Automated Fault Diagnosis in Embedded Systems[⋆]

Peter Zoeteweij, Jurryt Pietersma, Rui Abreu, Alexander Feldman, and Arjan J.C. van Gemund

Delft University of Technology,
P.O. Box 5031 NL-2600 GA Delft, The Netherlands

Abstract. Automated fault diagnosis is emerging as an important factor in achieving an acceptable and competitive cost/dependability ratio for embedded systems. In this paper, we introduce model-based diagnosis and spectrum-based fault localization, two state-of-the-art approaches to fault diagnosis that jointly cover the combination of hardware and control software typically found in embedded systems. In this paper we present an introduction to the field, discuss our recent research results, and report on the application on industrial test cases.

Keywords: diagnosis, embedded systems, dependability.

1 Introduction

The complexity of the systems that we use on a day-to-day basis is constantly growing. This trend is particularly strong in the area of embedded systems, where new functionality can quickly be realized in software. In combination with an ever decreasing time-to-market, and a practically constant rate of faults per line of code, this implies that system reliability is decreasing.

Automated, or computer-aided diagnosis techniques are emerging as an important means to counter this trend. They serve to localize the faults that are the root causes of system failures. As such, they help to shorten the test-diagnose-repair cycle in the software development process, allowing more faults to be removed. In addition, automated diagnosis techniques can be used in maintenance, and can serve as the basis for (automated) recovery. In this role they are a vital ingredient of dependable autonomic systems.

In this paper we introduce two approaches to automated diagnosis: model-based diagnosis (MBD), and spectrum-based fault localization (SFL). The former technique, which originated in the artificial intelligence domain, infers a

[⋆] This work has been carried out as part of the Tangram, Trader, and Finesse projects. The Tangram and Trader projects are carried out under the responsibility of the Embedded Systems Institute, and are partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026 and the BSIK03021 program, respectively. The Finesse project is supported by STW grant DES.7015.

diagnosis from a compositional, behavioral model combined with real-world observations, and has successfully been applied to digital circuits and complex mechanical systems. The latter technique, SFL, overcomes the dependability of MBD on suitable behavioral models, and applies naturally to software, for which these models are generally not available. Together, MBD and SFL cover the combination of hardware devices and control software that is typically found in the area of embedded systems.

The remainder of this paper is organized as follows: In Sect. 2 we take a high-level view on the diagnosis problem, which allows us to compare and relate MBD and SFL. In Sect. 3 and 4 we introduce the actual techniques, and discuss our current research and successful applications of MBD and SFL on industrial test cases. We conclude in Sect. 5.

2 The Diagnosis Problem

Central to a discussion on diagnosis are the notions of failure and fault. A *failure* is a discrepancy between expected and observed behavior, and a *fault* is a property of the system that causes such a discrepancy. The notion of an *error* is sometimes used to indicate a system state that potentially leads to a failure [4], but for our purposes the distinction between failure and error is largely artificial, and depends only on what can be observed and what has been specified.

The purpose of diagnosis is to identify the system components that are the *root cause* of observed failures. We consider that a system consists of n components, and that it applies some system function $\underline{y} = f(\underline{x}, \underline{h})$, where $\underline{x}$ and $\underline{y}$ represent observations of system input and output, respectively, and where $\underline{h} = (h_1, \ldots, h_n)$ indicates the *health state* of each of the n components (see Fig. 1). The basic health states of a component are healthy and faulty, but as we shall see in Sect. 3, this can further be refined. Diagnosis can be understood as solving the inverse problem $\underline{h} = f^{-1}(\underline{x}, \underline{y})$, i.e., find the combinations of component health states that explain the observed output for a given input. Note that the internals of the system are not observable, which distinguishes the diagnosis problem from a component testing problem.

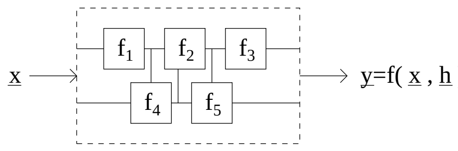


Fig. 1. Conceptual view of a system

Because the exact system function f is generally not known, diagnosis always involves the use of a system *model*. Depending on the amount of information that this model M provides about the system components, it can be used for diagnosis as follows.

– If M is composed from models $M_1, \ldots, M_n$ for each of the system components, and thus specifies their interaction and nominal behavior, it can be used for diagnosis by searching for combinations of health states $\underline{h}$ such that $M(\underline{x}, \underline{h}) = \underline{y}$.
– If, in addition to the nominal system behavior $M(\underline{x}) = \underline{y}'$, the model only specifies how the components interact to determine the various elements y_i of $\underline{y}$, without describing their actual behavior, it can still be used for diagnosis by identifying the extent to which components are exclusively involved in the observations y_i for which $y_i \neq y_i'$.

The former approach to diagnosis is known as model-based diagnosis, which is the subject of Sect. 3. The latter approach is known as spectrum-based fault localization, which is the subject of Sect. 4.

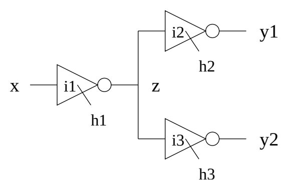
3 Model-based Diagnosis

Model-based diagnosis was first proposed by Reiter [16] and De Kleer [6] and implemented in the General Diagnostic Engine. The best known practical examples can be found in the space domain [19] and in automotive applications [18]. Examples in the space domain are the Deep Space 1 [14] and Earth Observing 1 [15] missions, both part of NASA's New Millennium Program. Related but separate fields are that of Fault Tolerant control and diagnosis with Bayesian networks. In this section we describe the principles of MBD, the modeling technique, the underlying inference algorithms, and industrial applications.

3.1 Principles of Model-based Diagnosis

We demonstrate the principles of model-based diagnosis using the circuit of Fig. 2(a) as an example. This system consists of three logical inverters, which we model as components, each with an input u and an output v. The healthy, nominal behavior of an inverter is that the output is equal to the inverted input, i.e., $h \Rightarrow (v \Leftrightarrow \neg u)$.

(a)



(b)

$$h1 \Rightarrow (z \Leftrightarrow \neg x)$$
$$h2 \Rightarrow (y1 \Leftrightarrow \neg z)$$
$$h3 \Rightarrow (y2 \Leftrightarrow \neg z)$$

Fig. 2. Three-inverters example: (a) circuit and (b) model

We can *compose* a model of the circuit of Fig. 2(a) from three models of inverter components, by relating their inputs and outputs according to the connections of the circuit. This model is shown in Fig. 2(b). Note that this only models healthy behavior. Faulty behavior is left implicit, which makes this a *weak* model.

If we were able to probe the inputs and outputs of all components of a system, the diagnosis problem would be trivial: we could simply search for the components whose behavior does not correspond to its component model. Typically, however, a significant part of the system is hidden from observation (the inside of the dashed box in Fig. 1), and we have to *reason* about possible explanations for the observed, faulty behavior instead. In the case of our example circuit, we assume that only x, $y1$, and $y2$ can be observed.

Suppose we observe $(x, y_1, y_2) = (1, 0, 1)$. The observation $y_1 = 0$ indicates a system failure. The combinations of component health states that explain this observation in the context of our model are listed in the first column of Table 1. Because of the model weakness and limited observability, multiple explanations exist. This means that after this observation we still have some residual uncertainty about the actual system health state.

weak	strong	$z = 1$	multiple
$(1, 0, 1)$	$(1, 0, 1)$		
$(0, 0, 1)$	$(0, 0, 1)$		$(0, 0, 1)$
$(1, 0, 0)$			$(1, 0, 0)$
$(0, 1, 0)$		$(0, 1, 0)$	$(0, 1, 0)$
$(0, 0, 0)$		$(0, 0, 0)$	$(0, 0, 0)$

Table 1. Diagnoses (h_1, h_2, h_3) for the example circuit, with $(x, y_1, y_2) = (1, 0, 1)$

There are a number of ways to reduce this uncertainty and improve the quality of a diagnosis. One possibility is to add constraints to the model that further specify the faulty behavior. For example, if we want to explore the possible explanations for the specific fault that an inverter has its output permanently set to 0 (called *stuck-at-zero*), we can add the constraint $\neg h \Rightarrow \neg v$ to the component model.

For the resulting model, which is called *strong* because it makes the faulty behavior explicit, the diagnosis is listed in the second column of Table 1. The additional constraints reduce the number of solutions, which improves the quality of the diagnosis. Note that we have to be careful with this strong modeling approach as our model can no longer explain any behavior resulting from unanticipated faults, and may prove to be inconsistent with some observations.

Another approach is to improve quality by increasing the observability. Consider adding $z = 1$ to the observation. The resulting diagnosis is shown in the third column of Table 1. Note that for this simple example, all variables are now observable, which makes the diagnosis problem trivial. In general, this will not be the case. Even so, while all explanations indicate that i_1 and i_3 are broken, there is still uncertainty about the health state of i_2.

Besides increasing the number of observable variables we can also choose to use multiple observations, which capture the behavior of the system for different inputs. The results of combining the initial observation with a second observation $(x, y_1, y_2) = (0, 0, 1)$ is shown in the last column of Table 1.

3.2 Diagnostic Performance

The model of Fig. 2(b) constitutes a propositional formula, and standard techniques for propositional satisfiability solving can be used to automatically find those explanations (h_1, h_2, h_3) that are consistent with the system description and observations. We have implemented MBD with the modeling language LYDIA (Language for sYstem DIAgnosis, [13]) and accompanying tools. LYDIA is designed to facilitate a conversion to a propositional formula in polynomial time.

However, the underlying satisfiability problem is NP-complete, and requires efficient, and specialized methods for searching the solution space. One such method is to use a probability heuristic that can be applied in an A* search algorithm. A further important improvement on such an algorithm is the use of conflict sets to skip over inconsistent solutions. For LYDIA we have implemented Conflict-directed A* (CDAS) as proposed by [19].

Another promising approach, which has also been implemented, is to exploit model hierarchy [9]. A hierarchical system description is composed of smaller partial system descriptions that are organized in a hierarchical structure with one system description on the highest level. By exploiting the hierarchical information and selectively compiling parts of the model it is possible to increase the diagnostic performance and to trade cheaper preprocessing time for faster run-time reasoning. Our hierarchical algorithm, being sound and complete, allows large models to be diagnosed, where compile-time investment directly translates to run-time speedup. Experiments with a diagnosis benchmark based on ISCAS-85 circuitry models have shown speed-up factors between 2 to 5.6 compared to the CDAS algorithm.

Last, but not least, the use of non-deterministic, i.e., stochastic algorithms to traverse the search space provides an important speed-up for multiple fault diagnoses. We have implemented a greedy stochastic algorithm called SAFARI (StochAstic Fault diagnosis AlgoRIthm, [8]). For weak fault models, it can compute 80-90% of all cardinality-minimal diagnoses, several orders of magnitude faster than state-of-the-art deterministic algorithms, such as CDAS, allowing systems with several hundreds of components to be diagnosed in seconds. This algorithmic research will be applied to the fault diagnosis of Océ copiers within the STW/PROGRESS project FINESSE (see [13]).

LYDIA supports variables both in the Boolean and finite integer (FI) domains. The use of FI domains is costly in terms of diagnosis time. We have shown that an algorithm working directly in the FI domain is a preferred option over Boolean encodings, as it offers speed-ups of up to two orders of magnitude [7].

3.3 Industrial Applicability

As part of the TANGRAM project five subsystems of ASML [3] wafer scanners have been modeled in LYDIA for diagnosis. Table 2 lists the following characteristics of these cases: engineering discipline, whether it involved dynamic system functionality, the model size, the time spent on the modeling, and the (estimated) improvement in diagnosis time.

system	**engineering disciplines**	**dynamic**	**model size [loc]**	**modeling time [days]**	**diagnosis time order of magnitude**	
					original	MBD
LASER	E, M, S, O	X	806	20	days(*)	ms (*)
EPIN	E, M		37	7	days	ms
POB	E, M, O		500	12	hours (*)	s (*)
ILS	E		82	8	minutes	ms
WS	E, M, S, H	X	2151	15	days	s

Table 2. Overview of modeling cases. E = electric, M = mechanic, S = software, O = optical, and H=hydraulic. Estimates are indicated with (*).

The EPIN case is a good example of how even a relative simple system consisting of three sensors, an actuator, and some safety monitoring logic, can be problematic for a diagnosis based on human reasoning. In one particular case it took two days to finally correctly identify the fault sensor because of an initial mistake in the diagnostic reasoning. The EPIN and WS cases have both been extended by exploring automatic model derivation from Netlists and VHDL code respectively. This automatic modeling step reduces modeling effort and decreases model maintenance as part of the model can be kept automatically up-to-date to design.

In half of the cases the diagnosis time had to be estimated based on earlier cases and simulation experiments. Based on these estimates and actual diagnosis times we find that the investment in modeling time yields significant speed-ups in diagnosis time.

4 Spectrum-based Fault Localization

While MBD is well-suited for circuits and hardware devices, software is rarely modeled in sufficient detail during development, and derivation of suitable behavioral models from source code is troublesome at best. However, as we already indicated in Sect. 2, diagnosis can be performed in absence of such models by analyzing the involvement of components of a system in the faulty behavior.

Consider the example failure $(x, y_1, y_2) = (1, 0, 1)$ of the previous section. Without knowing the functionality of the components, from Fig. 2(a) we can still deduce that i_1 is involved in both output observations. Inverter i_3 is only involved in the correct output observation y_2, but i_2 is the only component that is exclusively involved in the faulty output observation y_1. This makes i_2 the most likely cause of the observed failure, while i_3 is least likely to be involved.

This technique, which is known as spectrum-based fault localization, applies quite naturally to software, which can be seen as an executable model that indicates, through profiling instrumentation, the involvement of its various components in correct and faulty behavior. Examples of existing systems for diagnosis and debugging that implement SFL are Pinpoint [5], which focuses on large, dynamic on-line transaction processing systems, and Tarantula [12] which focuses on C software. In this section we introduce the principles of SFL, and describe our recent research on its diagnostic performance. In addition, we discuss a successful application to industrial (embedded) software.

4.1 SFL Principles

The name SFL refers to the use of so-called *program spectra* [17] for measuring the activity of software components. Here we will be using *block hit spectra*, which are arrays of binary flags with an entry per block of source code (see Fig. 3), indicating the activity or inactivity of that block.

For the application of SFL to software, we require the program spectra of several runs of a program, some of which have demonstrated an error or failure (called *failed* runs). The other runs are called *passed* runs. The block hit spectra of several runs constitute a binary matrix, whose columns correspond to the different components (blocks in this case) of the program. Fig. 3 shows an example for six runs and four components. The passed / failed information constitutes another column vector, which is called the error vector, and encodes the comparison of $\underline{y}$ and $\underline{y}'$ in Sect. 2. Fault localization essentially consists in identifying the component whose column vector resembles the error vector most.

To quantify this resemblance we use a *similarity coefficient*, as known from data clustering (see, e.g., [11]). As an example, the *Jaccard* similarity coefficient (see also [11]) expresses the similarity of two binary vectors $\underline{x}$ and $\underline{y}$ as as the number of positions in which they share an entry 1, divided by this same number plus the number of positions in which they differ: $s = \frac{a_{11}}{a_{11}+a_{01}+a_{10}}$, where $a_{pq} = |\{i \mid x_i = p \wedge y_i = q\}|$, and $p, q \in \{0, 1\}$.

To illustrate the approach, consider the function of Fig. 3. It is meant to sort, using the bubble sort algorithm, a sequence of rational numbers, but it has a bug in the swapping code of block 4: only the numerators are swapped. The table in Fig. 3 shows the block hit spectra for six input sequences. The fault in the swapping code only manifests itself for the fifth input, $\langle \frac{3}{1}, \frac{2}{2}, \frac{4}{3}, \frac{1}{4} \rangle$, for which the output is $\langle \frac{1}{1}, \frac{2}{2}, \frac{4}{3}, \frac{3}{4} \rangle$ instead of $\langle \frac{1}{4}, \frac{2}{2}, \frac{4}{3}, \frac{3}{1} \rangle$. Consequently, we mark the fifth run as failed by an entry 1 in the error vector. The Jaccard similarities $s_1, \ldots, s_4$ of the error vector to the four column vectors in the spectrum data are listed in the bottom row of the table. Block 4 has the highest similarity, which (correctly) identifies the swapping code as the most likely location of the fault.

4.2 Diagnostic Performance

The calculated similarity coefficients rank the components of a system with respect to the likelihood that they cause the detected failures. If the actual location of a fault is known, we can then assess the quality of the SFL diagnosis based on its position in the ranking. This way, we have investigated the influence of several parameters on the diagnostic quality, using a benchmark set of software faults known as the *Siemens set* [10]. This set consists of seven C programs that

```
void RationalSort(int n, int *num,
                  int *den){
  /* block 1 */
  int i,j,temp;
  for ( i=n-1; i>=0; i-- ) {
    /* block 2 */
    for ( j=0; j<i; j++ ) {
      /* block 3 */
      if (RationalGT(num[j], den[j],
              num[j+1], den[j+1])) {
        /* block 4 */
        temp = num[j];
        num[j] = num[j+1];
        num[j+1] = temp;
}}}}
```

	block				
input	1	2	3	4	error
$\langle\,\rangle$	1	0	0	0	0
$\langle\frac{1}{4}\rangle$	1	1	0	0	0
$\langle\frac{2}{1},\frac{1}{1}\rangle$	1	1	1	1	0
$\langle\frac{4}{1},\frac{2}{2},\frac{0}{1}\rangle$	1	1	1	1	0
$\langle\frac{3}{1},\frac{2}{2},\frac{4}{3},\frac{1}{4}\rangle$	1	1	1	1	1
$\langle\frac{1}{4},\frac{1}{3},\frac{1}{2},\frac{1}{1}\rangle$	1	1	1	0	0
$s_j =$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	

Fig. 3. A faulty C function for sorting rational numbers and its program spectra

range in size from 20 to 124 blocks of code. For every program, a number of faulty versions is available, each with a known bug. In addition, every program has a set of test cases that ensures full code coverage.

One of our research results concerns the so-called *Ochiai* similarity coefficient. This coefficient is known from biology, and to our knowledge, it has not previously been applied to SFL. Using the 120 single-site faults of the Siemens set, we observed that the Ochiai coefficient outperforms several other coefficients, including the ones used by the Pinpoint (the Jaccard coefficient) and Tarantula tools mentioned above. This is illustrated in Fig. 4, which shows the quality of the diagnosis for these three coefficients and different rates at which fault activations lead to failures. Here, the quality of the diagnostic is expressed as the percentage of code that need not be examined if the SFL ranking is followed when searching for the fault, averaged over all faults in our benchmark set. It also shows that SFL can already provide a useful diagnosis at low failure rates.

Further experiments have shown that including more failed runs is always safe because the accuracy of the diagnosis either improves or remains the same. We observed little or no improvement for more than six failed runs. However, while stabilizing around twenty runs, the effect of including more passed runs is unpredictable, and may actually lead to worse diagnoses. To what extent these results depend on program characteristics is subject to further investigations. See [1, 2] for details on these experiments.

4.3 Industrial Application

To some extent, the Siemens set faults are artificial, and to evaluate its practical applicability we implemented SFL for the control software of a product line of analog television sets from Philips (now NXP), and diagnosed two faults, one existing, and one seeded to replicate a problem in another product line.

A known problem with the specific version of the control software that we had access to, is that after teletext viewing, the CPU load when watching television (TV mode) is approximately 10% higher than before teletext viewing. To

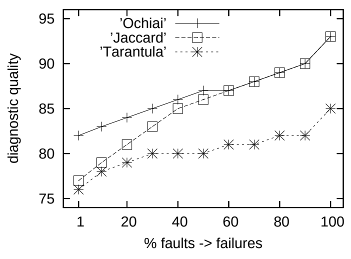


Fig. 4. SFL diagnostic quality

diagnose this problem, we obtained hit spectra for the functions tied to the ±300 logical threads in the control software. We generated a new spectrum every second, and used a scenario of 60 s. TV mode, 30 s. teletext viewing, and 60 s. TV mode, where we marked the spectra for the last 60 s. as failed. In the resulting ranking, the function that was known to activate the fault came second.

The other fault entails that a text search in a teletext page without visible context locks up the teletext functionality. A likely cause for this lock-up, by which we reproduced it in our experimental platform, is an inconsistency in the values of two state variables in different subsystems. To diagnose this problem, we collected hit spectra for all blocks of code in the control software (over 60,000). Each time a key was pressed on the remote control we started recording a new spectrum. For error detection we used an assert-like check on the two state variables, that would flag a failure if they assumed an invalid combination. This way, we were quickly able to find a straightforward scenario of 26 key presses, including a magic sequence to activate the fault, that yields a perfect diagnosis.

These experiments have confirmed our belief that because of its low time and space complexity, SFL is well suited for the embedded software domain, which is characterized by scarce memory and CPU resources, and high concurrency. See [20] for a more in-depth discussion.

5 Conclusion

Model-based diagnosis and spectrum-based fault localization are two practicable approaches to automated diagnosis that have successfully been applied to systems with embedded software. The experiments reported in Sect. 3.3 have led ASML to adopting MBD as a means to increase the efficiency of the manual diagnosis process. As a result of the promising outcome of the experiment reported in Sect. 4.3, further SFL experiments are now conducted at NXP to evaluate the technique on actual problem reports filed during the development of a more recent product.

In addition to a further investigation of factors that influence the diagnostic accuracy of MBD and SFL, we believe that there are many opportunities for combining the two diagnosis techniques, and in our future work we plan to investigate how these can be exploited.

Acknowledgments. The authors gratefully acknowledge their discussions with the TANGRAM, TRADER, and FINESSE project partners.

References

1. R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proc. TAIC PART'07*. IEEE Computer Society, 2007 (to appear).
2. R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proc. PRDC'06*, pp. 39 – 46. IEEE Computer Society, 2006.
3. ASML website. http://www.asml.com.
4. A. Avižienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.
5. M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proc. DSN 2002*, IEEE Computer Society, 2002.
6. J. de Kleer and B. C. Williams. Diagnosing multiple faults. In M. L. Ginsberg, ed., *Readings in Nonmonotonic Reasoning*, pp. 372–388. Morgan Kaufmann, 1987.
7. A. Feldman, J. Pietersma, and A. J. C. van Gemund. A multi-valued SAT-based algorithm for faster model-based diagnosis. In C. A. Gonzáles, T. Escobert, and B. Pulido, ed., *Proc. DX-06*, pp. 93–100, June 2006.
8. A. Feldman, G. Provan, and A. J. C. van Gemund. Approximate model-based diagnosis using greedy stochastic search, In *Proc. SARA'07*, LNCS 4612, pp. 139–154. Springer, 2007.
9. A. Feldman and A. J. C. van Gemund. A two-step hierarchical algorithm for model-based diagnosis. In *Proc. AAAI'06*, pp. 827–833. AAAI Press, 2006.
10. M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. ICSE'94*, pp. 191–200, IEEE Computer Society / ACM Press, 1994.
11. A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Prentice-Hall, 1988.
12. J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proc. ASE 2005*, pp. 273–282, ACM Press, 2005.
13. LYDIA website, http://www.fdir.org/lydia/.
14. NASA. Deep space 1 website,. http://nmp.nasa.gov/ds1/.
15. NASA. Earth observing 1 website,. http://eo1.gsfc.nasa.gov/.
16. R. Reiter. A theory of diagnosis from first principles. In M. L. Ginsberg, ed., *Readings in Nonmonotonic Reasoning*, pp. 352–371. Morgan Kaufmann, 1987.
17. T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In M. Jazayeri, H. Schauer, ed., *Proc. ESEC/FSE'97*, LNCS 1301, pp. 432–449. Springer, 1997.
18. P. Struss. A model-based methodology for the integration of diagnosis and fault analysis during the entire life cycle. In H.-Y. Zhang, editor, *Preprints of SAFEPROCESS 2006*, pp. 1225–1230. IFAC, 2006.
19. B. C. Williams and R. J. Ragno. Conflict-directed A* and its role in model-based embedded systems. *J. Discrete Applied Math*, 155(12)1562–1595, Elsevier, 2003.
20. P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. C. van Gemund. Diagnosis of embedded software using program spectra. In *Proc. ECBS'07*, pp. 213–218. IEEE Computer Society, 2007.