A model-based approach to fault diagnosis of embedded systems[1,2]

Jurryt Pietersma, Arjan J.C. van Gemund and Andre Bos

The problems that arise from the integration of subsystems into complex, multi-disciplinary embedded systems, are a potential obstruction for the expected, exponential growth in embedded systems applications. Faults that occur because of the dynamic behavior of the integrated system are difficult to trace back to individual subsystems or components. The Model-Based Diagnosis (MBD) methodology offers a solution for the fault diagnosis of the integrated system by inferring the health of a system from a compositional system model and real-world measurements. In this article we present the initial results of our MBD research as applied on the lithography systems of ASML. We explain our methodology based on a modelling language LYDIA which is specifically being developed for the purpose of MBD. Furthermore we discuss the results of our first diagnosis test case.

Introduction

As the exponential increase in hardware performance-per-cost ratio is expected to continue, the number of embedded systems is to increase accordingly. The associated complexity crisis is a potential show stopper for the continued pervasion of embedded systems in our society. This is particularly true for complex, multi-disciplinary systems that are integrated from multiple subsystems. While these subsystems might function well separately, integrating them can cause unexpected faults. Because of the dynamic interaction between these subsystems, these faults take a lot of time and effort to diagnose, let alone fix.

One of the solutions is to automate the fault diagnosis of these integrated embedded systems. The classical way of automated diagnosis e.g., by means of application-specific code or, more generically, by using expert systems, has disadvantages. The mapping from symptoms to diagnosis is explicitly coded in the software, which means that even a minor design change of the system may require a major redesign of the diagnosis software. It also means that while trying to decrease system complexity, we actually increase it by adding a lot of diagnosis software.

A promising way of overcoming these problems is to apply a *model-based* approach to diagnosis. In the Model-Based Diagnosis (MBD) approach [5], knowledge about the system is expressed in terms of a compositional model. A generic fault diagnosis engine, using AI search algorithms, consults this model during run-time, while tracking the system. Because information about the system design is separated from the fault finding method, a design change only requires a similar change in the model. This curbs the increase in complexity.

[1]This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

[2]This article was originally presented at the ASCI Conference 2004.

Within the TANGRAM project [5], a multi-university research project aimed at model-based testing and diagnosis of multi-disciplinary embedded systems, the MBD approach is applied to lithography systems as produced by ASML. While the ever increasing performance of these chip manufacturing systems actually provides us with the aforementioned exponential increase of the hardware performance-per-cost ratio, these systems themselves are by no means free from the complexity crisis. Hence, MBD is seen as an important solution to decrease the cost of design, integration and operation of these systems.

Despite recent advances in MBD [6, 10, 11, 13] complex, multi-disciplinary systems as found in ASML are currently beyond the state-of-the-art. Furthermore, given an adequate MBD technique, a subsequent problem is model specification, which is a labor-intensive and error-prone process. Within the TANGRAM project MBD research focuses on extension of MBD technology with respect to time, state and probability.

Our MBD approach is based on the modelling language LYDIA (Language for sYstems DIAgnosis) [7]. LYDIA is model-based systems specification language aimed at systems fault diagnosis and simulation using the same model. In this article we present the initial results of our MBD research as pursued in the TANGRAM project. We demonstrate how LYDIA can be used for diagnosis in general. In addition, we describe how this methodology has been applied in terms of a case study within the TANGRAM context.

The article is organized as follows. In the first section we introduce the principles of MBD with an example. In the second section we present the LYDIA modelling language and accompanying tools, including two examples on how to use these tools for diagnosis. In the third section we present the case study and discuss the resulting model and its diagnosis. In the final section we draw our conclusions from this initial research.

Model-Based Diagnosis

Diagnosis is the process of finding differences between models and reality. Model-Based Diagnosis (MBD), first suggested by Reiter [12] and continued by de Kleer, Mackworth and Reiter [4], is the process of finding faults in a system on the basis of observations from reality and reasoning about a model of the system. Formally, model-based diagnosis can be seen as finding faulty components that explain the difference between behavior predicted by a model and behavior observed in reality.

For example, consider an example of MBD using a digital circuit, consisting of three inverters: A, B, and C (Figure 1). Let $w = 1$. Then y and z should be 1 as well. If observations indicate that $y = 0$ and $z = 1$ then the diagnosis could be that component B is faulty. Another option is that A and C are faulty, as this also explains the symptoms. The trivial solution, A, B, and C all faulty, also explains the observations but is of no added value, as any superset of $\{B\}$ or $\{AC\}$ explains the observations. A subset of $\{B\}$ or $\{AC\}$ does not. That is why $\{B\}$,$\{AC\}$ can be called the *minimal* fault set. This diagnosis can be formalized, using a logical model, as follows.

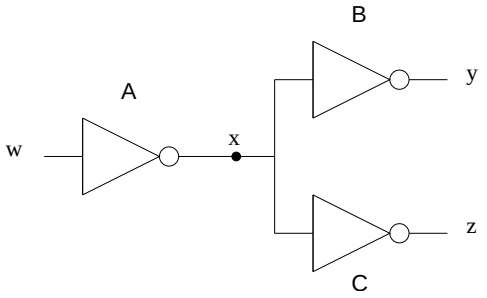

Figure 1: Three-inverters example

Let h indicate the *health* of a component. If $h = 1$ then the component is "healthy" and obeys certain behavioral rules. The three inverter example has three components (A,B,C), so it has three such rules:

$$h_A \rightarrow x = \overline{w}$$
$$h_B \rightarrow y = \overline{x}$$
$$h_C \rightarrow z = \overline{x}$$

As the observations are: $w = 1, y = 0, z = 1$, it follows (applying the rule $p \rightarrow q \Leftrightarrow \overline{p} + q$):

$$(\overline{h}_A + \overline{x}) \cdot (\overline{h}_B + x) \cdot (\overline{h}_C + \overline{x}) = 1 \qquad (3)$$

This can be rewritten to DNF-form:

$$\overline{h}_A\overline{h}_B\overline{h}_C + \overline{h}_A\overline{h}_B\overline{x} + \overline{h}_A\overline{h}_C x + \overline{h}_B\overline{h}_C\overline{x} + \overline{h}_B\overline{x} = 1$$

This formula reduces to the following prime implicants:

$$\overline{h}_A\overline{h}_C x + \overline{h}_B\overline{x} = 1$$

Thus $\overline{h}_A\overline{h}_C = 1$ (A and C are faulty when $x = 1$) or $\overline{h}_B = 1$ (B is faulty when $x = 0$).

Another possibility to calculate the faulty components is by using conflict sets. Applying the resolution rule $(p+q)\cdot(r+\overline{q}) \rightarrow (p+r)$ and De Morgan's Laws, from equation (3) it follows:

$$\begin{aligned} (\overline{h}_A + \overline{h}_B) \cdot (\overline{h}_B + \overline{h}_C) &= 1 \\ \overline{(\overline{h}_A + \overline{h}_B) \cdot (\overline{h}_B + \overline{h}_C)} &= 0 \\ \overline{(\overline{h}_A + \overline{h}_B)} + \overline{(\overline{h}_B + \overline{h}_C)} &= 0 \\ h_A h_B + h_B h_C &= 0 \end{aligned}$$

so $h_A h_B = 0$, and $h_B h_C = 0$ which means $\{$AB$\}$ and $\{$BC$\}$ are conflict sets. Finding the minimal fault set, or minimal conflicts, can be done using algorithms for the Hitting Set problem. This, of course, also results in the sets $\{$AC$\}$ and $\{$B$\}$. In summary, in MBD of combinational systems the model is solved for h using propositional logic. In the next section we describe our tool for MBD.

Model-Based Diagnosis with LYDIA

LYDIA

In the following, we briefly present some of the major features of LYDIA. Due to space constraints we only present those constructs that are used in the sequel. For a comprehensive introduction to LYDIA we refer to [7]. Each LYDIA statement is a Boolean equation (proposition), and all statements apply concurrently. Each variable, e.g., `x` is a function of (continuous) time, i.e., `x(t)`. The time argument is omitted. All operators are functions that operate on each time argument (i.e., element-wise data flow). Thus,

```
op(x)  <=> for all t: op(x(t)) = true
x op y <=> for all t: x(t) op y(t) = true
```

Roughly speaking, LYDIA can be placed in the "functional" category of the functional (equational) vs. imperative (state-transition) dichotomy. It resembles synchronous languages [2], such as Lustre [9] and Signal [8], with the major difference being the absence of synchronous time. Timed actions are asynchronous, i.e., signals (and events) are not sampled at regular time intervals. State transitions may also be timeless (cf. timed and immediate transitions in timed Petri nets [1]), with only the transitions that are enabled at the same time being synchronous. In this respect, LYDIA resembles a synchronous language with infinite clock resolution, which is implemented through a discrete-event propagation scheme. Although based on a functional approach, many of the LYDIA models are expressed in a state-transition style as syntactic sugar. The reason for this is that the description of some systems (e.g., state-machines) in a functional language sometimes proves awkward, where a more state-transition-oriented dialect offers a much more natural model.

Combinational Operators

Apart from the usual operators, such as `=`, `+`, `-`, `/`, `*`, `and`, `or`, `not`, `>`, `<`, `>=`, `<=`, `sin`, `cos`, `tan`, `sqrt`, `pow`, `log`, `exp`, `max`, `min`, `abs`, etc., the derived operators include `!=`, `if`, `if-else`, defined as:

```
a != b <=> ! (a = b)

if (c) x <=> (! c) + x

if (c) x else y <=> (c * x) + ((! c) * y)
```

where `!`, `+`, `*`, are equivalent to `not`, `or`, `and`, respectively.

Time Operator

Time delay is described by the `after` function:

```
y = (x after delta default x0)
```

that defines a signal (variable) `y` that lags behind the signal `x` according to

$$y(t) = \begin{cases} x(t-\delta), & t \geq \delta; \\ x0, & 0 \leq t < \delta. \end{cases}$$

The default clause is optional.

Apart from the above constructs, LYDIA also features state transition operators, the treatment of which, however, is beyond the scope of this article.

LYDIA tools

Currently we have developed a number of tools that operate on LYDIA models. There is a LYDIA compiler called `lydia` that translates LYDIA models into C source code for the purpose of simulation, or into symptom-diagnosis lookup tables for the purpose of diagnosis. The latter tables are generated using propositional SAT solving and are consulted by a diagnostic engine, called `scotty`, that monitors the system's input and output, and returns a list of possible diagnoses, in order of probability. Currently the C compilation mode only works for models that operate in the discrete time domain. A second simulator `lsim` has been developed which interprets and simulates continuous-time LYDIA models.

Examples

This section describes some basic LYDIA examples. The first LYDIA system models an electronic inverter with a 10ns propagation delay, after which `y` becomes the inverted of `x`. The second example produces a clock signal `c` with a period of 1.0s. The last example simulates a bouncing ball with height `h` and velocity `v`. The velocity is reversed when the velocity and height are less than zero. The velocity and height are calculated using explicit first order Euler integration as specified by the function `integrate`.

Example 1:

```
system inverter (x: bool, y: bool)
{
  t_p = 1e-08
  y = ( not x after t_p )
}
```

Example 2:

```
system clock (c: bool)
{
  period = 1.0
  c = ( ( not c ) after period / 2 )
}
```

Example 3:

```
system ball (h: float,
             v: float,
             g: float,
             d_t: float,
             c: float)
{
  h = (integrate(h,v,dt)
      after dt default 5.0)
  v = ((if (b) (-c * v)
      else
      integrate(v,-g,dt))
      after dt default 0.0)
  b = ((v < 0.0) and (h < 0.0))
  exit = (time > 10.0)
}

function integrate (y: float,
                    f: float,
                    dt: float) : float =
{
  integrate(y,f,dt) = (y + f * dt)
}
```

Diagnosis of inverter model

Consider the inverter of the previous section, which this time either inverts a Boolean signal if healthy, or is stuck-at-zero, if at fault. The LYDIA model is given by:

```
system inverter (x: bool,
                 h: bool,
                 y: bool)
{
  t_p = 1e-08
  y = if ( h )
      ( !x after t_p default false )
      else false
}
```

where x, y denote input, output respectively, and h denotes the so-called health variable. We can run this model with lsim and a data input file, which results in the following output:

```
time:      x: h: y:
0.00000000 1  1  0
1.00000000 0  1  0
1.00000001 0  1  1
2.00000000 1  0  0
3.00000000 0  0  0
```

The first column indicates the simulation time, the second and third column are the input variables which are repeated from the input file. The result of the simulation is shown in the last column and corresponds to the expected output, y is only true, 10ns after the moment the inverter is healthy and the input is false.

It is instructive to note that the functional character of LYDIA allows us to use lsim simulator as a limited diagnostic engine. Instead of providing lsim with x and h we provide it with the observations x and y from which it deduces h, as shown below. The U symbol indicates an unknown value.

```
time:      x: y: h:
0.00000000 1  0  U
1.00000000 0  0  U
1.00000001 0  1  1
2.00000000 1  0  0
2.00000001 1  0  U
3.00000000 0  0  U
3.00000001 0  0  0
```

We observe that a diagnosis for this system is only possible in two out of four cases, namely only when the output of a healthy system, with a delay of 10ns, does not coincide with the output of an unhealthy system. Thus, for this system, only when the output is true can we distinguish between an h that is true or false.

Diagnosis of three-inverters model

Of course, the real goals for using LYDIA for MBD is diagnosis of far more complex, real-life systems than the one mentioned in the previous section. To perform diagnosis of these systems we can compose models out of simpler components. To illustrate this, we expand our initial model of one inverter to a model of the three-inverters example mentioned in the first section:

```
#include inverter.sys
system inverter3 (w: bool,
                  hA: bool,
                  hB: bool,
                  hC: bool,
                  y: bool,
                  z: bool)
{
  probability ( hA = false ) = 0.01
  probability ( hB = false ) = 0.01
  probability ( hC = false ) = 0.01

  inverter ( w, hA, x)
  inverter ( x, hB, y)
  inverter ( x, hC, z)
}
```

A diagnostic approach based on mere simulation can no longer be used to diagnose this system because, as explained earlier, one combination of input and outputs can be caused by different types of failures. The simulator can only solve single equations for only one solution variable. To solve this general combinational problem we use the specialized diagnostic engine scotty, mentioned in Section 2, which can handle these combinatorics. At this point, our diagnosis algorithm does not allow time delay. Consequently in the following we consider the inverter model without the after statement. To make the model more generic and compliant with our logical three-inverters model, we also leave out the specific stuck-at-zero fault mode. To allow LYDIA to work with failure probabilities, we introduce the keyword probability, to indicate a health variable that has a certain probability of being false or true. As an example, we run the diagnostic engine with the input/output combination mentioned in the first section: w=1, y=0 and z=1. The result of the diagnostic engine is given by:

```
(0.97049200)  hA=true  hB=false hC=true
(0.00980295)  hA=false hB=false hC=true
(0.00980295)  hA=false hB=true  hC=false
(0.00980295)  hA=true  hB=false hC=false
(9.90197e-05) hA=false hB=false hC=false
```

The results correspond to the fault cases that can be derived from the minimal fault set {B},{AC} as calculated in the first section. The cases with two faulty inverters all have the same probability because all three inverters have the same individual failure probability. From the results it is also clear

that the trivial case of three failing inverters is extremely unlikely.

A current disadvantage of using `scotty` instead of `lsim` is the lack of support for time and state. As mentioned in the introduction, extending the diagnostic engine to incorporate this, is one of the goals of our ongoing research.

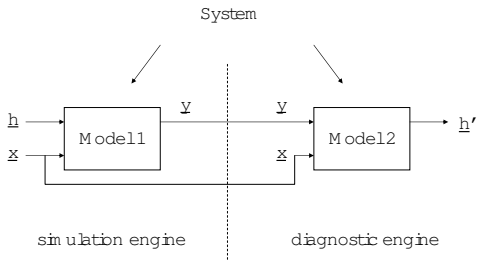

Figure 2: Connection between the simulation (M1) and diagnosis (M2) model of target system.

Modelling case study

Methodology

While the ultimate goal of our research is to diagnose lithography systems in the real world, our current goal is to gain experience in the specification of real-world models and our diagnosis algorithms. For this we need as few uncertainties as possible, which is why currently we only apply our diagnosis on the simulation models and not on the real system. Consequently, we proceed according to the following approach. We derive a simulation model M1 of the system under study. Its purpose is to:

1. document our understanding of the ASML systems including the possible failure modes of each component;
2. serve as a starting point for the derivation of a diagnosis model M2.

Our current experimental setup is shown in Figure 2. In this figure our simulation model M1 is on the left. We can insert failures (h) in this model, which we can then diagnose (h') using our diagnostic model M2. Ideally, h' should equal h for all (fault) scenarios.

In the current early stage of our research these models are generally not equal, because, as mentioned in the third section, while we have no problem *simulating* models with time and state, we are only able to *diagnose* combinational models. As we make progress, our diagnostic model M2 will evolve in the direction of M1. In the following we describe M1 and the subsequent derivation of M2.

Simulation model

At present, a laser sub-system is chosen as a case study for the TANGRAM project. The purpose of this system is to provide the lithography scanner with an exact dose of light energy to expose the wafer. The dose is provided in the form of laser pulses. Besides the laser, the model for this system also includes the interface with the scanner and the laser control software located at the scanner side.

To build this model of the laser system both a top-down and bottom-up approach is followed. In the top-down approach we model the entire structure of the whole system. We start out by interfacing with empty LYDIA systems and gradually add functionality and fault modes. In the bottom-up approach we choose a specific sub-system, of which the basic functionality is implemented in a LYDIA model. Furthermore, we also investigate known or interesting failure modes of this sub-system and introduce health variables to simulate this behavior. An example of this approach is the shutter module. The shutter can be thought of as part of the optical interface that blocks or passes on the light emitted by the laser. Beside this nominal functionality we also implemented the following faulty behavior. A nominal shutter would start opening when the "open" command is given, and would only report that it is fully opened when done. A fault mode of this shutter, which has been known to exist in an earlier design, is that it would not wait to be fully opened,

but would immediately return the "open" status after the command has been given. The following LYDIA code implements both the nominal and fault behavior.

```
% common.sys contains the clip and
% latch functions
#include common.sys

system shutter_M1 (
  % commands
  cmd_open: bool, cmd_close: bool,

  % health parameters
  h_open: bool, h_close: bool,

  %light coming in and going out
  light_in: float, light_out: float,

  % status
  sts_open: bool, sts_close: bool)
{
  % latch the mode based on the command
  latch (cmd_close, cmd_open, mode_open)
  latch (cmd_open, cmd_close, mode_close)

  sts_open = (h_open and (pos = 0.0))
           or (!h_open and mode_open)

  sts_close = (h_close and (pos = SHUT))
           or (!h_close and mode_close)

  step = if (mode_close) (CONST_STEP)
         else
           (if (mode_open) (-CONST_STEP)
           else (0.0))

  % integrate and clip position
  % between 0.0 and SHUT
  pos = clip ( 0.0, integrate (SHUT,
        pos, step, TIME_STEP ), SHUT )

  % calculate beam attenuation
  light_out = ((SHUT - pos) * light_in)
}
```

In this model the shutter latches the open or close command (pulse) to an internal mode (level). Depending on this mode the shutter position is either decreased (opened) or increased (closed). The LYDIA systems `latch`, `clip` and `integrate` are defined in the included LYDIA file `common.sys`. The `sts_open` and `sts_close` status signals are based on the shutter position if the sensors are healthy, and otherwise simply by the internal mode. The latter corresponds to the non-nominal behavior of the shutter.

Diagnostic model

As explained earlier, due to the limitation of our diagnostic algorithm, the diagnostic model for the current experiments is a simplified version of our simulation model. Again, we will use the shutter model as an example. The shutter model makes use of time, as it takes time to open or close, and uses state, as it has internal modes, `pos`, `mode_open` and `mode_close`, which determine the shutter position and whether it is opening or closing. The associated time and state variables prohibit our combinational diagnosis approach and therefore we have to convert M1 to a model M2 specifically suited for diagnosis.

In our conversion from M1 to M2 we take the following approach:

1. isolate the equations with health parameters, on the condition that they are combinational. For each health parameter we also introduce its probability of being false or true;
2. re-use those (auxiliary) equations from M1 that are required to solve the isolated, health equations.

Thus our diagnostic approach includes simulation next to diagnosis. The result of applying these two steps on our shutter model is as follows:

```
system shutter_M2
{
  % combinational health equations
  probability (h_open = false) =0.01
  probability (h_close = false) =0.01

  sts_open = (h_open and (pos = 0.0))
         or (!h_open and mode_open)

  sts_close = (h_close and (pos = SHUT)
         or (!h_close and mode_close)

  % auxiliary equations
  latch (cmd_close, cmd_open, mode_open)
  latch (cmd_open, cmd_close, mode_close)

  step = if (mode_close) (CONST_STEP)
         else
           (if (mode_open) (-CONST_STEP)
           else (0.0))

  pos = clip ( 0.0, integrate (SHUT,
        pos, step, TIME_STEP ), SHUT )
}
```

We use the M2 model to diagnose our M1 model with the setup shown in Figure 2. In this setup `lsim` simulates M1 as well as the auxiliary equations of M2. The combinational health equations of M2 are compiled into a symptom-diagnosis lookup table and used by `scotty` for the actual diagnosis, as explained in the second section.

Diagnostic test results

In the next experiment we use the following values for the constants: `SHUT`=1.0, `SHUTTER_STEP`=0.1 and `TIME_STEP`=0.01. As our models have a symmetric description for the open and close sensor, the simulation and diagnosis results for both sensors are also symmetric. Therefore we limit our discussion to the open sensor. In the first 6.51s we simulate a healthy open sensor. The first test starts at 1.00s and we allow the shutter to fully open, after which we close it again at 2.0s. The second run starts at 3.00s but now we interrupt the shutter at 3.01s, before it can open completely. At 5.0 we do the same but after the interrupt we open it again. In the second half (t $\geq$ 7.00s) we perform the same tests, only now with an unhealthy sensor. The experiment yields the following results:

```
time:
|    h_open_M1:
|    | cmd_open:
|    | | mode_open:
|    | | | (pos=0.0):
|    | | | | sts_open:
|    | | | | | h_open_M2:
|    | | | | | | probability:
|    | | | | | | |
1    2 3 4 5 6 7 8

0.00 1 0 0 0 0 1 0.9801
1.00 1 1 1 0 0 1 0.9900
1.11 1 1 1 1 1 1 0.9801
2.00 1 0 0 1 1 1 0.9900
2.01 1 0 0 0 0 1 0.9801
3.00 1 1 1 0 0 1 0.9900
3.01 1 0 0 0 0 1 0.9801
5.00 1 1 1 0 0 1 0.9900
5.01 1 0 0 0 0 1 0.9801
5.02 1 1 1 0 0 1 0.9900
5.13 1 1 1 1 1 1 0.9801
6.00 1 0 1 1 1 1 0.9801
6.50 1 0 0 1 1 1 0.9900
6.51 1 0 0 0 0 1 0.9801

8.00 0 1 1 0 1 0 0.9900
 8.11 0 1 1 1 1 1 0.9801
 9.00 0 0 0 1 0 0 0.9900
 9.01 0 0 0 0 0 1 0.9801
10.00 0 1 1 0 1 0 0.9900
10.01 0 0 0 0 0 1 0.9801
12.00 0 1 1 0 1 0 0.9900
12.01 0 0 0 0 0 1 0.9801
12.02 0 1 1 0 1 0 0.9900
12.13 0 1 1 1 1 1 0.9801
13.00 0 0 1 1 1 1 0.9801
```

The second column `h_open_M1` gives the inserted sensor health of our simulation model. The seventh column gives the diagnosed health `h_open_M2` as inferred from M2 and the last column the probability of this diagnosis. From the first part of the results we can see that `scotty` correctly predicts that the sensor is healthy. The second part shows that a correct diagnosis is only performed when the `(pos=0.0)` expression in the fifth column is unequal to the `sts_open` variable in the sixth column. In other words, when the output of the healthy shutter, for which `sts_open` is only true if `pos`=0.0, does not coincide with that of the unhealthy sensor, for which `sts_open` is only true if `mode_open` is true. This corresponds with the results from the diagnosis of the single inverter example in the first section.

Conclusions

In this article we have presented our MBD approach and research objectives as pursued in the TANGRAM project. We have also demonstrated how to use the modelling language LYDIA in this approach. The examples show that we can already model and simulate the basic functionality of a realistic subsystem. Furthermore we have shown how we can make these models suited for combinational diagnosis. In the coming period we will put more emphasis on the diagnosis of existing fault scenarios. From this we expect to learn more about how to deal with the occurrence of time and state behavior in our diagnosis models.

Acknowledgements

We gratefully acknowledge the feedback from the discussions with our TANGRAM project partners from ASML, Eindhoven University of Technol-

ogy, Embedded Systems Institute, NLR, TNO-TPD, Twente University and the University of Nijmegen.

References

[1] M. Ajmone Marsan, G. Balbo and G. Conte, "A class of Generalized Stochastic Petri Nets for the performance analysis of multiprocessor systems," *ACM Tr. on Comp. Syst.*, vol. 2, May 1984, pp. 93–122.

[2] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, January 2003, pp. 64–82.

[3] Tom Brugman and Frans Beenker, "Project plan for the TANGRAM project on model-based testing," Tech. Rep. Doc. Nr. 2002-10060 version 09, Embedded Systems Institute, Nov. 2002.

[4] Johan de Kleer, A. K. Mackworth and R. Reiter, "Characterizing diagnoses and systems," *Artificial Intelligence*, vol. 56, 1992, pp. 197–222.

[5] Johan de Kleer and Brian C. Williams, "Diagnosing multiple faults," in *Readings in Nonmonotonic Reasoning* (Matthew L. Ginsberg, ed.), Los Altos, California: Morgan Kaufmann, 1987, pp. 372–388.

[6] A. Fijany, F. Vatan, A. Barrett and R. Mackey, "New approaches for solving the diagnosis problem," 2002.

[7] A.J.C. van Gemund, "LYDIA Version 1.1 Tutorial," Tech. Rep. PDS-2003-001, Delft University of Technology, Nov. 2003.

[8] P.L. Guernic, M.L. Borgne, T. Gautier and C.L. Maire, "Programming real time applications with Signal," *Proceedings of the IEEE*, vol. 79, Sept. 1991, pp. 1321–1336.

[9] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, "The synchronous data-flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, September 1991, pp. 1305–1320.

[10] James Kurien, "Model-based monitoring, diagnosis and control." Ph. D. Thesis Proposal, 2000.

[11] Sriram Narasimhan and Gautam Biswas, "An approach to model-based diagnosis of hybrid systems," in *Hybrid Systems: Computation and Control HSCC* (C. J. Tomlin and M. R. Greenstreet, eds.), vol. 2289 of *LNCS*, Springer, Mar. 2002, pp. 465–478.

[12] R. Reiter, "A theory of diagnosis from first principles," in *Readings in Nonmonotonic Reasoning* (Matthew. L. Ginsberg, ed.), Los Altos, California: Kaufmann, 1987, pp. 352–371.

[13] Brian C. Williams and Robert J. Ragno, "Conflict-directed A* and its role in model-based embedded systems." To appear in Journal of Discrete Applied Math.

Contact Information

Jurryt Pietersma

Parallel and Distributed Systems Group
Faculty of Electrical Engineering
Mathematics and Computer Science
Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft
The Netherlands
j.pietersma@ewi.tudelft.nl

Arjan J.C. van Gemund

Parallel and Distributed Systems Group
Faculty of Electrical Engineering
Mathematics and Computer Science
Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft
The Netherlands
a.j.c.vangemund@ewi.tudelft.nl

Andre Bos

Science & Technology BV
P.O. Box 608, NL-2600 AP Delft
The Netherlands
bos@science-and-technology.nl