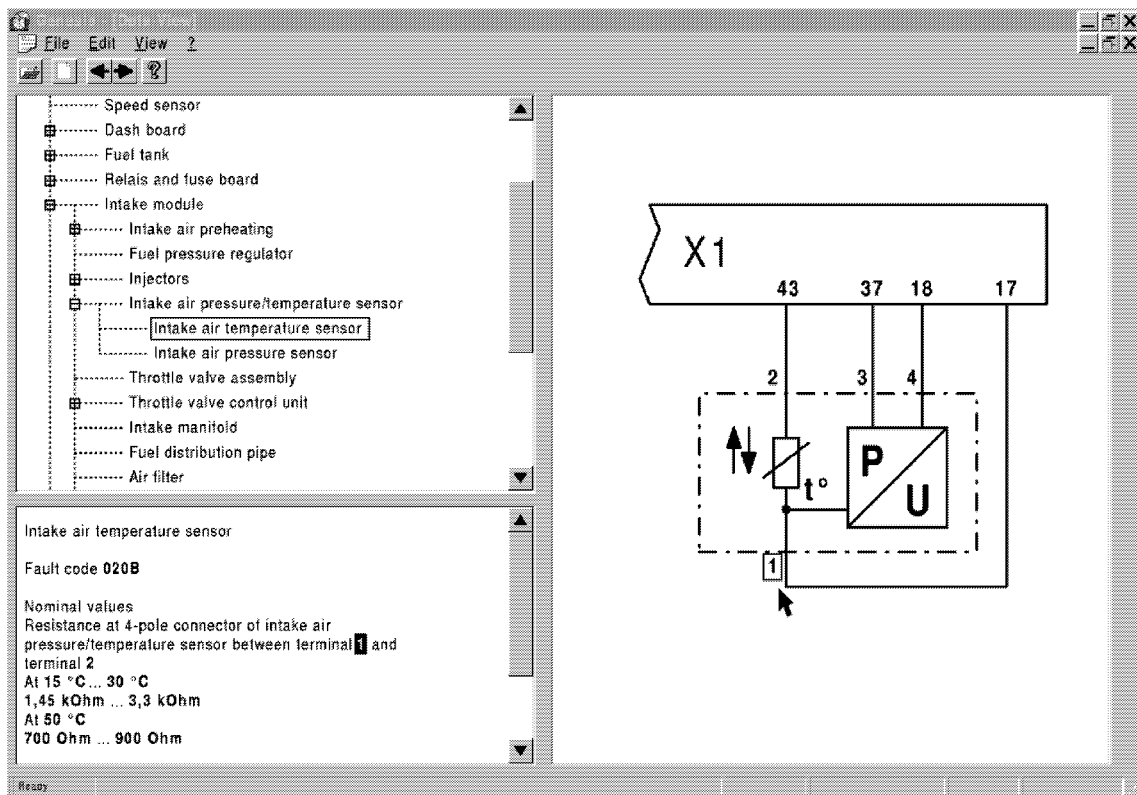


Fig. 13. Second stage of Genesis.

position and can export fully composed device models using XML.

Pragmatics are given by aspects of the components or the device which are considered specific to a particular device and its physical realization and/or a task as it is carried out in reality. In the testing and repair context this comprises information like

- probing points and their accessibility,
- available test equipment,
- assembly and disassembly actions,
- setup, treatment and repair actions,
- action costs, preconditions and safety constraints,
- description of device symptoms,
- failure probabilities.

Such aspects have to be captured by a representation layer that is separate from, but related to elements of the generic behavior models. For instance, a variable in the behavior model might be linked to a probing point and its accessibility conditions in the pragmatics.

It is important to note that there are some interdependencies between the generic behavior model and the pragmatic aspects of testing that can complicate the task significantly. For instance, measuring the resistance of, say, a sensor requires disconnecting it from the control unit and attaching the measurement device which amounts to a significant structural change of the original device. In our current solution, we have chosen to include the respective potential structural changes in the device model by adding switch connections to the measurement devices. Clearly, this is not a satisfactory and principled solution.

Nevertheless, we chose, to maintain the distinction between behavior models and representation of the pragmatics for reasons of re-usability: because the former are generic, they can be reused in different tasks and for devices that share the principled features of behavior, the latter can vary considerably in form, content, and the reasoning schemes required. For instance, for different types of vehicles, the device topology for the temperature sensor circuit will look like the one depicted in Fig. 13. But the physical layout and installation could be vastly different, including variations in accessibility of probing points, distance between components etc. A typical consequence is that the measurement device has to be connected to different plugs.

As a consequence, we also decided to maintain the test generation algorithm described in Section 3.1 (which operates on the generic behavior model only) as a separate software component. Rather than extending this *abstract test generator* to include the pragmatic as-

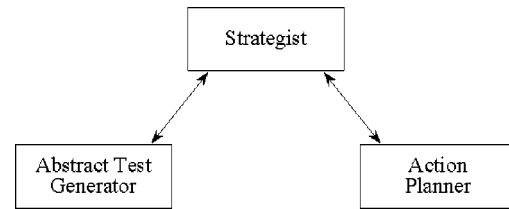


Fig. 14. The three modules of the repair plan generator.

pects and their exploitation, we organize this knowledge and reasoning in a separate component, the *action planner*, and coordinate the two components by a *strategist* component (see Fig. 14).

This approach differs from the one applied in the fork lift application (Section 2) and decision tree generation for on-board diagnosis as described in [5]. They treat the task basically as a **classification** problem, while we consider a **planning** approach as the ultimate appropriate solution which can take into account actions, goals, cost etc. in a more flexible way (although we have not employed a sophisticated planner at this time). The alternative solutions can be seen to reflect different importance of actions and cost in on-board vs. off-board diagnosis.

Dependent on the relevant criteria for the quality and cost of repair plans, the action planner can vary, and the strategist can be used to organize different ways of interaction between the abstract test generator and the planner. Figure 15 shows the interfaces and major phases of the iterative repair plan generation process for Genesis which we will briefly explain in the following.

This input of the strategist comprises

- a **behavior model** of the device that is to be tested,
- a description of the available **actions** and possible **observations**,
- **pragmatic information** about their cost, preconditions, etc. (For example, the connector of the electronic control unit will only be accessible if the ECU housing is accessible and open. To get to the ECU housing, depending on the location, the hood may need to be open and a splash guard removed,)
- a description of the **current situation** which changes dynamically at each stage in the generation process.

The latter covers

- the **system state** as expressible by the behavior model (e.g., switch positions, engine temperature, etc.)

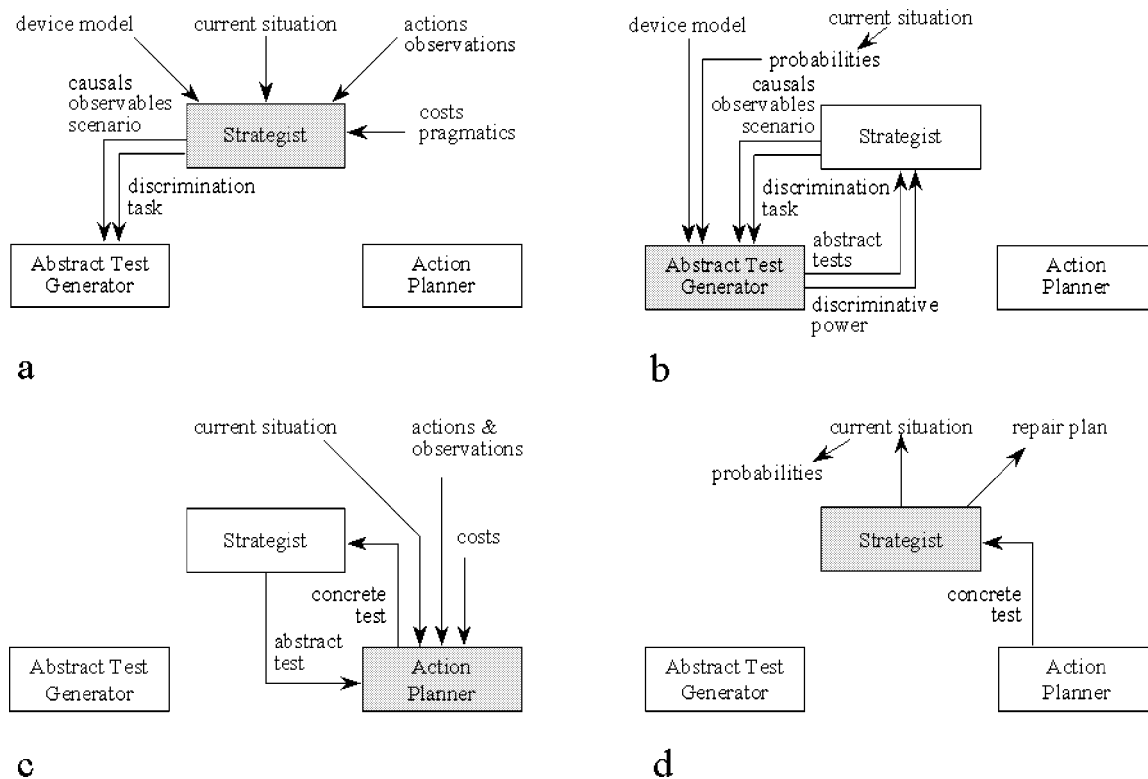


Fig. 15. Four phases of a cycle in the plan generation process.

- the state of the **physical system** as a result of previous actions, for instance installation of measurement devices, disassembly, etc.
- the **state of the problem solving process**, e.g., given as the set of remaining fault hypotheses and their probabilities.

At each stage (basically each node in a discrimination tree), the strategist generates a discrimination task (sets of faults to be discriminated) for the abstract test generator and, according to the current situation, identifies the variables that can be influenced and observed as well as a *scenario*, the current system state and further restrictions (see Fig. 15a). The latter may reflect, for example, critical states to be avoided. This results in a restriction of the behavior model by an additional relation. For example, a first step in a test plan may be to determine whether the device has some critical fault, which requires additional safety measures and restrictions on tests. After critical faults have been exonerated, the discrimination focus can be refined to discriminate among the remaining faults, which will then be easier to perform because the safety measures can be relaxed.

Based on this input and fault probabilities, if available, the abstract test generator produces candidate tests based on the principles described in Section 3.1 along with a characterization of their discrimination power in terms of remaining sets of fault hypotheses under different test outcomes and, perhaps, changes in entropy (Figure 15b). A test is characterized in an abstract way as a set of input values including directly accessible states and a set of variables to be observed.

While the *abstract test*, thus, characterizes *what* happens when the test is performed, the plan needs *concrete tests* providing details on *how* to perform the test. For example, an abstract test for a car may state that we want to determine the voltage across the battery. A concrete realization of this abstract test could be to open the hood, disconnect the battery, connect a voltage meter and observe its display. Obviously the concrete realization depends on the current situation, because if the voltage meter is already connected to the battery, the test is reduced to just observing the display.

The strategist selects one of the abstract tests to be turned into a concrete test by the action planner, if possible. Injecting the required input and observing the relevant variables specifies the goal which the planner

```

<TestPlan>
<TestStep ID=1>
<SetupAction> Connect pocket-tester to diagnosis plug
<Test> Retrieve value for intake-air temperature signal with pocket-tester
<TestResult NextStep=9> Signal between -20°C and +40°C
<TestResult NextStep=2> Signal below -20°C
<TestResult NextStep=5> Signal above +40°C
<TestStep ID=2>
<SetupAction> Disconnect plug of processing unit
<SetupAction> Select "Voltage measurement" at multi-meter
<SetupAction> Connect multi-meter to plug of processing unit, pins 17 and 34
<Test> Retrieve value for intake-air temperature signal with pocket-tester
<TestResult NextStep=3> Signal between -20°C and +40°C
<TestResult NextStep=4> Signal below -20°C
<TestStep ID=3>
<RepairAction> Replace intake-air temperature sensor and its connector
<TestStep ID=4>
<RepairAction> Replace processing unit
<TestStep ID=5>
<SetupAction> Disconnect plug of processing unit
<SetupAction> Select "Voltage measurement" at multi-meter
<SetupAction> Connect multi-meter to plug of processing unit, pins 17 and 34
<Test> Retrieve value for intake-air temperature signal with pocket-tester
<TestResult NextStep=6> Signal above +40°C
<TestResult NextStep=8> Signal between -20°C and +40°C
<TestStep ID=6>
<RepairAction> Replace power supply
<Test> Retrieve value for intake-air temperature signal with pocket-tester
<TestResult NextStep=9> Signal between -20°C and +40°C
<TestResult NextStep=7> Signal above +40°C
<TestStep ID=7>
<RepairAction> Replace processing unit
<TestStep ID=8>
<RepairAction> Replace intake-air temperature sensor
<TestStep ID=9>
<Comment> No fault found.

```

Fig. 16. Test plan of temperature sensor circuit.

has to achieve based on the available actions respecting or establishing their preconditions and observing cost of actions (Fig. 15c). As mentioned earlier, we only have a basic implementation of this component which nevertheless seems appropriate for the problems posed by the existing repair plans. In general, though, we feel that much more research needs to be dedicated to the development of more powerful model-based planners.

In the final phase of each cycle, the strategist extends the repair plan by a selected concrete test (mainly based on cost criteria) and updates the current situation to the ones after the application of the test and the observation of its different possible outcomes (Fig. 15d).

The integration of abstract test generation and action planning makes the algorithm more effective, because, in general, the process of computing a concrete realization for an abstract test involves a planning task, which can become cumbersome. Early pre-selection thus helps to keep the plan generation task feasible. Moreover, abstract tests mainly depend on the device model and accessibility conditions. If the test generator is used in an interactive repair planning context, where it is invoked at different stages, much of the reasoning involved in abstract test generation remains valid and can be reused. If the device model does not change, only the ranking of tests and, of course, the concrete

realizations, need to be re-computed to fit the new situation.

In Fig. 16, a plan (including some verbalization for the sake of readability) is presented which is generated by the system for a temperature sensor circuit similar to the one depicted in Fig. 13. As can be seen, the plan has a structure and tagging of the abstract representation of a test plan in Genesis, as it is presented in the right-hand window of Fig. 12. Hence it can be imported to the authoring system and automatically turned into a textual representation based on the primitive text building blocks for the different nodes in the graph. Thus, the results of the model-based system take on the same form as the current test plans and can be further processed without requiring new tools or immediate changes in the actual work process.

4. Exploiting statecharts in modelling for diagnosis

4.1. Application scenario

One of the three domains investigated in INDIA has been dyehouse machinery as developed by THEN GmbH. In particular, Fraunhofer IITB has looked at post mortem diagnosis of the chemical distributor (CHD), a system to dose and distribute chemicals and colours in a dyehouse automatically (see Fig. 17). A failure of this system may stop the overall production and damage material as well as environment considerably. It is therefore important to identify and remove the underlying fault as fast as possible. A post mortem diagnostic system that assists dyehouse staff with this task can considerably reduce downtime, since in many cases it allows them to get the CHD up and running again without having to wait for THEN's service technicians.

Although the benefits of a diagnostic system for CHD are therefore quite clear, developing such a system has been out of question for THEN before we started the INDIA project. One of the main reasons is that introducing this technology into a company is too expensive. Firstly, using today's tools and techniques requires not only a lot of experience in diagnosis, but knowledge about concepts from Artificial Intelligence or mathematics that are unknown to the ordinary domain expert. Hence, either domain experts have to be trained on the tools or diagnostic experts have to be introduced to the domain – both costly alternatives. Secondly, existing documents with engineering knowledge are hard to re-use or to import into the diagnostic

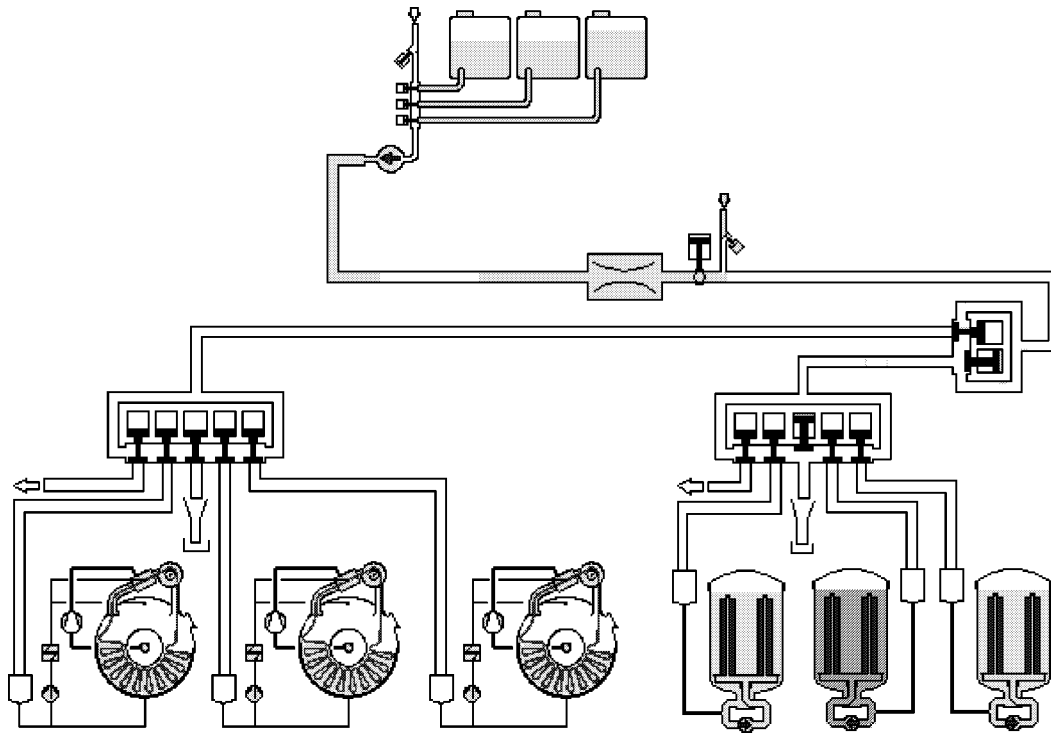


Fig. 17. Sketch of the Chemical Distributor CHD (© THEN GmbH).

knowledge base. Therefore, knowledge has to be represented several times, with additional costs for maintaining consistency.

These reasons are not specific for THEN but, in our experience, are typical for many manufacturers of machinery. They have to be overcome to foster a more widespread industrial application of intelligent diagnostic systems.

This chapter sketches Fraunhofer's approach to reduce these costs considerably. It is based on our vision to integrate not only knowledge for diagnosis with existing engineering or product data; but to integrate business processes: development of diagnostic systems for a device with development of the device itself. To achieve a smooth integration, diagnostic development should rely on the same people and found on existing knowledge and tools.

4.2. The chemical distributor CHD

In the course of dyeing, specific mixtures of chemicals have to be filled in the dyeing machine about three or four times. CHD automates measuring every chemical in the mixture off its tank, transporting the mixture through a network of pipes, valves and flaps to the re-

questing dyeing machine and finally rinsing the pipes involved with water.

Figure 18 shows the structure of the network. The basic components are the PLC, a flow meter, pipes, valves, pumps and flaps. Valves are grouped into collectors, which join different input pipes to an output, and distributors, which multiplex a single input pipe to multiple outputs. The flap between drain valve and air valve separates the collector side, which initially is filled with water, from the distribution side, which initially is filled with air.

Roughly, CHD processes a request for chemicals as follows: Initially, all the valves and flaps are closed, all the pumps are off. To compose the correct cocktail of chemicals, the PLC signals all the flaps and valves on the path from the tank of the first chemical to the requesting machine to open. Then it starts the pump on that path. When the flow meter signals that the requested amount has been measured, the pump stops and all the valves are signalled to close. The other chemicals are dosed in the same way, with certain amounts of water between them.

After the last chemical has been dosed, water is added to push the cocktail past the flap into the distribution part of the network. Then the flap is closed, the valves and flaps between air valve and requesting ma-

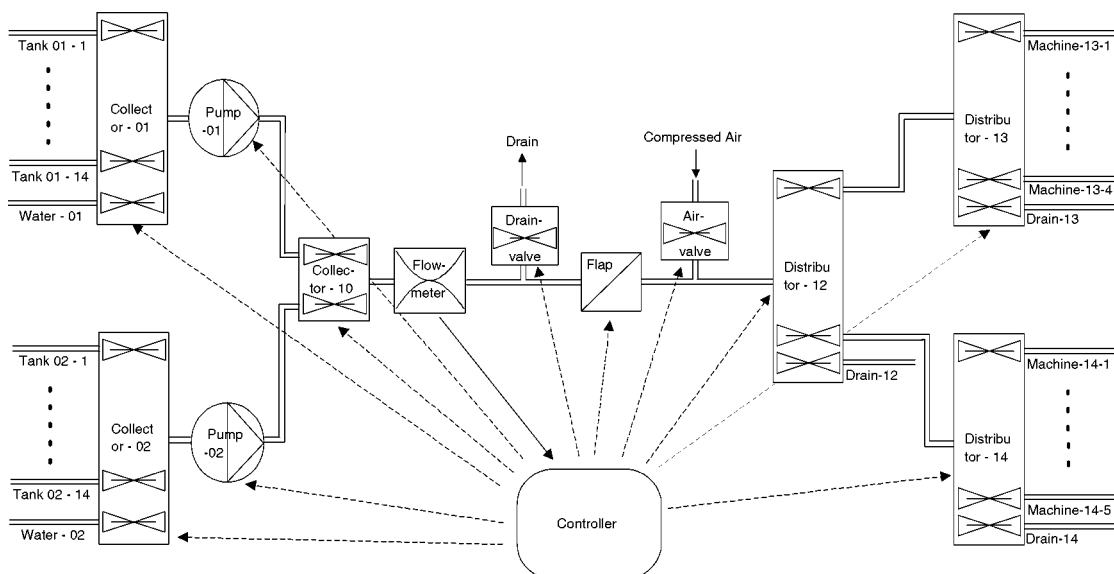


Fig. 18. Structure of the CHD.

chine are opened and compressed air is used to blow the cocktail to the machine.

Finally, a cocktail of water from all the segments of the collector network involved is composed in the same way and transported to the drain valve close to the requesting machine.

4.3. Model based diagnosis of the chemical distributor with CHD-DIAS

According to THEN, the very rare faults of the chemical distributor are typically valves stuck at closed or pumps that do not work. In that case an operator may activate a program like our post mortem diagnostic system CHD-DIAS (**C**hemical **D**istributor-**D**iagnostic **A**ssistant). CHD-DIAS is a model based system in the tradition of GDE [8], which analyses inconsistencies between observations and device models.

Initially, CHD-DIAS compares device models with histories of the control signals to pumps and valves as well as the history of sensor signals from the flow meter. In principle, these can be observed by the programmable logic controller (PLC) and reported to a logging facility. In practice, this proved impossible, since THEN was reluctant to modify the PLC-program controlling a real dyehouse to add a logging facility. This points at a typical problem in process industry: frequently suppliers of equipment do not have their own test sites but have to test innovations at the sites of certain pilot customers. This also supports our claim to integrate development of diagnostic functions with

development of the device itself, since it involves considerably less risks than adding new functionality to an already running system.

From the histories of the control signals CHD-DIAS can deduce if all components on a path through CHD should have enabled flow of chemicals. Using the structural model of CHD, this information is passed on to the model of the flowmeter. If this sensor does not detect any flow, this contradiction can be exploited for diagnosis: CHD-DIAS will suspect every component on the path. To discriminate between them it will ask the dyehouse staff for observations about these components.

Using the approach presented above CHD-DIAS is able to localize the typical faults of the chemical distributor. Nevertheless, CHD-DIAS is still a prototype. To turn it into a product, one has to improve the user interface and integrate it more closely with monitoring and control systems in the dyehouse. Furthermore, the time required for diagnosis has to be decreased considerably.

4.4. To reduce introduction costs

Fraunhofer IITB has used the process of developing CHD-DIAS as a guiding example to our search for ways to reduce the costs for introducing machine or plant building companies to diagnostic technology. The resulting approach seems to be applicable at least to diagnosis of devices whose behaviour is described by domain experts using (extended) state transition di-

agrams. Its main elements, which will be demonstrated in a second, are:

1. We start from existing state transition diagrams describing the behaviour of the relevant device components or processes. If no such diagrams exist for certain components or processes, domain experts may use a commercial CAE-tool like StateMate to design such diagrams.
2. The diagrams are checked, and experts are asked to provide certain missing information about states or transitions, which may be useful or necessary for diagnosis. The heuristics which enable such pointed questions are a central result of our work. Checking diagrams and questioning experts should be done automatically in the future.
3. We import the diagrams into the respective diagnostic engine. To this end we export the diagrams from a commercial tool like StateMate into a new intermediate language called AML. AML is tailored to the representation of extended state transition diagrams. Another program called SCI will then translate the AML-representation into the formalism understood by our diagnostic engine MuDia. By this two step approach we minimize double work in translating models from different commercial discrete event modelling tools into our own formalism.
4. The models are used in standard consistency based diagnosis.

Thus, domain experts can specify much of the knowledge required for diagnosis without relying on expensive diagnostic experts. In the following we will demonstrate this approach while developing a diagnostic model for the behaviour of a valve.

Initial diagram

We start with the straightforward state transition diagram shown in Figure 19. The events `doOpen` and `doClose` correspond to the positive flanks of the control signals from the PLC. Since we cannot observe if a particular valve is open or closed (at least not without recurring to the operator), the model specifies that states `OPENING` and `CLOSING` will have been left after a certain time signalled by some timeout event. The behaviour of pumps, flaps and even blocks of valves may be described in a similar way: by specifying a flow blocking state, a flow enabling state and two intermediate states.

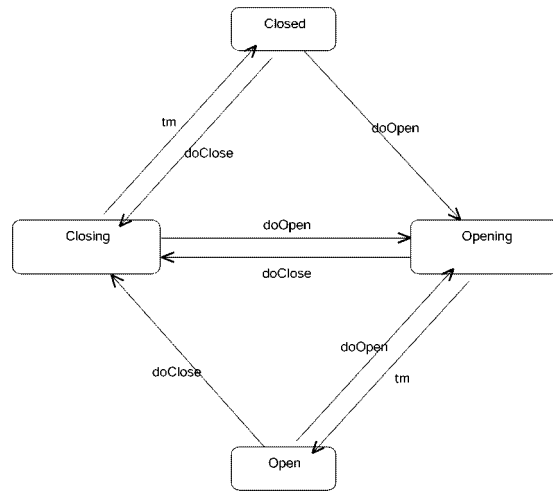


Fig. 19. State Transition Diagram describing the behavior of a valve.

Augment the diagram

The diagram from Fig. 19 is not very helpful with respect to diagnosis, since it does not allow for any predictions that could be compared to observations. We have developed a set of guidelines to help domain experts in augmenting the model:

1. For every state try to identify *invariants* which are refutable by considering the available signal histories. An invariant is a feature which does not change throughout the state. In our diagram we can state that flow is enabled throughout `OPEN` and disabled throughout `CLOSED`. However, there do not seem to be any invariants for `OPENING` or `CLOSING`. In particular, flow may be disabled at the beginning of `OPENING` and enabled towards the end.
2. Identify *intermediate states*, i.e., states during which the operation mode of the device changes qualitatively. In our example, `OPENING` and `CLOSING` are obviously intermediate states while `OPEN` and `CLOSED` are stable.
3. For every intermediate state try to identify refutable *post conditions* that indicate success of the mode transition. (Again, ‘refutable’ means ‘refutable by considering the available signal histories’.) In our example the post conditions coincide with the invariants of the successor states: flow has to be enabled after `OPENING` and it has to be disabled after `CLOSING`.

- For every intermediate state try to identify refutable *preconditions*, i.e., conditions that necessarily have to be satisfied to successfully complete the corresponding operation mode transition.

There are of course a lot of preconditions for successfully opening or closing a valve; but none of them can be refuted by considering the available signal histories. However, if there were such conditions and if some of them are not met, then a failing mode transition may be due to the same failure that invalidated the precondition.

- For every intermediate state try to identify refutable *conditions sufficient for success* of the corresponding mode transition.

Again, there do not seem to be any such conditions for opening or closing the valve. However, if there were such conditions we could use them to prove that some failure is not an effect of some previous failure by proving that at least one sufficient condition was satisfied.

- For every intermediate state try to identify *lower or upper bounds for the duration* of the corresponding mode transition.

In particular, the upper bound for the duration can be combined e.g., with the post conditions of an intermediate state to refute that the corresponding mode transition has finished in time. Fortunately, domain experts can very often give such bounds. In our example, a valve needs no longer than 1 second to open and 0.5 s to close.

- Try to specify refutable *operation conditions* of the device, i.e., conditions on the interaction of the device with its environment. For instance specify conditions on the frequency of external events or whether certain events are not allowed in certain states.

If some operation condition is not satisfied, one might hypothesize a fault in the past of the system or its environment rather than a fault in the current state. However, one has to be careful to meet the ‘No structure in function’ principle: operation conditions have to hold in every environment if the device is supposed to behave as intended. Therefore, we will not require that `doClose` is only allowed in state `OPEN` or that `doOpen` is only allowed in state `CLOSED`:

One may ask whether specification of invariants, preconditions and post conditions of a mode transition does not suffer from the notorious frame problem, qualification problem and ramification problem, resp. But these problems do not seem to be significant in this context, since we require conditions to be refutable

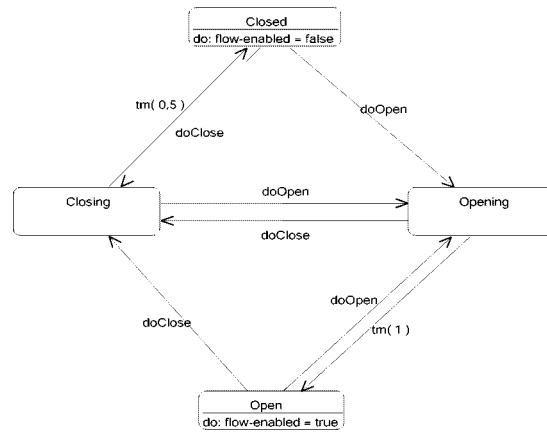


Fig. 20. Augmented State Transition Diagram describing the behavior of a valve for diagnosis.

from the limited set of available signal histories. Following these guidelines to augment the valve model of Fig. 19 yields Fig. 20.

In [14] we have argued that these heuristics may be transferred to flow diagrams in general. Such diagrams are used by experts in many domains, e.g., diagrams of material or energy flow in process industry [9], function structures in engineering design [24], function graphs in plant automation [1] or phase models to describe production processes [4]. This indicates that developing ‘wizards’ which rely on these heuristics to guide domain experts in modelling for diagnosis is indeed a promising approach.

Import the state transition diagram into the diagnostic engine

Domain experts should use their favourite discrete modelling tool, e.g., StateMate or StateFlow, to specify state transition diagrams. However, these tools will usually not provide export filters to languages understood by today’s diagnostic engines. In particular, most diagnostic engines require that implicit assumptions like Persistence by Default (‘if event e arrives in state s , but there is no transition labelled with e from s whose guard conditions are satisfied, then the system stays in s ’) are specified explicitly. Our diagnostic tool MuDia is no exception. Hence we have to translate the state transition diagram and explicate such assumptions on the way.

To minimize double work in developing translators from different discrete event modelling tools into our own formalism DEEP, we use a two step approach. In the first step we translate the representation con-

```

ENTITY valve-ok OF valve;
  ATTRIBUTES
    flow-enabled: boolean;
    doOpen: event;
    doClose: event;
  STATES
    closed: do / flow-enabled = false;
    opening;
    open: do / flow-enabled = true;
    closing;
  TRANSITIONS
    FROM closed TO opening
      doOpen;
    FROM opening TO closing
      doClose;
    FROM opening TO open
      tm(entry(opening),1);
    FROM open TO closing
      doClose;
    FROM closing TO closed
      tm(entry(closing),0.5);
    FROM closing TO opening
      doOpen;
END-ENTITY

```

Fig. 21. AML-representation of the augmented state transition diagram.

structured by the commercial tool into a new intermediate language called AML. [3] describes AML in more detail. Figure 21 shows the AML-representation of the diagram from Fig. 20.

In the second step, a program called SCI (State Chart Integrator) translates AML-representations into DEEP-representations. A first version of SCI has been described in [3]. The current version described in [14] eliminates a couple of serious shortcomings.

From the AML-representation of a state transition diagram SCL generates five types of implication formulae in a temporal logic language:

- *Transition rules* describe the preconditions and effects of transitions.
- *Intra-state action rules* describe the actions which are processed whenever entering or leaving a state or reacting to an event without leaving the current state.
- *Activity rules* describe invariants of a state and activities that proceed as long as the state is active. They correspond to traditional intra-state constraints.
- *Definition rules* define certain types of events, like timeouts or changing values of system variables.

- *Explanation closure rules* describe under which circumstances state and variable values remain unchanged.

Unfortunately there is not enough space to give examples for these rules or the DEEP-representation of the valve model. The reader should look at [14] for the former and contact the author for the latter.

4.5. Discussion

The development of CHD-DIAS has shown again: while today's diagnostic systems are powerful enough to provide real value to operators of complex technical devices, high costs prevent the routine development of such systems. In this chapter we have sketched an approach to reduce these costs in domains, where experts model behaviour with state transition diagrams. The approach strives at enabling domain experts to construct diagnostic models in the environment they are used to, thus minimizing the need for costly external experts.

Other approaches like [2,6,7,17,28,29] do also use state transition diagrams for model based diagnosis. It is still unclear how they relate to our approach. There are also interconnections to systematic requirements engineering and to FMEA which need to be further investigated.

Diagnosis based on (extended) state transition diagrams like statecharts should gain relevance in the future, since they are increasingly used in industry to completely or partially model so called reactive systems. This class encompasses e.g., many embedded systems or control systems [15]. Moreover, the approach does also seem to be applicable to flow diagrams in general, as argued in [14].

During the last years the idea of identifying sets of more or less standardized modules for implementation, called 'well known solutions', 'basic steps' or 'pattern', resp. has come up in different disciplines (cf [10, 12,16,24]). Founding diagnosis on libraries with diagnostic models for these standard modules would not only further reduce development costs but present another step towards the integration of diagnostic development into device development.

5. Summary

The INDIA project was an attempt to promote a major step in the transfer of model-based systems into industrial applications. This required not only the fur-

ther development of modeling and model-based reasoning techniques, but also consideration of the context in which the systems were to be used as tools supporting the current work process. The solutions included approaches to

- exploiting existing engineering knowledge, as in the chemical distributor case study,
- generating diagnostic output in a form that is familiar to experts involved in the task and that can be processed by existing tools like fault trees in the fork lift application, and
- exporting results to a (new generation of a) standard tool like authoring systems for the automotive repair plans.

The case studies indicate that with the mature part of the model-based technology, it is possible to provide effective support and speed-up of current steps in the work processes. This applies particularly to those steps that require technical knowledge about the devices and plants, but apply in a very structured way under well-defined goals. But it is often this part that is time-consuming and limits the exploitation of expert knowledge for more creative and ‘non-standard’ tasks. Model-based systems could be shown to provide a solution to the problem of having to carry out the same task for a number of variants. Of course, the project also helped to reveal shortcomings and limitations of the current technology. We feel that much more work is needed to effectively support the generation of models for the various tasks and also to model the tasks to be supported as real WORK PROCESSES including their practical conditions and constraints in order to provide the appropriate tools.

Acknowledgments

We would like to thank our colleagues at IITB, LKI and TUM as well as our partners in INDIA for their valuable contributions. This work has been supported by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) under the grant 01 IN 509 D 0, INDIA.

References

- [1] W. Ahrens, H.-J. Scheurlen and G.-U. Spohr, *Informationsorientierte Leittechnik*, Oldenbourg-Verlag, 1997 (in German).
- [2] P. Baroni, G. Lamperti, P. Pogliano and M. Zanella, Diagnosis of large active systems, *Artificial Intelligence* **110** (1999), 135–183.
- [3] A. Brinkop, Statecharts zur Simulation und Diagnose. Fraunhofer-Institut für Informations- und Datenverarbeitung IITB, Karlsruhe, INDIA-Report 99-01, 1999 (in German).
- [4] H. Buchner, J. Lauber and M. Polke, Das Informationsmodell: Basis für die interdisziplinäre Prozeßbeschreibung, *AT – Automatisierungstechnik* **42**(1) (1994), 5–10 (in German).
- [5] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano and D. Theseider-Dupré, Strategies for on-board diagnostics of dynamic automotive systems using qualitative models, *AI Communications* **12**(1), 33–44.
- [6] Y.L. Chen and G. Provan, Modeling and diagnosis of timed discrete event systems – a factory automation example, in: *Proc. of the American Control Conference*, Albuquerque, New Mexico (USA), 1997, pp. 31–36.
- [7] M.-O. Cordier and S. Thiébaux, Event-Based Diagnosis for Evolvable Systems, Internal Report 819, IRISA, May 1994.
- [8] J. de Kleer and B. Williams, Diagnosing multiple faults, *Artificial Intelligence* **32** (1987), 97–130.
- [9] DIN Deutsches Institut für Normung e.V. (Hrsg.). DIN-28004, Teil 1-4: Fließbilder verfahrenstechnischer Anlagen. Beuth-Verlag, Berlin, 1988 (in German).
- [10] B.P. Douglass, *Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*, Addison-Wesley, 1999.
- [11] P.-P. Faure, L. Trave-Massuyes and H. Poulard, An interval model-based approach for optimal diagnosis tree generation, in: *Proc. DX-99, 10th International Workshop on Principles of Diagnosis*, 1999.
- [12] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [13] T. Guckenbiehl, H. Milde, B. Neumann and P. Struss, Meeting re-use requirements of real-life diagnosis applications, in: *Proc. XPS-99: Knowledge-Based Systems*, Springer, LNAI 1570, 1999, pp. 90–100.
- [14] T. Guckenbiehl, Ansätze zur Senkung der Einführungskosten von Diagnosesystemen, in: *INDIA – Intelligente Diagnose in der Anwendung*, P. Struss, L. Hotz and T. Guckenbiehl, eds, Ch. III.4, Shaker-Verlag, 2000 (in German).
- [15] D. Harel and M. Politi, *Modelling Reactive Systems with Statecharts*, McGraw-Hill, 1998.
- [16] W. Hemming, *Verfahrenstechnik*, Vogel-Verlag, 1993 (in German).
- [17] J. Lunze and J. Schröder, Process diagnosis based on a discrete-event description, *At – Automatisierungstechnische Praxis* **47**(8) (1999), 358–365.
- [18] R. Inderst, Automatische Testgenerierung auf der Basis einer qualitativen Modellierung physikalischer Systeme, Diploma Thesis, Technical University of Munich, Dept. of Comp. Sci., 1995 (in German).
- [19] R. Inderst, P. Struss and O. Dressler, Automatische Testgenerierung für Weichenschaltungen auf der Basis qualitativer Modellierung, *Tagungsband zum Treffen der GI-Fachgruppe 1.5.2 “Diagnostik und Klassifikation”*, Goslar, 1995 (in German).

- [20] J. Mauss and B. Neumann, Qualitative reasoning about electrical circuits using series-parallel-star trees, in: *Proc. QR '96, 10th International Workshop on Qualitative Reasoning about Physical Systems*, 1996.
- [21] H. Milde, L. Hotz, J. Kahl and S. Wessel, Qualitative analysis of electrical circuits for computer-based diagnostic decision tree generation, in: *Proc. QR '99, 13th International Workshop on Qualitative Reasoning about Physical Systems*, and in: *Proc. DX-99, 10th International Workshop on Principles of Diagnosis*, 1999.
- [22] H. Milde and L. Hotz, Facing diagnosis reality – model-based fault tree generation in industrial application, in: *Proc. DX-00, 11th International Workshop on Principles of Diagnosis*, 2000.
- [23] www.occm.de.
- [24] G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach*, Springer, 1996.
- [25] C. Price and D. Pugh, Interpreting simulation with functional labels, in: *Proc. QR '96, 10th International Workshop on Qualitative Reasoning about Physical Systems*, 1996.
- [26] D. Pugh and N. Snooke, Dynamic analysis of qualitative circuits, in: *Proc. of Annual Reliability and Maintainability Symposium*, IEEE Press, 1996, pp. 37–42.
- [27] S. Russel and P. Norvig, *Artificial Intelligence a Modern Approach*, Prentice Hall, 1995.
- [28] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D.C. Teneketzis, Failure diagnosis using discrete-event models, *IEEE Transactions on Control Systems Technology* **4**(2) (1996), 105–124.
- [29] M. Sampath, S. Lafortune and D.C. Teneketzis, Active diagnosis of discrete-event systems, *IEEE Transactions on Automatic Control* **40**(7) (1998), 908–929.
- [30] P. Struss, Problems of interval-based qualitative reasoning, in: *Readings in Qualitative Reasoning about Physical Systems*, D. Weld and J. de Kleer, eds, Morgan Kaufmann, 1990.
- [31] P. Struss, Testing physical systems, in: *Proc. AAAI-94*, Seattle, USA, 1994.
- [32] P. Struss, A Theory of Testing Physical Systems Based on First Principles, Technical Report CD-TR 94/63 of the Christian Doppler Laboratory for Expert Systems, Vienna University of Technology, 1994.