Facing Diagnosis Reality - Model-Based Fault Tree Generation in Industrial Application

Heiko Milde and Lothar Hotz

Laboratory for Artificial Intelligence, University of Hamburg
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany
milde@kogs.informatik.uni-hamburg.de
phone: ++49 (0)40 42883-2606, fax: ++49 (0)40 42883-2572

Abstract

Computer diagnosis systems grounded on hand-crafted fault trees are wide-spread in industrial practice. Since the complexity of technical systems increases and innovation cycles get shorter, the need for systematic fault tree generation and maintenance arises. In this paper, the MAD system is introduced which generates fault trees based on models of technical devices. Since a fault tree can be generated automatically based on a device model, the cost for providing, modifying, and maintaining diagnosis equipment can be drastically reduced and quality management of diagnosis equipment can be facilitated. Furthermore, the cost of fault-tree-based fault identification can be reduced because model-generated fault trees can be optimized. MAD is specifically designed to facilitate its integration into existing service workflow. We have successfully evaluated the MAD system in cooperation with the German forklift manufacturer STILL GmbH Hamburg.

1 Introduction

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, service technicians attach a computer diagnosis system to a forklift. Then the diagnosis system performs automated testing and it 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, the central service division manually generates or adapts fault trees. Dealing with this task, service engineers apply detailed expert knowledge concerning faults and their effects. Adapting fault trees to new model ranges can take a service engineer several months.

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. The STILL GmbH defined the following requirements for the introduction of a new computer diagnosis system.

- First, cost of new diagnosis system integration into existing STILL service and diagnosis workflow has to be low. A new diagnosis system must be easily accepted by service technicians and service engineers. Thus, identifying faults with a new diagnosis tool has to be intuitive for service technicians which are familiar with the current STILL diagnosis system. Long terms of training for service technicians and service engineers are not acceptable.
- Second, cost of diagnosis equipment generation, modification, and maintenance has to be reduced. Generating diagnosis equipment for new forklift model ranges as well as adapting diagnosis equipment to forklift updates has to be carried out cost-efficient.
- Third, the current diagnostic performance should be exceeded. As a minimal requirement, a new diagnosis system has to be able to identify all faults handled by the current STILL system. Incorrect fault identifications cannot be accepted.
- Fourth, average fault identification cost has to be reduced.

As a participant of the German joint research project INDIA (Intelligent Diagnosis in Industrial Applications) the Laboratory for Artificial Intelligence of the University of Hamburg develops concepts of innovative computer diagnosis techniques for the STILL application scenario. Facing the above-described requirements for the introduction of new diagnosis techniques, we consider innovative model-based techniques to be advantageous because they provide a systematic way for design, modification, and optimization of diagnosis equipment. In particular, automatically generating fault trees from device models is a promising strategy because completely replacing fault trees is not an immediate option for economical reasons. In our application, two major requirements for successful model-based fault tree generation arise.

- First, to secure acceptance among service technicians, model-based fault trees have to represent established STILL fault identification strategies. That is, fault identification guided by automatically generated fault trees should be similar to current STILL fault identification. Thus, modeling circuit faults and representing current fault identifying actions is essential as well as accurate symptom prediction.
- Second, to secure acceptable cost of fault tree generation and maintenance the complexity of the device modeling process has to be minimized. Thus, massive utilization of model libraries and extensive reuse of

existing diagnosis system components and integration of available resources such as design data, service expertise, and expert knowledge from the design process is essential.

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 (Cascio et al. 99) and (Faure et al. 99), 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 Figure 1). 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, behavior predictions are automatically computed from the device model and 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 preferable fault tree topologies. In order to realize these concepts, we implemented the MAD system which is described in this paper.

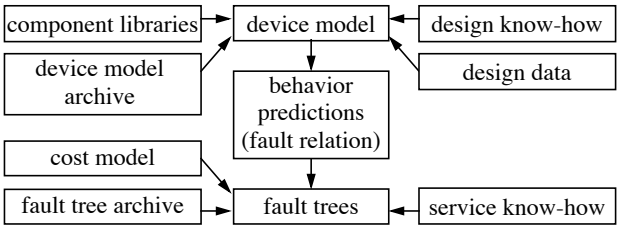

Figure 1. Basic concepts of model-based fault tree generation

This paper is structured as follows: In Section 2, we introduce the accelerator pedal circuit which is a simple example showing typical characteristics of diagnosis in our application. In the remainder of the paper, model-based fault tree generation is described for this circuit. Section 3 presents circuit modeling. Automated fault tree generation is described in Section 4. Finally, Section 5 summarizes our findings including evaluation and conclusions. In this paper, we outline how STILL diagnosis scenarios and fault identification strategies can be handled more than presenting theoretical background.

2 The Accelerator Pedal Circuit

In this section, we describe the forklift accelerator pedal circuit and enumerate faults which can be handled by the current STILL computer diagnosis system. Furthermore, we demonstrate how fault identification is currently performed. Although considering a certain forklift circuit, faults, symptoms, and diagnosis strategies seem to be representative for a wide range of applications.

Figure 2 shows the wiring diagram of the accelerator pedal circuit which enables the 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.

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 9.8V and 10.1V. Additionally, the mechanical connection between accelerator pedal and potentiometer 1B1, may not be adjusted correctly or may even be broken.

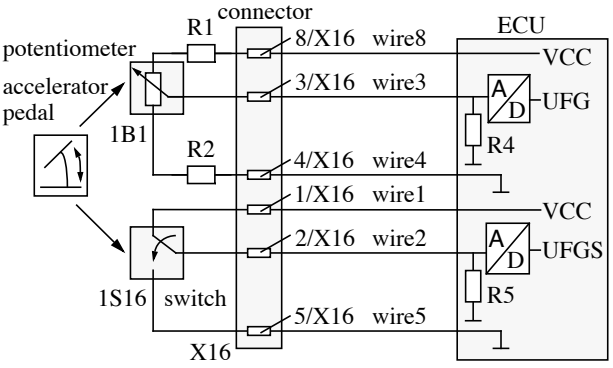

Figure 2. Wiring diagram of a forklift accelerator pedal circuit

For fault identification, the current diagnosis system utilizes *measurements* and direct component *fault identifications*. Additionally, certain *operating modes* are effectuated to simplify fault identification.

- **Measurements.** Current and voltage are automatically or manually measured in steady states of electrical circuits only. Automatically measured parameter values are frequently interpreted by the ECU such that qualitative information is exploited for diagnosis whereas the quantitative values may not be utilized at all. In the accelerator pedal example, the ECU automatically measures UFG and UFGS. 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*. Considering these value names, one could think that these values describe the switch 1S16 but, in fact, they represent certain quantitative value ranges of UFGS. In addition to automated measurements, service technicians manually measure the voltage drop from wire 8 at connector X16 to ground.
- **Fault identification.** Some components can be removed from the assembly in order to connect them to test devices which non-ambiguously identify correct and faulty component behavior. For example, the breaking of some wires of the accelerator pedal circuit

is directly verified by attaching these wires to a test device which measures the conductance of these cables.

- **Operating modes.** Technical systems usually show different modes of operation. Intendedly installing certain operating modes can be helpful for fault identification. In the accelerator pedal example the pedal can be kicked down and released again which defines two different operating modes. The current diagnosis system explicitly exploits these two operating modes for diagnosis. To facilitate fault identification certain operating modes may be installed which never occur in regular operation of a device. These operating modes are exclusively effectuated for diagnostic purposes. For example, connectors can be opened and short circuits (bridges) can be inserted into electrical systems. Components can be replaced assuming that their substitutes behave correctly. For diagnosis of the accelerator pedal circuit, the current STILL diagnosis system may instruct the service technician to open connector X16. After opening this connector, in some cases, there is a bridge inserted between wire 1 and wire 2. This bridge is a substitute for switch 1S16 in the state shown in Figure 2.

The accelerator pedal 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. In Section 3 of this paper, we demonstrate how circuit faults, fault identifying actions, and diagnosis strategies of our application can be represented using MAD.

3 Electrical Circuit Analysis

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. Qualitative models rather represent fundamental system behavior than sharp quantitative parameter values and a single qualitative fault model can cover a wide range of faulty component behavior. Thus, a limited number of qualitative fault models can suffice for extensive faulty behavior representation. Therefore, in principle, qualitative circuit modeling is useful for model-based fault tree generation. In our application, qualitative electrical device models are adequate because, in current 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 Section 3.1, MAD's qualitative network analysis is briefly presented. Details of MAD's internal models of electrical circuits and the computation of qualitative parameter values can be found in (Milde et al. 99).

3.1 MAD's Internal Electrical Circuit Models

As known from electrical engineering, MAD represents electrical circuits by equivalent networks. These networks consist of standard component models which show no internal structure but they show well-defined and idealized behavior. Controlled versions of standard component models exist. MAD only provides two different types of standard component models, i.e. passive and active models showing passive and active behavior modes, respectively. Passive behavior modes are "consumer", "insulator", and "conductor". Active models qualitatively represent idealized voltage sources providing different voltage levels. Physical parameters are described by qualitative parameter values standing for intervals or landmarks. To facilitate the analysis of faulty device behavior, actual values, reference values, and deviations of parameters are explicitly represented.

3.2 COMEDI (COmponent Modeling EDItor)

To improve acceptance among engineers, MAD provides a user interface called COMEDI which is similar to a CAD tool. For device modeling, COMEDI offers three different model libraries shown in Figure 3, i.e. the generic component library, the application component library, and the device model archive.

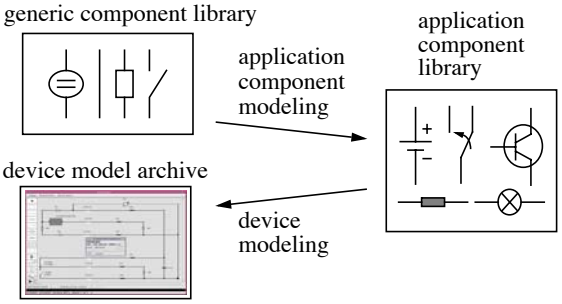

Figure 3. COMEDI libraries and the device modeling process

- The generic component library contains all elementary component class models that are provided by MAD's internal representation of electrical circuits. Thus, the generic component library cannot be extended. It consists of 728 different component class models.
- The application component library contains models of component classes that are explicitly designed for certain applications. These component class models are interactively generated, based on generic component class models. Thus, the application component library can be extended.
- 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.

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. As shown in Figure 3, COMEDI device models are assemblies of application component class models which are based on generic component class models. Hence, to prepare the description of circuit modeling, in the following, we outline COMEDI's generic component class models and the hierarchical structure of the generic component library which is shown in Figure 4.

Each of COMEDI's generic component class models shows a set of basic behavior modes which are briefly presented in Section 3.1. These behavior modes are possible actual behaviors, i.e. each of these behavior modes can be chosen to model actual component behavior (correct and faulty). Additionally, each generic component class model shows one behavior mode describing component reference behavior. Thus, a generic component class model comprises a selection of actual behavior modes and exactly one reference behavior mode. Modeling a certain actual behavior means selecting one behavior mode from the set of possible actual behaviors. When the reference and the actual behavior mode are identical, usually, correct component behavior is modeled.

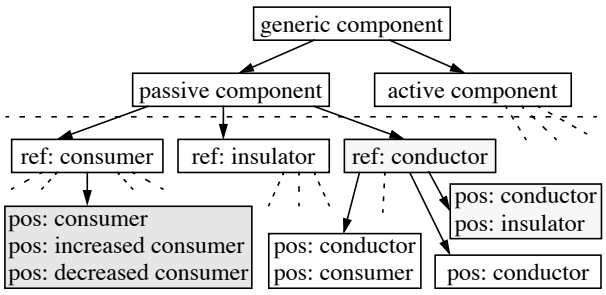

Figure 4. Selected parts of the hierarchical structure of COMEDI's generic component library

The internal representation of the generic component library is hierarchical such that the most general model is the root of a tree and the most specific models can be found in the leaves (see Figure 4). The generic component model (the root) consists of two sets of behavior modes, i.e. possible and reference behavior modes. Both sets contain all basic behavior modes described in Section 3.1, both active and passive behavior modes. In the next lower level in Figure 4, passive and active component models also show sets of possible and reference behavior modes but these sets contain only active and passive behavior modes, respectively. For each model in the next lower level in Figure 4, the set of reference behavior modes is restricted to exactly one reference (ref) behavior mode. Models found in the leaves additionally restrict the number of possible (pos) behavior modes. Since generic component class models show exactly one reference behavior mode, for application component class modeling, only generic component models from the two lowest levels in Figure 4 can be utilized. Using COMEDI, modeling electrical circuits consists of application component modeling and device modeling (see Figure 3). Both tasks are described in the following two subsections.

3.3 Application Component Modeling

In this subsection, the interactive generation of models of application component classes is described. These models are based on generic component class models. We consider wire 4 of the accelerator pedal circuit as an illustrating example.

1. **Operating mode modeling.** In the first step, the *electrical* behavior of an application component class is modeled. Components may show different modes of operation. Switches, for example, can be open or closed. For each operating mode of a component class, a generic component class model has to be selected, i.e. reference and possible actual component behavior have to be modeled. Thus application component class models show one or more generic component class models.

   Reference behavior modeling. According to Figure 4, first of all, it is decided whether the model is active or passive. Then an adequate reference behavior mode is selected. Usually, this step is performed to determine correct component behavior. Obviously, wire 4 is a passive component. Its internal resistance is very small and negligible in the context of accelerator pedal circuit analysis. Hence, we determine the correct behavior of wire 4 by selecting the reference behavior mode "ref: conductor" which is shaded pale in Figure 4.

   Possible actual behavior modeling. In this step, all possible actual component behavior is modeled which is relevant for automated fault tree generation. That is, correct and faulty behavior has to be represented by choosing an available set of possible behavior modes. Available sets of possible behavior modes are represented by the node selected in the previous modeling step and its successors (see Figure 4). Correct behavior of wire 4 is already discussed above. If the wire is broken there is no current and the wire behaves like an insulator. Hence, for possible actual behavior modeling, we choose the "pos: conductor, pos: insulator" model from the generic component library (see Figure 4). Note that, for some reference behavior modes, there are corresponding deviative behavior modes offered for actual behavior modeling. For instance, if the reference behavior is "ref: consumer" then automatically "pos: increased consumer" and "pos: decreased consumer" are offered which is exemplarily shown in the dark shaded box in Figure 4. Internally, these behavior modes can be realized due to qualitative deviation values.
2. **Measurement modeling.** After the electrical behavior of a certain application component class is modeled, measurements can be defined. Both, the voltage drop across the generic component class model and the current through the component model can be subject of a measurement. As a consequence of modeling a certain measurement, there is a column for the parameter in the fault relation showing corresponding qualitative parameter values.
3. **Observation modeling.** Similar to *functional labeling* (Price and Pugh 96), user-defined strings such as "light bulb shining" can be attached to certain qualitative current and voltage values to model typical observations used for diagnosis. For each observation in a device model, there is a column in the fault relation. Note that, observations can be utilized to model the error flags in our application.
4. **Replaceability modeling.** As noted above, during

diagnosis sessions, components may be replaced by correct-working substitutes to simplify fault identification. If replaceability of an application component class is modeled, then, first, each generic behavior model describing a certain operating mode is duplicated. In a second step, for each duplicate, all possible behavior modes different from the reference behavior mode are removed. Third, the resulting generic behavior models are added to the application component class model. By this means, for each operating mode, there is a corresponding operating mode automatically modeled showing no faulty behavior.

5. **Fault identification modeling.** For each application component class model, fault identification can be interactively declared to be possible. For each fault identification in the device model, there is a column in the fault relation directly indicating whether a fault has occurred.

If complex components such as amplifying circuits and logical circuits have to be modeled, frequently, abstract behavior models may be required neglecting unnecessary structural and behavioral details. COMEDI explicitly supports the interactive generation of abstract application component class models. These models are based on controlled versions of generic standard component models, the definitions of external non-electric controlling parameters, user-defined qualitative values for controlled and controlling parameters, and causal dependencies between these parameters represented in behavior tables. Abstract application component models can also be stored in the application component library. Engineers can easily build these models exploiting their qualitative understanding of how certain components or subsystems work. Due to lack of space, we cannot elaborate on abstract component models.

Considering the accelerator pedal example, the application component class model of the potentiometer 1B1 is an abstract model showing a controlled voltage source to model the voltage at the middle of the potentiometer. This voltage causally depends on the voltage drop across the potentiometer and the position of the accelerator pedal. The pedal position is modeled as an external non-electric controlling parameter.

3.4 Device Modeling and Fault Relation Generation

For device modeling, MAD users assemble icons on the screen which represent models from the application component library. Device operating modes are interactively defined by certain combinations of component operating modes. Considerable fault combinations can also be defined. Automated behavior predictions are performed for all possible component behavior (correct and faulty) and all device operating modes considered in the device model. That is, for all device operating modes, the impact of faults on measurements, observations, and fault identification is computed. The results are stored in the fault relation. Note that, in MAD's fault relations, measurements, observations, and fault identifications are simply called "tests" because they are identically utilized for fault tree generation.

In Figure 5, the COMEDI device model of the accelerator pedal circuit is presented. As described in Section 3.3, all wires are modeled as idealized conductors that may break. Correct and faulty behavior of wire 4 is shown in the small window in the centre of Figure 5. Fault identifications exists for these wires which can be verified in the lowest section of the corresponding fault relation shown in Figure 6. Note that, results of fault identifications are independent from device operating modes.

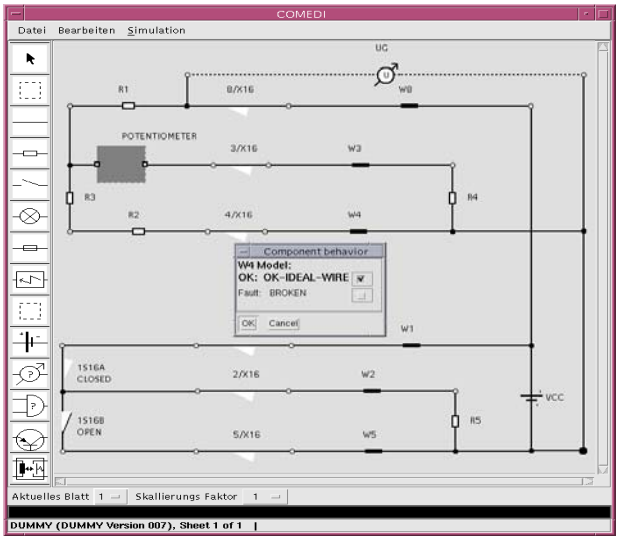

Figure 5. COMEDI model of accelerator pedal circuit

Fault Relation Frame

Behavior modes	Tests:	UFG_LOW_KICKDOWN	UG_KICKDOWN	UFG_KICKDOWN
1S16A-is_STUCK-AT-OPEN		OK	N_PMAX_US7	N_PB_US6
W5-is_BROKEN		OK	N_PMAX_US7	N_PB_US6
W2-is_BROKEN		OK	N_PMAX_US7	N_PB_US6
W1-is_BROKEN		OK	N_PMAX_US7	N_PB_US6
W8-is_BROKEN		NOT_OK	L_0_US7	L_0_US6
W4-is_BROKEN		OK	N_PMAX_US7	H_PB_US6
W3-is_BROKEN		NOT_OK	N_PMAX_US7	L_0_US6
PEDAL-is_DEFECT		NOT_OK	N_PMAX_US7	L_0_US6
PEDAL-is_NOT-ADJUSTED		NOT_OK	N_PMAX_US7	L_PB_US6
VCC-is_LOW-VOLTAGE		NOT_OK	L_PB_US7	L_PB_US6
VCC-is_HIGH-VOLTAGE		OK	H_PIMP_US7	H_PB_US6
VCC-is_NO-VOLTAGE		NOT_OK	L_0_US7	L_0_US6
OK-CASE		OK	N_PMAX_US7	N_PB_US6

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

Behavior modes	Tests:	W3	W4	W8	W1	W2	W5
1S16A-is_STUCK-AT-OPEN		NO	NO	NO	NO	NO	NO
W5-is_BROKEN		NO	NO	NO	NO	NO	BROKEN
W2-is_BROKEN		NO	NO	NO	NO	BROKEN	NO
W1-is_BROKEN		NO	NO	NO	BROKEN	NO	NO
W8-is_BROKEN		NO	NO	BROKEN	NO	NO	NO
W4-is_BROKEN		NO	BROKEN	NO	NO	NO	NO
W3-is_BROKEN		BROKEN	NO	NO	NO	NO	NO
PEDAL-is_DEFECT		NO	NO	NO	NO	NO	NO
PEDAL-is_NOT-ADJUSTED		NO	NO	NO	NO	NO	NO
VCC-is_LOW-VOLTAGE		NO	NO	NO	NO	NO	NO
VCC-is_HIGH-VOLTAGE		NO	NO	NO	NO	NO	NO
VCC-is_NO-VOLTAGE		NO	NO	NO	NO	NO	NO
OK-CASE		NO	NO	NO	NO	NO	NO

Figure 6. Selected parts of fault relation of accelerator pedal circuit

Both resistors R4 and R5 show "insulator" behavior, no fault modes, and observations to model the measurements UFG_LOW, UFG_HIGH, and UFGS, respectively. Additionally, R4 shows a voltage measurement to model UFG (see fault relation in Figure 6). The manual voltage measurement UG described in Section 2 is modeled by a special multimeter component model (see Figure 5).There are two device operating modes defined for the accelerator circuit model, i.e. "STANDARD" and "KICKDOWN". "STANDARD" indicates that the accelerator pedal is not kicked down. Hence, switch 1S16 is in the operating mode shown in Figure 2 and the potentiometer is in a middle position. If the pedal is kicked down, operating mode "KICKDOWN" is reached. In this case, the switch has changed its operating mode and the potentiometer has reached an upper position.

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.

- **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.
- **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.

Since reduction of average fault identification cost is one of the basic requirements for the introduction of innovative computer diagnosis techniques in our application, in the following, we describe cost-optimized fault tree generation.

4.1 Modeling Costs of Tests

A widely used approach for fault tree generation is to apply the entropy criteria (Quinlan 86), (Struss 94), (Mauss and Neumann 96), (Price et al. 96). For cost-optimized fault tree generation, considering entropy only is not sufficient. Additionally, cost of tests and dependencies between them have to be taken into account. MAD enables users to define test models including cost models. As a basis of these models, an ordered sequence of basic working tasks can be introduced for each test. For example, these tasks have to be performed to obtain a value of voltage UG of the accelerator pedal example. Examples for basic working tasks are "open the front bonnet", "separate the battery", "pull plug P1". MAD users can specify costs of basic working task. Sequences of basic working tasks implicitly define dependencies between tests. Due to space limitations, we omit the description of processing these dependencies.

The cost *C(T)* of a test *T* is defined as the sum of costs of the basic working tasks to be performed for *T*. Costs of tests of the accelerator pedal circuit are given in Table 1. For simplicity, costs of operating mode changes and basic working tasks are not explicitly modeled in this example. Note that, measurements can be performed in different operating modes (KICKDOWN and STANDARD). Results of fault identifications (e.g. w1) are independent from operating modes. Fault identifications are the most expensive available tests.

test / operating mode	cost	test	cost
UFG_LOW / KICKDOWN	6	w1	10
UG / KICKDOWN	8	w2	10
UFG / KICKDOWN	7	w3	10
UFG_HIGH / KICKDOWN	1	w4	10
UFGS / KICKDOWN	1	w5	10
UFG_LOW / STANDARD	5	w8	10
UG / STANDARD	6		
UFG / STANDARD	9		
UFG_HIGH / STANDARD	1		
UFGS / STANDARD	1		

Table 1: Costs of tests of the accelerator pedal circuit

4.2 Cost-Optimized Fault Tree Generation

In an optimal fault tree, the average fault identification cost is minimal. This optimal solution can be computed by the algorithm GFT^{A*} (Generating Fault Trees with A*) which is described in this section. GFT^{A*} is an application of the well-known search algorithm A*. GFT^{A*} does not take fault probabilities into account because, in our application, fault probabilities are not available. A promising approach for fault tree generation considering fault probabilities is presented in (Faure et al. 99).

The basis for the generation of a fault tree is the fault relation (see Figure 6). The fault tree generation starts with combining all faults found in the fault relation to one fault set called root set. This fault set is split into subsets with respect to a certain test. This splitting is recursively repeated for every fault set as long as there are faults in the fault set that can be discriminated by some difference between their test values.

A test T_i which partitions a fault set into subsets is a simple equation of the form $T_i = v$. v is a value of the domain of T_i. In our case, for example, v is a qualitative value of a finite set qualitatively describing the result of a measurement. In general, test are not exclusive, i.e. if a test partitions a fault set into subsets, certain faults can occur in more than one subset. In the best case, all leaves of the fault tree only contain a single fault, i.e., all faults can be discriminated by tests given in the fault relation.

Optimal search. The GFT^{A*}-algorithm computes a fault tree with minimal average fault identification cost. To apply A*, a search model must be given by specifying the definition of a state, a start state, a goal state, a successor

function, a cost function *g* to evaluate a state, and a heuristic function *h* estimating the costs between a state and the goal state. For details of the A*-algorithm see (Russel and Norvig 95).

The GFT^{A*}-algorithm 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. Two states are equal if they consist of the same fault sets. A successor of a state is generated by partitioning one leaf fault set into at least two subsets by selecting a partitioning test. 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.

Each state is evaluated by the functions *g* and *h*. *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. In Figure 7, the definitions of *g()* and *h()* are given.

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. *kmax* 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 tests are different. 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 *kmax* 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 $\lfloor \log_{kmax}(|b|) \rfloor$, ($\lfloor \ldots \rfloor$ 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:

$$\sum_{i=1}^{\lfloor \log_{kmax}(|b|) \rfloor} C(T_i), \quad \textit{with } T_i \textit{ is i-th cheapest unused Test } T$$

g monotonically increases for successive states because an additional measurement is selected to determine a successor state. *h* ensures that the real costs necessary for reaching the goal are never overestimated. Besides this evaluation of states, GFT^{A*}, because based on A*, stores alternative and currently not best states. Thus, GFT^{A*} can reactivate the most cost-effective alternative state *s* if a selected path yields to states being more expensive than *s*. This yields to an optimal fault tree.

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

$$h(state) = \sum_{b \in leaves(state)} h(b), \quad \textit{with}$$

$$h(b) = |b| \cdot \sum_{i=1}^{\lfloor \log_{kmax}(|b|) \rfloor} C(T_i), \quad \textit{with } T_i \textit{ is i-th cheapest unused Test } T$$

Figure 7. Cost function g and heuristic function h

Figure 8 shows a fault tree of the accelerator pedal circuit generated by GFT^{A*}. The fault tree is based on the fault relation shown in Figure 6 and the test model shown in Table 1. Note that, in Figure 8, the final row of a fault set is the corresponding test. The root contains 13 faults which are not explicitly shown. The average fault identification cost of this tree is 17.

MAD's model-based behavior prediction and automated fault tree generation guarantee that fault trees are correct and complete with respect to the underlying device model and the faults and fault combinations considered in the fault relation. 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. These properties are essential for industrial utilization of MAD.

5 Evaluation and Conclusions

Investigating our application, we figured out that the following challenges arise when innovative diagnosis techniques are applied to industrial applications. First, innovative diagnosis concepts have to be incorporated into existing diagnosis equipment. Replacing existing diagnosis systems is not acceptable. Second, the process of diagnosis equipment generation, modification, and maintenance has to allow the integration of existing resources such as expert knowledge as well as existing diagnosis equipment. Third, the diagnostic performance of the service division should be exceeded and fault identification

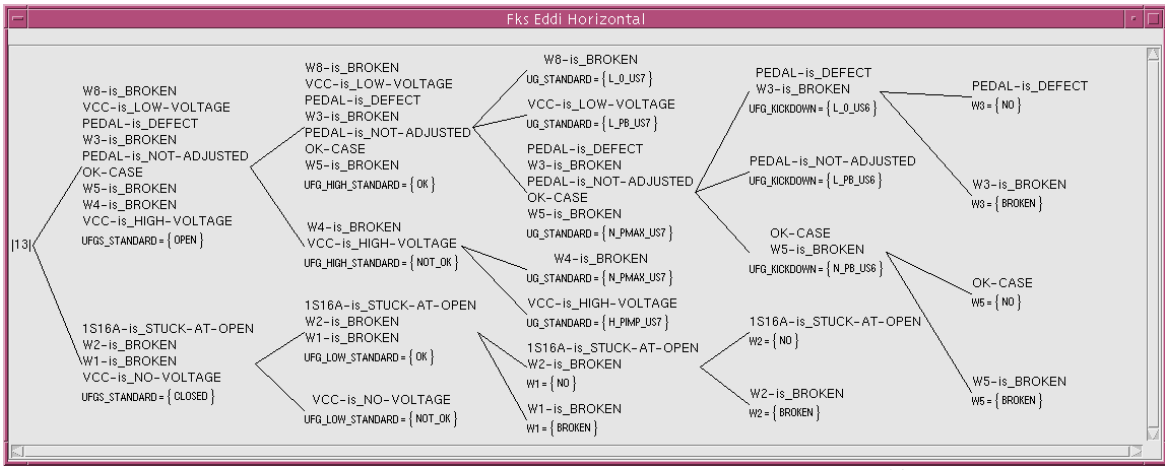

Figure 8. Fault tree of accelerator pedal circuit computed by GFT^{A*}

cost should be reduced. Furthermore, considering model-based approaches, it is essential to predict symptoms adequately and to model application specific fault identifying actions accurately. Additionally, the complexity of the device modeling process has to be minimized.

Developing MAD, we paid massive tribute to these challenges. In particular, MAD's fault trees are integrated into existing STILL diagnosis systems. Since MAD allows accurate reproduction of current STILL fault identification strategies, automatically generated fault trees will be accepted by service technicians. As another point, MAD's fault trees allow cost-optimized fault identification. To reduce complexity of the device modeling process, MAD provides several model libraries which allow extensive reuse of models build in former modeling sessions. For further reduction of the modeling complexity, we are currently working on automatic model generation from electronic product data such as CAD. MAD provides multiple possibilities for design engineers and diagnosis experts to guide automated fault tree generation. MAD's internal representation of electrical circuits is hidden from users such that device models can be interactively assembled from prespecified generic models. A deeper understanding of MAD's internal electrical models is not necessary for this task. To assure high diagnostic performance, we developed a new method for qualitative electrical network analysis which shows certain features avoiding spurious symptom predictions. The prototypical implementation of MAD allows model-based behavior prediction and automatic generation as well as manual modification of fault trees.

In cooperation with the STILL GmbH Hamburg, we have evaluated the MAD system in the application scenario and 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 handcrafted 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 diagnosis system. STILL service experts found that this fault tree can be used for fault identification.

Acknowledgments

This research has been supported by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) under the grant 01 IN 509 D 0, INDIA - Intelligente Diagnose in der Anwendung.

References

(Cascio et al. 99) Cascio, F., Console, L., Guagliumi, M., Osella, M., Panati, A., Sottano, S., Theseider Dupré, D.: On-board diagnosis of automotive systems: from dynamic qualitative diagnosis to decision trees, IJCAI-99, Workshop on Qualitative Reasoning for Complex Systems and their Control,1999.

(Faure et al. 99) Faure, P.-P., Trave-Massuyes, L., Poulard, H.: An Interval Model-Based Approach for Optimal Diagnosis Tree Generation, in: Proc. DX-99, 10th International Workshop on Principles of Diagnosis, 1999.

(Mauss and Neumann 96) Mauss, J., and Neumann, B.: How to Guide Qualitative Reasoning about Electrical Circuits by Series-Parallel Trees, in Proc. QR'96, Tenth International Workshop on Qualitative Reasoning, 1996.

(Milde et al. 99) Milde, H., Hotz, L., Kahl, J., Wessel, M.: Qualitative Analysis of Electrical Circuits for Computer-based Diagnostic Decision Tree Generation, in: Proc. DX-99, 10th International Workshop on Principles of Diagnosis, 1999.

(Price and Pugh 96) Price, C., Pugh, D.: Interpreting Simulation with Functional Labels, in: Proc. QR'96, 10th International Workshop on Qualitative Reasoning about Physical Systems, 1996.

(Price et al. 96) Price, C., Wilson, M., Timmis, J., Cain, C.: Generating Fault Trees from FMEA, in: Proc. DX-96, The Seventh International Workshop on Principles of Diagnosis, 1996.

(Quinlan 86) Quinlan, J. R.: Induction of Decision Trees, in: Machine Learning, I:81-107, 1986.

(Russel and Norvig 95) Russel, S., Norvig, P., in : Artificial Intelligence A Modern Approach, Prentice Hall, 1995.

(Struss 94) Struss, P.: Testing for Discrimination of Diagnoses, in Working Papers DX-94, Fifth International Workshop on Principles of Diagnosis, 1994.