An Efficient Diagnostic Algorithm based on Boolean Rules

Peter Tondl, Patrick Rammelt, Oliver Berger, Ulrich Siebel
Carmeq GmbH
[Peter.Tondl | Patrick.Rammelt | Oliver.Berger | Ulrich.Siebel]@carmeq.com

Abstract

This paper describes an efficient diagnostic engine algorithm based on Boolean rules which fits the requirements of the diagnostic concept introduced in [1]. The suggested algorithm benefits from combining failure notifications of several observers, expressed by indicators, to calculate a minimal and complete and therefore most effective diagnostic result.

Kurzfassung

Der folgende Artikel beschreibt einen auf der Anwendung Bool'scher Regeln basierenden, effizienten Algorithmus zur Ermittlung systeminterner Fehlerursachen bei einem gegebenen, äußeren Fehlerbild. Das erstmals in [1] vorgeschlagene Verfahren nutzt die und profitiert von der Kombination von Fehlerbildern verschiedener Beobachter, repräsentiert durch Fehler-Indikatoren, um ein minimales und vollständiges und daher möglichst effektives Diagnose-Resultat zu erzeugen.

1 Introduction

At least two conditions have to be fulfilled to find the origin of an observed failure pattern of a technical system. First, knowledge about the system itself and therefore the ability to generate a precise description of it is needed. And second, an algorithm to put both, observation and description, together to a true and complete *failure pattern explanation*. An example of such an algorithm based on Boolean rules is given in this paper.

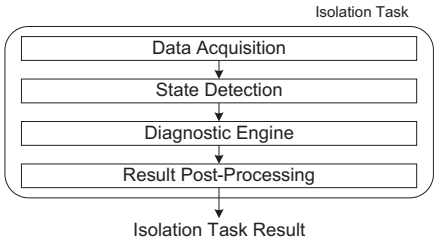

Figure 1: Isolation Task [1]

To work properly, a failure pattern explanation algorithm that implements a diagnostic engine has to be surrounded by appropriate pre- and post-processing steps. The solution described here takes advantage of two steps of pre-processing: *Data Acquisition* and *State Detection*. While the Data Acquisition part collects messages of different systems or system components and converts them to "understandable" observations, the State Detection part is responsible for filtering unneeded observations as well as for merging "low-level" observations to a higher level whenever this might be helpful. Thus, State Detection can be seen as a system wide or even cross-system monitor, able to detect states invisible to all other parts of the observed object.

Finally, a *Result Post-Processing* part concludes the entire operation. It orders, classifies and splits the result of the *Diagnostic Engine* part in order to convert them into a more maintenance friendly form, i.e. making them more understandable for humans or machines.

A second approach for a Diagnostic Engine algorithm which can be combined with the approach introduced here or used as stand-alone solution shows [2].

2 System Description

A system description used for diagnostic purposes must fit all requirements of the applied diagnostic function. Often this means that not only a logical description of the system is needed but a physical one as well. A second condition usually is that diag-

nostics per se is required to cause very little effort. For this reason alone it makes sense to develop a diagnostic algorithm that is capable of using already existing data, e.g. system design data like wirings, hardware components and their interconnections. Another advantage of this approach is that system design changes find their way into their corresponding "diagnostic world" almost instantly and the entire update process can be designed highly automated.

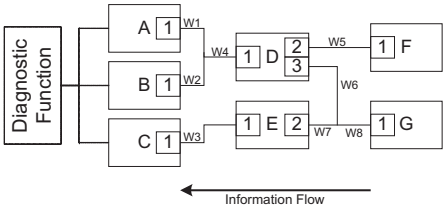

Figure 2: Test System

Figure 2 shows a simple test system. Seven devices (A to G) are connected together via wirings ($W1$ to $W8$) and port drivers ($A1$, $B1$, ..., $G1$). The entities A, B and C use data sent by F and G via D and E. An additional entity hosts the diagnostic function and is connected reliable to A, B and C. Each of these components host an application as well as a hardware based monitor.

Table 1 describes this system as logical, table 2 as physical *expressions of link descriptions*. Both link types are observed by different monitors: logical links by an application monitor, physical links by a hardware monitor. Links, logical as well as physical, always connect only two entities, a link is therefore well-defined by exactly these entities.

Device A: Information flow $F \rightarrow A$ $\rightarrow A1, W1, W4, D1, D2, W5, F1$ Device A: Information flow $G \rightarrow A$ $\rightarrow A1, W1, W4, D1, D3, W6, W8, G1$
Device B: Information flow $F \rightarrow B$ $\rightarrow B1, W2, W4, D1, D2, W5, F1$ $DeviceB$: Information flow $G \rightarrow B$ $\rightarrow B1, W2, W4, D1, D3, W6, W8, G1$
Device C: Information flow $G \rightarrow C$ $\rightarrow C1, W3, E1, E2, W7, W8, G1$

Table 1: Application Monitor Link Descriptions at Logical Level

However, for a diagnostic algorithm as described in this paper, information about the physical realization of links is also needed. Hence, each component, software module or wiring that is necessary to build up a link has to be included as part of its respective link description.

Device A: Hardware connection $D \rightarrow A$ $\rightarrow A1, W1, W4, D1$
Device B: Hardware connection $D \rightarrow B$ $\rightarrow B1, W2, W4, D1$
Device C: Hardware connection $E \rightarrow C$ $\rightarrow C1, W3, E1$

Table 2: Hardware Monitor Link Descriptions at Physical Level

This means for example, for the test system depicted in fig 2: The logical link $F \rightarrow A$, observed by the application monitor of A, needs the following components for its physical realization: $A1$, $W1$, $W4$, $D1$, $D2$, $W5$ and $F1$. If the link is broken, at least one of these components has to be the origin of this fault.

Fortunately, it does not matter whether an application, hosted on component A and receiving information sent by F, has knowledge about these components or not (usually it has not). Only the fact is important that the application monitor is still able to communicate a disorder of this logical link to its diagnostic function.

3 Data Acquisition

As already described in [1], the aim of a data acquisition function is to collect data from all available sources to provide all subsequent blocks with relevant diagnostic information. Figure 3 shows a data acquisition block, acquiring fault and operational states, e.g. by using application programming interfaces of system modules, and receiving messages from queuing and sampling ports.

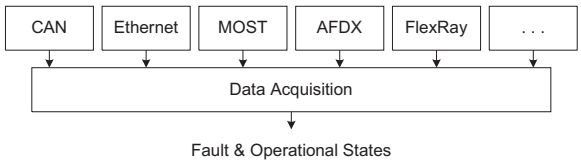

Figure 3: Data Acquisition [1]

Data acquisition delivers two types of information to its successor block: *Fault State* information and *Operational State* information. Fault states represent fault detections made by a monitor. Operational states stand for, e.g., physical values, operation conditions and user requirements.

4 State Detection

In order to run a diagnostic function with maximum performance pre-processing of its input data can be helpful or might even be necessary. The latter one becomes true immediately in case of data which represent operational or environmental values. Such state values or operational states may be very inconvenient regarding to their own data formats. Moreover, they do not necessarily indicate a problem by themselves. On the other hand, even in cases where input data gathered by the Data Acquisition task are only consisting of directly applicable values, they may be not valid and therefore not allowed in a certain situation. A third reason for applying a pre-processing task before performing a diagnostic function is the possibility of a combination of failure notifications. This means, e.g. that depending on a current configuration of a system, notifications may only become valid in combination with other notifications and should therefore be filtered out in cases where they are not useful or simply not allowed.

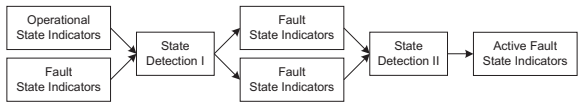

Figure 4: State Detection [1]

4.1 First Pre-Processing Part

The State Detection algorithm starts with reading its configuration data. An important part of this configuration is the (so called) *Indicator Values Table* which is used to build up a list of *Active Fault State Indicators*. At this point the list consists only of active fault state indicators that have been set and sent by different monitors of the system before.

After building the initial version of the Active Fault State Indicators list the first State Detection part checks whether operational state indicators exist or not. If so, it applies its rules straight forward to these indicators. This leads to a setting of additional fault state indicators. If not, the rule application part is bypassed.

4.2 Second Pre-Processing Part

As described above the State Detection compares operational state values with pre-defined rules in order to detect faulty states, e.g. a "value out of range" state. In a more advanced version of this concept an additional State Detection module can be implemented as well. In this step, fault indicators, previously generated by the State Detection algorithm itself as well as fault indicators received by monitors, can be combined together by applying a further rule set to express failure patterns of higher levels.

The diagnostic algorithm proceeds only in case of a non-empty Active Fault State Indicators list by calling the Diagnostic Engine.

5 Diagnostic Engine

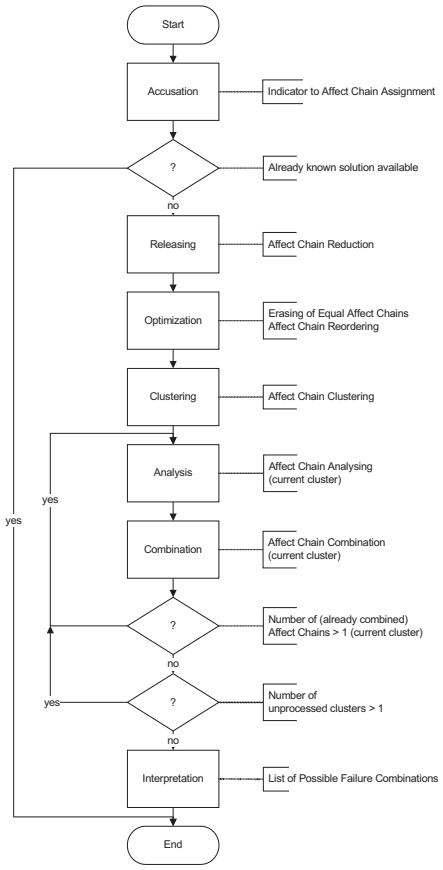

Figure 5: Diagnostic Engine Algorithm

The Diagnostic Engine algorithm becomes active by receiving a non-empty Active Fault State Indicators list, sent by its predecessor process. As shown in fig 5 the algorithm works in several stages: Accusation, releasing, optimization, clustering, analyzing, combination and interpretation. This means that a certain failure pattern, observed by monitors and sent to the diagnostic engine via Data Acquisition and State Detection, leads to a set of accused components in a first step. After this, a second step follows where the algorithm tries to release as many components as possible from accusation. A further

step combines all remaining components. Finally, the diagnostic result is presented to the user of the system in the last step.

5.1 Accusation

Example 1 ($D1$)
Using the test system shown in fig 2 under the assumption that a component part of D is faulty (in this case driver $D1$, fig 6). This leads to observations like "No data from F and G" for the application monitor and "No data from D" for the hardware monitor of device A as well as of device B.

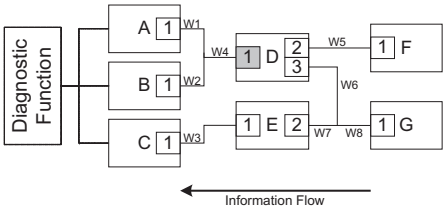

Figure 6: Faulty Driver $D1$

Table 3 describes the resulting *Affect Chains* of these observations expressed by Boolean rules. "X" denotes an element which may be faulty while "!X" an element which is definitely not faulty.

Application Monitor A:
$\rightarrow$ $A1$ or $W1$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow$ $A1$ or $W1$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor B:
$\rightarrow$ $B1$ or $W2$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow$ $B1$ or $W2$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor C:
$\rightarrow$!$C1$ and !$W3$ and !$E1$ and !$E2$ and !$W7$ and !$W8$ and !$G1$

Hardware Monitor A:
$\rightarrow$ $A1$ or $W1$ or $W4$ or $D1$
Hardware Monitor B:
$\rightarrow$ $B1$ or $W2$ or $W4$ or $D1$
Hardware Monitor C:
$\rightarrow$!$C1$ and !$W3$ and !$E1$

Table 3: Faulty Driver $D1$

□

Example 2 ($G1$)
Using the test system shown in fig 2 again, this time under the assumption that a component part of G is faulty (driver $G1$, fig 7). This leads only to observations like "No data from G" for the application monitors of A, B and C, while no hardware monitor of these devices reports any failure at all.

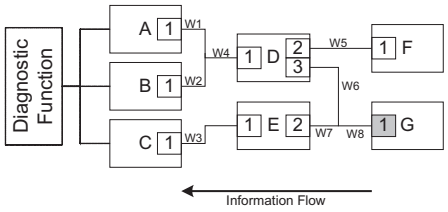

Figure 7: Faulty Driver $G1$

Table 4 describes the resulting Affect Chains of these observations expressed by Boolean rules. "X" denotes an element which may be faulty while "!X" an element which is definitely not faulty.

Application Monitor A:
$\rightarrow$!$A1$ and !$W1$ and !$W4$ and !$D1$ and !$D2$ and !$W5$ and !$F1$
$\rightarrow$ $A1$ or $W1$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor B:
$\rightarrow$!$B1$ and !$W2$ and !$W4$ and !$D1$ and !$D2$ and !$W5$ and !$F1$
$\rightarrow$ $B1$ or $W2$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor C:
$\rightarrow$ $C1$ or $W3$ or $E1$ or $E2$ or $W7$ or $W8$ or $G1$

Hardware Monitor A:
$\rightarrow$!$A1$ and !$W1$ and !$W4$ and !$D1$
Hardware Monitor B:
$\rightarrow$!$B1$ and !$W2$ and !$W4$ and !$D1$
Hardware Monitor C:
$\rightarrow$!$C1$ and !$W3$ and !$E1$

Table 4: Faulty Driver $G1$

□

The first step of the Diagnostic Engine algorithm is the accusation of each component that belongs to a monitor's affect chain when the respective observer detects a failure as described in the examples above. In the special case of using only one observer, i.e. either an application monitor or a hardware monitor, this is already the diagnostic result.

In case of having more than one observer, e.g. both, the application monitor *and* the hardware monitor of device A, the current observations can be combined to obtain a much more precise diagnostic result.

Summarized, example 1 leads to {$A1$, $B1$, $D1$, $D2$, $D3$, $W1$, $W2$, $W4$, $W5$, $W6$, $W8$, $F1$, $G1$} as accused components, while example 2 leads to {$A1$,

$B1$, $C1$, $D1$, $D2$, $D3$, $E1$, $E2$, $W1$, $W2$, $W3$, $W4$, $W6$, $W7$, $W8$, $G1\}$ as accused components.

5.2 Releasing

From a Boolean point of view a component which is part of a successful executed operation cannot be the source of a failure pattern observed at the same time ("either or" principle). Thus, such a component can be released *explicitly* from the list of accused components. Given the examples above, this leads in case of example 1 to exactly the same list of still accused components, while the list in example 2 can be significantly shortened to $\{D3$, $E2$, $W6$, $W7$, $W8$, $G1\}$.

Nevertheless, a combination of different failure observations may lead to an *implicit* releasing as well. A main advantage of using Boolean Rules lies in the fact that in a lot of cases not all chains have to be taken into account to identify a faulty component. This means that some of the chains which remain after accusation and explicit releasing might become redundant. For example, a Boolean combination of, chain $1 = \{A\}$ and chain $2 = \{A, B\}$ leads in Boolean terms to $A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A$ (with $\cdot \equiv$ "and" as well as $+ \equiv$ "or"). In other words, $\{A\}$ as a subset of $\{A, B\}$ makes the latter one redundant. Chains which represent a component superset of at least one other chain can be discarded completely because their additional elements are not needed to describe the underlying failure pattern (with the exception of "hidden failures" as shown in section 7). Taken this into account, the list of accused components of example 1 shortens to $\{A1$, $B1$, $D1$, $W1$, $W2$, $W4\}$.

5.3 Optimization

As a result of the releasing step some of the remaining affect chains may consist now of exactly the same components. Thus, equal chains can be grouped together and erased completely from the list with the exception of only one remaining chain per group.

At this point in the diagnostic process a minimized set of affect chains has been identified which is still able to explain its related failure pattern completely. "Minimized" in this context means that both of the following actions are impossible: Erasing of a further chain or component without getting a wrong diagnostic result as well as adding a chain or component without making the combination step unnecessarily more complex.

5.4 Clustering

In some cases one failure observation may be the result of more than one failure event. Furthermore, these failure events, or at least two of them, may be totally independent of each other. The combination of affect chains which belong to independent failures or failure groups leads probably to both: An increased complexity of the combination step as well as an unnecessarily complex diagnostic result. The clustering step separates affect chains which do not share common components into different groups or clusters. Thus, each cluster represents one or more of the currently observed failure events which possess a common relationship. In other words, failures belonging to different clusters are independent from each other and therefore do not need to be combined.

5.5 Analyzing

In case of a large number of affect chains it might become vital how, and especially in what order, these chains have to be combined together to keep hardware performance requirements as low as possible.

To determine a solution for this requirement the first assumption is that two chains that are to be combined have the same length n. Furthermore, the number of shared components between two chains is m, with:

$$0 \leq m \leq n \tag{1}$$

The rules of Boolean combinatorics hold as well, especially:

$$\begin{aligned} A + A &= A \\ A + B &= B + A \\ A \cdot A &= A \\ A \cdot B &= B \cdot A \\ A \cdot (A + B) &= A \end{aligned}$$

Now two affect chains of length $n = 4$ are considered with a different number of shared components m. In case of $m = 0$, i.e. completely different chains, the following result might occur after combination:

$(A + B + C + D) \cdot (E + F + G + H) = AE + AF + AG + AH + BE + BF + BG + BH + CE + CF + CG + CH + DE + DF + DG + DH$

Similarly for $m = 1$, i.e. with one shared component A:

$(A + B + C + D) \cdot (A + F + G + H) = A + BF + BG + BH + CF + CG + CH + DF + DG + DH$

As for $m = 2$, with A and B as components in common:

$(A+B+C+D)\cdot(A+B+G+H) = A+B+CG+CH+DG+DH$

With $m = 3$ results from the previously used procedure:

$(A+B+C+D)\cdot(A+B+C+H) = A+B+C+DH$

Finally, in complete compliance of the chains or $m = 4$:

$(A+B+C+D)\cdot(A+B+C+D) = A+B+C+D$

In summary, in case of $n = 4$ and $m = 0, 1, \ldots, n$ and for the number of (composite) components c after combination the following table of values applies:

n	m	c
4	0	16
4	1	10
4	2	6
4	3	4
4	4	4

In case of $n = 1, 2, 3$ and applying the previously used procedure again this table can be extended to:

n	m	c
3	0	9
3	1	5
3	2	3
3	3	3
2	0	4
2	1	2
2	2	2
1	0	1
1	1	1

Which leads to the following equation:

$$c = (n-m)^2 + m \tag{2}$$

The number n is fix, m mutable. Equation (2) is therefore a function $f(m)$ in dependence of variable m.

The question is now, which m minimizes $f(m)$:

$$\frac{\mathrm{d}}{\mathrm{d}m} f(m) = 0 \tag{3}$$

Which leads to $2\cdot(m-n)+1 = 0$ or:

$$m = n - \frac{1}{2} \tag{4}$$

In case of different lengths n_1 and n_2 equation (2) changes slightly to:

$$c_{12} = (n_1 - m)(n_2 - m) + m \tag{5}$$

Which leads to:

$$m = \frac{n_1 + n_2}{2} - \frac{1}{2} \tag{6}$$

For the special case of $n_1 = n_2$ equation (6) changes to equation (4) as expected.

For an efficient comparison of a given chain set k and according to equation (4) and (6), chains have to be found that are as similar as possible to each other. This means that each affect chain *pair* must be analyzed to determine the degree of identity between the two respective chains.

Equation (7) reflects the effort for these comparisons as a function of the number k of chains which have to be compared:

$$\sum_{i=1}^{k-1} i\cdot(k-i) \tag{7}$$

To facilitate the determination of the degree of complexity equation (7) can be converted to:

$$\begin{aligned}
&\sum_{i=1}^{k-1} i\cdot(k-i)\\
=\;& \sum_{i=1}^{k} i\cdot(k-i)\\
=\;& \sum_{i=1}^{k}(ki - i^2)\\
=\;& k\cdot\sum_{i=1}^{k} i - \sum_{i=1}^{k} i^2\\
=\;& k\cdot\frac{k(k+1)}{2} - \frac{k(k+1)(2k+1)}{6}\\
=\;& \frac{1}{6}(k^3 - k)
\end{aligned}$$

The result shows that for large values of k a cubic complexity can be expected, since the linear portion of $k^3 - k$ can be neglected then. For small values of k however, the analysis effort is slightly reduced by the linear portion.

Equation (4) and (6) show the possibilities for reducing the resulting chain length:

(1) Short resulting chains are generated by affect chains, which are similar. This means that both chains should have as many common components m as possible: $m \rightarrow\ min(n_1, n_2)$. The pefect situation is that a chain is a subset of another chain. This produces a resulting chain, which is identical to the shorter one of the two input chains.

 Published in: Bäker, Unger (Hrsg.), "Diagnose in mechatronischen Fahrzeugsystemen V", Dresden, Germany, May 2012, ISBN: 978-3-8169-3149-2

(2) Short chains n_i can produce only a short resulting chain. The maximum possible resulting chain length is $n_1 \cdot n_2$. Due to the approximately quadratic relationship, this effect is particularly significant for large chains.

For practical implementation, it is usually best to start searching for chains that are a subset of another chain. In such a case the chain which is the superset can be completely excluded from combinatorics and thus deleted.

In a second step, the remaining chains, sorted according to their length, are checked again for similarity. Since all superset chains have been already deleted, in this step it is only important to find another chain that is *as similar as possible*.

Because of the fact that the greatest reduction of the resulting chain length can be achieved already in case of only few common components, a test for similarity does not have to be complete necessarily. Therefore, a good compromise between analysis effort on the one hand and achieved results on the other hand has to be found.

5.6 Combination

In the combination step affect chains which belong to the same cluster are combined together. This means that the components of two chains are multiplied with each other by applying Boolean rules. The resulting chain is minimized if possible, again by applying Boolean rules, and multiplied with the following chain of the list.

$D1$ or
$W4$ or
$A1$ and $B1$ or
$A1$ and $W2$ or
$B1$ and $W1$

Table 5: Result List of Example 1

The combination is repeated until all chains of each cluster are combined together to only one list of components per cluster. This list is the result of the Diagnostic Engine process. Table 5 shows the result list of example 1 and table 6 the result list of example 2.

$G1$ or
$W8$ or
$D3$ and $E2$ or
$D3$ and $W7$ or
$E2$ and $W6$

Table 6: Result List of Example 2

5.7 Interpretation

In the result list examples shown in section 5.6 a weighting of accused components is not considered with the exception of errors involving a single component before those involving multiple faulty components at once. Hence, a user of the diagnostic result, e.g. a maintenance operation as shown is section 7, can choose their preferred proceeding in order to find the cause of the observed failure pattern. As the diagnostic results show, the true failure causes in these examples, component parts $D1$ and $G1$, are not only included in their lists but are also able to explain their observed failure patterns exhaustively, too. Due to the simplicity of the examples given with only six independent observers it is not possible to obtain a more precise diagnostic result which leads, e.g. to only one failure cause.

6 Result Post-processing

When the performance of a diagnostic process leads to a single and a double failure explanation of the observed failure pattern, the likelihood for the single failure as the correct diagnostic result is usually higher than the likelihood for the double failure. However, if an event which leads to a failure has an impact on more than one component of the observed system, failure explanations consisting of these components shall become much more likely. This demand holds even stronger when all subcomponents of one composed component are involved. A diagnostic result based on, e.g. three wirings as a possible cause of an observed failure pattern, should be rated as much more likely in a case where these wirings belong to a common harness than if they are parts of different harnesses. Furthermore, if these wirings do not only belong to the same harness but entirely build this harness instead, i.e. there is no other wiring belonging to the harness, the harness itself shall be accused as well.

The information needed to perform such a hierarchical reasoning as described above can be obtained directly from the configuration data of the system. A more flexible hierarchical reasoning can be realized by using additional information from an additional configuration file. Therefore, if further information about a system is available, a diagnostic process may not only accuse for example each driver of a device that appears totally “dead” but the responsible power supply too.

7 Hidden Failures

The diagnostic algorithm described in this paper is able to find the origin cause of an observed failure pattern. In most cases, there is only one iteration step necessary to fulfill this task successfully.

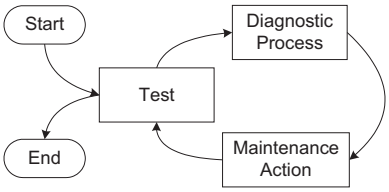

Figure 8: Cooperation of Diagnostics and Maintenance (Principle) [1]

Nevertheless, this is not necessarily true in any case. Diagnostics can only be successful in cooperation with adequate maintenance actions and is not finished until the system state is completely shifted back to "healthy" independent of the number of required iteration steps. Figure 8 shows this principle of cooperation between diagnostics and maintenance.

Example 3 ($D1 + F1$)
Using the test system shown in fig 2 once more, now under the assumption that a component part of D and a component part of F is faulty *at the same time* (in this case drivers $D1$ and $F1$, fig 9). This leads to observations like "No data from F and G" for the application monitor and "No data from D" for the hardware monitor of device A as well as of device B.

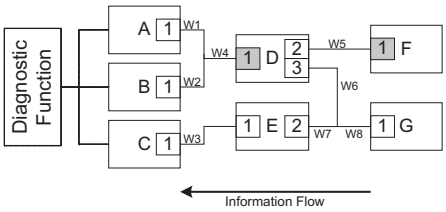

Figure 9: Faulty Drivers $D1 + F1$

Table 7 describes the resulting Affect Chains of these observations expressed by Boolean rules. As before, "X" denotes an element which may be faulty while "!X" an element which is definitely not faulty.

Table 8 shows the result list of example 3 after performing the diagnostic task. As it can be seen the result is exactly the same as in example 1. Obviously, a faulty driver $F1$ cannot be seen *through* a faulty driver $D1$ by the diagnostic function with the given system configuration. This means, a failure pattern caused by $F1$ alone must be a subset

Application Monitor A:
$\rightarrow A1$ or $W1$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow A1$ or $W1$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor B:
$\rightarrow B1$ or $W2$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow B1$ or $W2$ or $W4$ or $D1$ or $D3$ or $W6$ or $W8$ or $G1$
Application Monitor C:
$\rightarrow$!$C1$ and !$W3$ and !$E1$ and !$E2$ and !$W7$ and !$W8$ and !$G1$

Hardware Monitor A:
$\rightarrow A1$ or $W1$ or $W4$ or $D1$
Hardware Monitor B:
$\rightarrow B1$ or $W2$ or $W4$ or $D1$
Hardware Monitor C:
$\rightarrow$!$C1$ and !$W3$ and !$E1$

Table 7: Faulty Drivers $D1 + F1$ – First Step

of a failure pattern caused by $D1$ alone and thus "hidden" by the latter one.

$D1$ or
$W4$ or
$A1$ and $B1$ or
$A1$ and $W2$ or
$B1$ and $W1$

Table 8: Result List of Example 3 – First Step

A hidden failure requires always an application of the diagnostics and maintenance cooperation principle, as shown in fig 8, to obtain the desired solution. Thus, after a first step of fault isolation and elimination, e.g. by replacing component D, the following system test reveals a second failure pattern, this time caused only by $F1$ and therefore different from the first one (table 9).

Table 10 shows the result list of example 3 after performing the diagnostic task for the second time. Now, $F1$ is clearly visible as fault origin (beside $W5$ and $D2$) and no longer hidden by another faulty component.

□

8 Conclusions

Two conditions have to be fulfilled to find the origin of an observed failure pattern of a technical system. First, knowledge about and therefore the ability to generate a precise description of the system. And second, an algorithm to put both, observation and description, together.

Application Monitor A:
$\rightarrow A1$ or $W1$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow !A1$ and $!W1$ and $!W4$ and $!D1$ and $!D3$ and $!W6$ and $!W8$ and $!G1$
Application Monitor B:
$\rightarrow B1$ or $W2$ or $W4$ or $D1$ or $D2$ or $W5$ or $F1$
$\rightarrow !B1$ and $!W2$ and $!W4$ and $!D1$ and $!D3$ and $!W6$ and $!W8$ and $!G1$
Application Monitor C:
$\rightarrow !C1$ and $!W3$ and $!E1$ and $!E2$ and $!W7$ and $!W8$ and $!G1$

Hardware Monitor A:
$\rightarrow !A1$ and $!W1$ and $!W4$ and $!D1$
Hardware Monitor B:
$\rightarrow !B1$ and $!W2$ and $!W4$ and $!D1$
Hardware Monitor C:
$\rightarrow !C1$ and $!W3$ and $!E1$

Table 9: Faulty Drivers $D1 + F1$ – Second Step

$F1$ or $W5$ or $D2$

Table 10: Result List of Example 3 – Second Step

To work properly, a failure pattern explanation algorithm that implements a diagnostic engine, has to be surrounded by an appropriate pre- and post-processing steps. The solution described in this paper takes advantage of Data Acquisition as first pre-processing step and of State Detection as second. Finally, a Result Post-Processing part concludes the entire operation.

A system description used for diagnostic purposes must fit all requirements of the applied diagnostic function. This includes a logical as well as a physical description of the observed system. In addition, diagnostics per se must cause very little effort. Thus, it makes sense to develop a diagnostic algorithm that is capable of using already existing data. Another advantage of this approach is that system design changes find their way into their corresponding diagnostic world almost instantly and the entire update process can be designed highly automated.

The Diagnostic Engine algorithm described in this paper works in several stages: Accusation, releasing, optimization, clustering, analyzing, combination and interpretation. This means, a certain failure pattern leads to a set of accused components in a first step. A second step follows where the algorithm tries to release as much components as possible from accusation. A further step combines all remaining components. Finally, the diagnostic result is presented to the user of the system in the last step.

In most cases there is only one iteration step necessary to fulfill this task successfully. A hidden failure though, requires an application of the diagnostics and maintenance cooperation principle to obtain the desired solution.

The suggested Diagnostic Engine algorithm can be combined with other approaches like Bayesian networks [2], to get an even more precise diagnostic result. Another possible application is in the field of system fault simulation, i.e. to analyse a system design whether it can be properly diagnosed or not.

References

[1] Peter Tondl, Patrick Rammelt, and Ulrich Siebel. State based Diagnostics of System Internal Fault Origins by using Boolean Rules. In *Diagnose in mechatronischen Fahrzeugsystemen IV*. Bäcker, Unger, 2011. (978-3-8169-3068-6).

[2] Patrick Rammelt, Peter Tondl, and Ulrich Siebel. Diagnostic under Uncertainty. Scheduled to be published in 2012.