Proceedings of the

Thirteenth International Workshop on Principles of Diagnosis (DX-02)

Semmering, Austria
2-4 May 2002

Sponsored by

AVL DiTEST

European Office of Aerospace Research
and Development of the USAF

Materna

OCC'M Software

Siemens Austria, PSE PRO,
CES Design Services

Austrian Computer Society

Austrian Society for Artificial Intelligence

European Network of Excellence in
Computational Logic (COLOGNET)

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
17-05-2002	Conference Proceedings	2 May 2002 - 4 May 2002

4. TITLE AND SUBTITLE

Thirteenth International Workshop on Principles of Diagnosis (DX-2002)

5a. CONTRACT NUMBER
F61775-02-WF032

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Conference Committee

Editors: M. Stumptner and F. Wotawa

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

TU Wien
Favoritenstrasse 9-11
Vienna, A-1040
Austria

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
CSP 02-5032

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Final Proceedings for Thirteenth International Workshop on Principles of Diagnosis (DX-2002), 2 May 2002 - 4 May 2002

This workshop is based on the area of artificial intelligence. The focus of this year's event is on theories, principles and computational techniques for diagnosis, testing, reconfiguration and repair of complex systems. Specific topics of interest to the Air Force include: formal methods and computational methods, modeling (symbolic, numeric, discrete, continuous, and hybrid discrete/continuous), computational issues like combinatorial explosion, the diagnosis process, machine learning, nonmonotonic reasoning, and applications and technology transfer.

15. SUBJECT TERMS
EOARD, Modelling & Simulation, Software, Fault Detection and Correction, Computational Methods

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Christopher Reuter, Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL	191 (plus front matter)	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)20 7514 4474

Thirteenth International Workshop on Principles of Diagnosis (DX-02)

Conference Committee

Program Chairs

Markus Stumptner	*University of South Australia, Adelaide*
Franz Wotawa	*TU Graz, Austria*

Program Committee

Bert Bredeweg	*University of Amsterdam, Netherlands*
Marie-Odile Cordier	*Irisa, France*
Mireille Ducassé	*Irisa, France*
Daniele Theseider Dupré	*Universitá Piemonte Orientale, Italy*
Yousri El Fattah	*Rockwell, USA*
Jan Lunze	*Ruhr-Universität Bochum, Germany*
Pieter J. Mosterman	*The MathWorks Inc., USA*
Chris Price	*University of Wales, UK*
Bernhard Rinner	*TU Graz, Austria*
Meera Sampath	*Xerox, USA*
Howard E. Shrobe	*MIT, USA*
Peter Struss	*Occ'm Software and TU München, Germany*
Patrick Taillibert	*Thales, France*
Mugur Tatar	*DaimlerChrysler, Germany*
Takashi Washio	*Osaka University, Japan*
Marina Zanella	*Universitá di Brescia, Italy*
Feng Zhao	*Xerox PARC, USA*

Foreword

The Thirteenth International Workshop on Principles of Diagnosis (DX 02) is the latest in a series of annual workshops that focus on the presentation and exchange of current results in the field of diagnosis and related areas, including tasks such as monitoring, fault identification and isolation, testing, reconfiguration and repair. The workshops are historically centered on approaches from the Artificial Intelligence (AI) community, but aim at supporting wide range of different techniques and methodologies, as well as the integration of other research communities such as Process Engineering and FDI.

The papers included in this volume span a wide range of techniques and application areas, including such domains as complex hardware systems, software and knowledge bases, secure systems, and design problems, and deal with discrete and continuous, algebraic, logic-, constraint-, structure-, and probability-based approaches, dynamic and temporal systems, distribution and abstraction, and non-symbolic methods of diagnosis. They bear witness to the continuing existence of fertile ground for further theoretical and applied research.

The invited talks continue the choice of earlier workshops to bring in new and varying viewpoints to provide a wider context to the problem area, and address issues from related and neighboring areas of interest to the diagnosis community: constraint satisfaction, problem decomposition, and debugging.

We wish to thank the authors of the submitted papers, the program committee members, at least two of which reviewed each of the submitted full papers, for the time and effort spent, and the invited speakers for their participation. We especially wish to thank Sheila McIlraith for her help in organizing the review process.

We would also like to acknowledge the support of our sponsors for their contribution to the success of this workshop:

- AVL DiTEST
- COLOGNET
- European Office of Aerospace Research and Development, Air Force Office of Scientific Research, United States Air Force Research Laboratory
- Materna
- OCC'M Software
- Austrian Computer Society (OCS)
- Austrian Artificial Intelligence Society (OEGAI)
- Siemens Austria, PSE PRO, CES Design Services
- TU Graz, IICM - Software Technology
- TU Wien, Institut für Informationssysteme

Markus Stumptner and Franz Wotawa April, 2002

Content

Papers

Particle Filters for Real-Time Fault Detection in Planetary Rovers

Richard Dearden and Dan Clancy
Research Institute for Advanced Computer Science
NASA Ames Research Center
Mail Stop 269-3 Moffett Field, CA 94035 USA
Email:dearden,clancy@ptolemy.arc.nasa.gov

Abstract.

Planetary rovers provide a considerable challenge for artificial intelligence in that they must operate for long periods autonomously, or with relatively little intervention. To achieve this, they need to have on-board fault detection and diagnosis capabilities. Traditional model-based diagnosis techniques are not suitable for rovers due to the tight coupling between the vehicle's performance and its environment. Hybrid diagnosis using particle filters is presented as an alternative, and its strengths and weaknesses are examined. We also present some extensions to particle filters that are designed to make them more suitable for use in diagnosis problems.

1 Introduction

Planetary rovers provide a considerable challenge for artificial intelligence in that they must operate for long periods autonomously, or with relatively little intervention. To achieve this, they need (among other things) to have on-board fault detection and diagnosis capabilities in order to determine the actual state of the vehicle, and decide what actions are safe to perform. However, as we will discuss below, traditional approaches to diagnosis are unsuitable for rovers, and we must turn to hybrid approaches. In this paper we describe an approach to hybrid diagnosis based on *particle filters* [2, 7, 3]. We show that the characteristics of diagnosis problems present some difficulties for standard particle filters, and describe an approach for solving this problem. We will use rovers as a motivating example throughout this paper, but the techniques we describe can be applied to any hybrid diagnosis problem.

The diagnosis problem is to determine the current state of a system given a stream of observations of that system. In traditional model-based diagnosis systems such as Livingstone [14], diagnosis is performed by maintaining a set of candidate hypotheses (in Livingstone, a single hypothesis was used) about the current state of the system, and using the model to predict the expected future state of the system given each candidate. The predicted states are then compared with the observations of what actually occurred. If the observations are consistent with a particular state that is predicted, that state is kept as a candidate hypothesis. If they are inconsistent, the candidate is discarded. Traditional diagnosis systems typically use a logic-based representation, and use *monitors* to translate continuous-valued sensor readings into discrete-valued variables. The system can then reason about the discrete variables, and compare them with the predictions of the model using constraint propagation techniques.

Unlike the spacecraft domains that Livingstone has been applied in, rover performance depends significantly on environmental interactions. The on-board sensors provide streams of continuous valued data that varies due to noise, but also due to the interaction between the rover and its environment. For example, a rover may have a sensor that reports the current drawn by a wheel. In normal operation, this quantity may vary considerably, increasing when the vehicle is climbing a hill, and decreasing on downward slopes. The diagnosis system needs to be able to distinguish a change in the current drawn due to the terrain being traversed from a change due to a fault in the wheel. A second issue for rovers is that their weight and power is very tightly constrained. For this reason, any on-board diagnosis system must be computationally efficient, and should be able to adapt to variations in processor availability. Ideally, we would also like it to adapt based on its own performance, spending more time on diagnosis when a fault is likely to have occurred, and less time when the system appears to be operating normally.

A rover's close coupling with its environment poses a considerable problem for diagnosis systems that use discrete models. A particular sensor reading may be normal under certain environmental conditions, but indicative of a fault in others, so any monitor that translates the sensor reading into a discrete value such as "nominal," or "off-nominal high" must be sophisticated enough to take all the environmental conditions into account. This can mean that the diagnosis problem is effectively passed off to the monitors—the diagnosis system is very simple, but relies on discrete sensor values from extremely complex monitors that diagnose the interaction between the system and its environment as part of translating continuous sensor values into discrete variables. To overcome this problem, we need to reason directly with the continuous values we receive from sensors. That is, our model needs to be a hybrid system, consisting of a set of discrete modes that the system can be in, along with a set of continuous state variables. The dynamics of the system is described in terms of a set of equations governing the evolution of the state variables, and these equations will be different in different modes. In addition, a transition function describes how the system moves from one mode to another, and an observation function defines the likelihood of an observation given the mode and the values of the system variables.

This hybrid model can be seen as a partially observable Markov decision process (POMDP) [1]. POMDPs are frequently used as a representation for decision-theoretic planning problems, where the task is to determine the best action to perform given the current estimate of the actual state of the system. This estimate, referred to as

the belief state,is exactly what we would like to determine in the diagnosis problem, and the problem of keeping the belief state updated is well understood in the decision theory literature. The belief state is a probability distribution over the system states—that is, for every state it gives the probability of being in that state, given our prior beliefs about the state of the system, and the sequence of observations and actions that have occurred so far.

Unfortunately, maintaining an exact belief state is computationally intractable for the types of problem we are interested in. Since our model contains both discrete and continuous variables, the belief state is a set of multidimensional probability distributions over the continuous state variables, with one such distribution for each mode of the system. These distributions may not even be unimodal, so just representing the belief state is a complex problem, but updating it when new observations are made is intractable for hybrid models in all but the simplest of models (see [8] for an illustration of this). Therefore, an approximation needs to be made. As we said above, we will use a *particle filter* to approximate the belief state and keep it updated.

A particle filter represents a probability distribution using a set of discrete samples, referred to as particles, each of which has an associated weight. The set of weighted particles constitutes an approximation to the belief state, and has the advantage over other approaches such as Kalman Filters [6] that it can represent arbitrary distributions. To update the distribution when a new observation is made, we treat each particle as a hypothesis about the state of the system, apply the model to it to move it to a new state, and multiply the weight of the particle by the likelihood of making the observation in that new state. To prevent a small number of particles from dominating the probability distribution, the particles are then resampled, with a new set of particles, each of weight one, being constructed by selecting samples randomly based on their weight from the old set.

Particle filters have already proven very successful for a number of tasks, including visual tracking [7] and robot navigation [4]. Unfortunately, they are less well suited to diagnosis tasks. This is because the mode transitions that we are most interested in detecting namely transitions to fault states typically have very low probability of actually occurring. Thus, there is a risk that there will be no particle in a fault state when a fault occurs, and the system will be unable to diagnose the fault. We propose a solution to this problem by thinking of a particle filter as a convenient way to divide the computation time that is available for doing diagnosis between the candidate states that the system could be in. A conventional particle filter splits the particles (and hence the computation time) according to how well the states predict the observations, but with this approach we will also spend some computation time on fault states that are important to diagnose. We do this simply by ensuring that there are always some particles in those states. As we will show, the details of the particle filter algorithm mean that we can add these additional particles without biasing the diagnosis that results.

In the next section we discuss the hybrid model of the rover in detail. In Section 3 we describe particle filtering and demonstrate its weaknesses when applied to diagnosis problems, and in Section 4 we will describe our modifications to the standard particle filter in detail. In Section 5 we present some preliminary results on real rover data, using a simple version of our proposed approach. The final section looks at the relationship between this work and some previous approaches to this problem, and discusses some future directions for this work.

2 Modeling a Planetary Rover

As we said above, we model a rover as a hybrid system. The discrete component of the rover's state represents the various operational and fault modes of the rover, while the continuous state describes the speed of the wheels, the current being drawn by various subsystems, and so on. Following [13], our rover model consists of a tuple $\langle M, V, T, E, O \rangle$ where the elements of the tuple are as follows:

- M is the set of discrete modes the system can be in. We assume that M is finite, and write m for an individual system mode.
- V is the set of variables describing the continuous state of the system.
- T is a transition function that defines how the system moves from one mode to another over time. We write $\Pr_T(m, m')$ for the probability that the system moves from mode m to mode m' . We may also include a second transition function $\Pr_T(m, a, m')$ which is used when an action a occurs. This gives the probability of moving from m to m' when action a is executed.
- E is a set of equations that describe the evolution of the continuous variables over time. The equations that apply at a given time potentially depend on the system mode, so we write E_m for the equations that apply in mode m. These equations will in general include a noise term to account for random variations in the state variables. Here we will assume Gaussian noise, with the parameters of the Gaussian determined individually for each equation.
- O is a function mapping the system state into observations. We will assume that the observable system characteristics are some subset of the system variables V, with their values corrupted by Gaussian noise (again with parameters that may be a function of the variable, and the system mode), so we write $O(v, m)$ for the observed value of some variable v in mode m.

We will also write $\Pr(s'|s)$ for the probability distribution over future states s' given some state s,where s and s' are hybrid states, so $\Pr(s'|s)$ includes both the distribution over the future mode given by $\Pr T(m'|m)$,and the distributions over the continuous variables given by E_m.

The diagnosis problem now becomes the task of determining the current mode m that the system is in, and the values of all the state variables in V (the results we will present will only show the probability distribution over discrete modes, but the algorithm produces a distribution over the full hybrid state).

The experiments we will present in Sections 3 and 5 use actual telemetry data from NASA Ames Marsokhod rover. The Marsokhod is a planetary rover built on a Russian chassis that has been used in field tests from 1993–99 in Russia, Hawaii, and the deserts of Arizona and California. The rover has six independently driven wheels, and for the experiments we present here, the right rear wheel had a broken gear, and so rolls passively. The Marsokhod has a number of sensors, but we will restrict our attention to diagnosing the state of the broken wheel, and will therefore use only data from the wheel current and wheel odometry sensors. We will treat each wheel independently in the diagnosis. For each wheel, we have a model, taken from [13], with the following characteristics:

- M consists of 23 system modes of which 14 are fault states.
- V consists of variables for the wheel current and wheel speed, and the derivatives of current and speed.
- T is a fairly sparse matrix, with at most six successors for any given mode. The probability of a transition to a fault state is 0.01 or less. All commands are described by one transition function

for the start and one for the end of a command because the data doesn't identify which command occurred.
- The state equations in E consist of the previous value plus a constant term and noise. The noise is Gaussian with standard deviation in the range 0.001 to 1.0, and the equations are independent for each state variable.
- The equations in O are independent for each variable (but vary depending on the mode), and include a Gaussian noise term with a standard deviation that varies from 0.001 to 1.0.

3 Particle Filters

A particle filter approximates an unknown probability distribution using a weighted set of samples. Each sample or particle consists of a value for every state variable, so it describes one possible complete state the system might be in. As observations are made, the transition function is applied to each particle individually, moving it stochastically to a new state, and then the observations are used to re-weight each particle to reflect the likelihood of the observation given the particle's new state. In this way, particles that predict the observed system performance are highly weighted, indicating that they are in likely states of the system. A major advantage of particle filters is that their computational requirements depend only on the number of particles, not on the complexity of the model. This is of huge importance to us as it allows us to do diagnosis in an anytime fashion; increasing or decreasing the number of particles depending on the available computation time.

To implement a particle filter, we require three things:

- A probability distribution over the initial state of the system.
- A model of the system that can be used to predict, given the current state according to an individual particle, a possible future state of that particle. Since T is stochastic, and E includes noise terms, the predictive model selects a new state for the particle in a Monte Carlo fashion [5], choosing by sampling from the probability distribution over possible future states.
- A way to compute the likelihood of observing particular sensor values given a state. In our case, this is given by the observation function O.

The particle filtering algorithm is given in Figure 1. Step (i) is the *predictive step*, where a new state is calculated in a Monte Carlo way for each particle, and this new state is then conditioned on the observations in step (ii) (we call this the *re-weighting step*). The predictive step is performed by applying T to each particle, and then applying the appropriate equations from E to the state variables, sampling values from the Gaussian error terms. Once the particles have been re-weighted, we can then calculate the probability of each mode simply by summing the weights of the particles in the mode. We refer to step (b) as the *resampling step*. For more details on the properties of particle filters see e.g. [3].

3.1 Problems with Particle Filters for Diagnosis

Unfortunately, there are a number of difficulties in applying particle filters to diagnosis problems. In particular, the filter must have a particle in a particular state before the probability of that state can be evaluated. If a state has no particles in it, its probability of being the true state of the system is zero. This is a particular problem in diagnosis problems because the transition probabilities to fault states are typically very low, so particles are unlikely to end up in fault states during the Monte Carlo predictive step. This situation is known as *sample impoverishment*.

1. Create a set of n particles where each particle p_i has a state s_i and a weight w_i. s_i is sampled randomly from the prior state distribution, and $w_i = 1$.
2. For each time step, do:
   (a) Replace each particle p_i with p'_i as follows:
      i. Select a future state s'_i by sampling from $\Pr(s'_i|s_i)$, the distribution over possible future states given by the model.
      ii. Re-weight p_i by multiplying its weight by the probability of the observations o given s'_i as follows:

$$w'_i = \Pr(o|s'_i)w_i$$

   (b) Resample n new particles $p_1, \ldots, p_n$ by copying the p' current particles where each particle p'_j is added to the new samples with the following probability:

$$\Pr(p_i = p'_j) = \frac{w'_j}{\sum_{k=1}^{n} w'_k}$$

Figure 1. The particle filtering algorithm.

Figure 2 illustrates this problem. Each graph shows the most likely modes that the wheel is in (the y-axis is the total weight of the particles in each mode, so a value of 10000 implies that all particles are in that mode), shown over part of one of the trials in which the wheel is initially idle, and then at step 12 is commanded to drive forward at a fixed speed. The graphs on the left show the performance of Wheel 1, which is operating nominally. The graphs on the right show the performance of Wheel 6, which is faulty. In the top line, the probability of the fault occurring is 0.1 rather than its true value of 0.01. Here the fault is quickly detected in Wheel 6. In the bottom line of graphs, the fault probability is set to its true value, and in this case the fault is not successfully detected because insufficient particles enter the fault state. One might expect that once a particle enters the fault state its weight would be high since it would predict well, and at the resampling step it should lead to several new particles being created. Unfortunately, this did not occur in this situation because although some particles did enter the fault state, their continuous parameter values did not agree with the observations well, so they still had low weights. The continuous parameters did not match because each of the particles that entered the fault state came from the COMMANDEDRUNNING state, in which the current and wheel speed are expected to be much higher than the observed values.

4 Importance Sampling

The simplest solution to the sample impoverishment problem is to increase the number of particles being used. Given the constraints imposed on on-board systems, this approach is probably unrealistic. The data presented above was implemented in Java, using 10,000 particles per wheel, and runs in approximately 0.5 seconds per update on a 750MHz Pentium 3. This is probably at the upper limit of the number of particles we could expect to use on-board a rover as the time available for diagnosis is longer, but there will be less computation devoted to diagnosis. Thus running with ten times as many particles (which is roughly equivalent to multiplying the fault

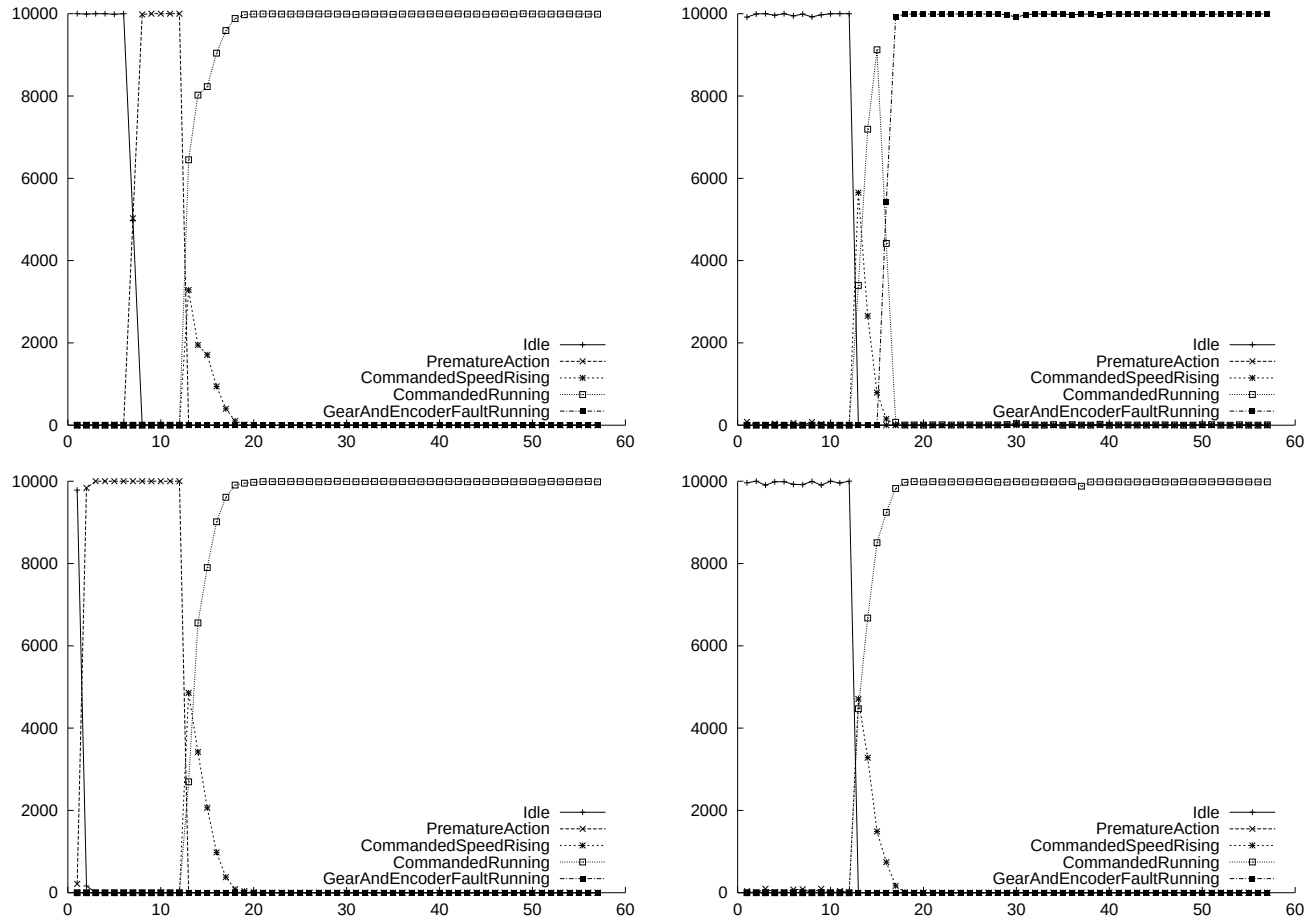


Figure 2. Results for Wheel 1 (left side) and Wheel 6 (right side). In the top row, the probability of a fault is ten times its true value, while the bottom row uses the true probabilities. The fault (GEARANDENCODERFAULTRUNNING) in Wheel 6 is quickly detected in the top row, but is never discovered in the bottom row due to sample impoverishment.

probability by ten) is probably impractical on the rover, and even 10,000 particles may be unrealistic as the model gets more complex. This could be somewhat overcome by only increasing the number of particles when there is some evidence that the system is predicting poorly. In order to achieve this, we need some measure of when this occurs. The obvious measure is to look at the total weight of the particles after conditioning on the observations. If no particles are predicting the observations well the total weight should drop. Unfortunately, in practice this is rarely useful because there are a number of other possible causes for this behavior. For example, particles moving from a state in which there is high confidence in the sensor readings to a state with more sensor noise will tend to drop in weight even if they are still predicting the observations well. We see this in the Marsokhod model because the IDLE mode has relatively large variance for the observation noise, whereas the COMMANDEDRUNNING mode has smaller variance, so the total particle weight increases when the system moves from the IDLE to the COMMANDEDRUNNING mode, even for wheel 6 where COMMANDEDRUNNING predicts the observations poorly.

Another way to reduce the likelihood of sample impoverishment is to take advantage of some results from *importance sampling* (see e.g. [9]). In importance sampling, we want to sample from some distribution P, but we can't. Instead, we sample from some other distribution Q, and weight each sample s by $\Pr_P(s)/\Pr_Q(s)$, the ratio of the likelihood of sampling s from P to the likelihood of sampling it from Q. The weighted sample is then an unbiased sample from P, as long as Q is non-zero everywhere that P is non-zero. In fact, importance sampling is exactly what the particle filter algorithm is doing. For a particle filter, the unknown distribution P is the posterior distribution we are trying to compute, Q is the prior distribution (the set of samples before the observation), and the re-weighting step corresponds to the importance sampling weight computation.

Given that whatever we choose for Q, the weighted samples are an unbiased sample from P, we can add arbitrary samples to our particle filter at the resampling step, and still end up with an unbiased posterior distribution. We will use this property to ensure that we have samples in the system modes that are important to us (hence the name importance sampling). The question then is how to choose Q. We can imagine an oracle that provides a set of candidate states the system might end up in, given the current distribution over state. When we resample, we simply make sure that there are always some particles in the states provided by the oracle. If those states explain the subsequent observations well, the particles in them will get high weight, and are likely to be resampled with more particles at the next

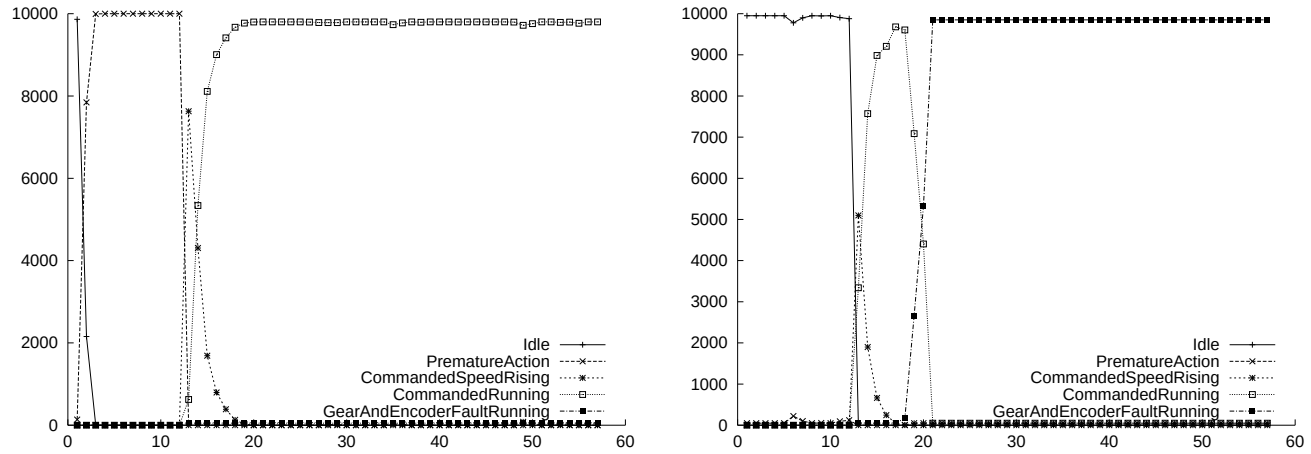


Figure 3. Results for the importance sampled particle filter. All states with $> 25\%$ probability were used as starting points for the forward search, and 0.5% of the particles were assigned to each of the found states. On the left are results for wheel 1, and on the right for wheel 6.

step. On the other hand, if they predict the observations poorly, the particles will quickly disappear again.

The question that remains is how to implement the oracle. For a complex system such as a planetary rover, with many components each with its own set of possible failure modes, there are exponentially many possible failure modes, so this is a non-trivial problem. However, one approach that seems promising is to use a traditional model-based diagnosis system such as Livingstone [14]. These discrete systems typically operate more quickly than hybrid approaches, so can be used to suggest hypotheses without significantly affecting computation time. We pointed out in the introduction that they are not in general suitable for diagnosing rovers, but they could be used to identify sets of likely system modes for the hybrid system to be used in. The integration of Livingstone with the particle filter approach is currently work in progress, as it adds a number of additional complications including building an additional system model, and ensuring that the discrete and hybrid models agree with one another and can easily be translated back and forth.

For simpler systems such as the Marsokhod wheel diagnosis we have used in this paper, the above approach is unnecessary. Instead, we can use an oracle based on forward search from the current high-probability states. Since each system mode in this model has at most six possible successors, and there are typically only two to three high probability modes at any time, we find in practice that in most cases a simple one-step look-ahead search adds fewer than five modes to those that already contain particles.

5 Results

The results we present here are based on the Marsokhod model we described in Section 2. Dr. Rich Washington supplied the model and the data, which came from his work on using Kalman filters for rover diagnosis [13]. The only changes made to the model were to make it suitable for use with a particle filter; no changes were made to model parameters or transition probabilities. To demonstrate our approach we use a small piece of one of the telemetry data files (the same piece used in Section 3) in which the rover is initially idle, and then a drive command is issued, resulting in an increase in current to each wheel, followed by a corresponding increase in speed, and then a constant speed. As before, wheel 6 is faulty, with a broken gear (this corresponds to the GEARANDENCODERFAULTRUNNING state in the model).

Figure 3 shows the results for the importance sampled particle filter. We used single step forward search from all states with probability > 0.25 to select the set of bias states. Each of these states was then guaranteed to receive at least 0.5% of the total number of particles at each re-sampling step. The left hand graph is the probable states for Wheel 1, as before. Like the graph in the bottom row of Figure 2, the PREMATUREACTION state was given high probability before step 13. This state appears where the effects of an action are seen before the signal to perform the action, due to problems with the rover telemetry. In this case it is a spurious result due to the model of the IDLE state not allowing sufficient noise in the observations. A small adjustment to the model would remove this problem, which is only present in the data for two of the wheels. The right hand graph shows the same data for Wheel 6. In this case, the fault state dominates the probability distribution after step 20, seven steps after the command to drive the wheel was observed, as compared with three steps for the model with increased fault probabilities (Figure 2).

6 Discussion and Relation to Other Work

An important thing to note is that standard particle filters treat the model essentially as a black box, using it only to predict future states of the system. We have described one approach that exploits the structure in the model to make diagnosis using particle filters more effective. This is by no means the only way to exploit that structure, and we are in the process of looking at other techniques. Possibilities include making a single-fault assumption (but relaxing it if it predicts the observations poorly), and taking advantage of independence between components in the system to reduce the number of samples needed, or even to diagnose subsystems independently.

One closely related piece of work is Verma et al.'s decision-theoretic particle filter [11]. Their approach is similar to ours, but they assign a utility—which corresponds to how important each state is to diagnose—to every state and multiply the probability of a transition by the utility of the state that results. This alters the transition function to favor important states, rather like the approach we took

in Figure 2. For relatively simple diagnosis tasks such as the one we have presented here, the approaches seem very similar. However, designing a utility function to produce the right diagnoses, without causing too many false diagnoses of faults is a difficult task, especially as any reasonable utility function would give all fault states a high utility. In [10], they refine this approach, again using a *risk function* that scores states by how important it is to diagnose them correctly, but this time modifying the particle filter algorithm so that the samples are distributed according to the product of the posterior probability distribution and the risk factor. This ensures that samples in high-risk states have higher weights, and the true posterior can be recovered from the risk-sensitive posterior, but still suffers from the problem of sample impoverishment. These approaches are orthogonal to the approach described here, and we are currently working on combining the two.

Another related effort is the work of Washington [13] that applies Kalman Filters to this problem. In this work, the continuous dynamics in each mode is tracked by a set of Kalman filters. The main problem with the approach is that the number of filters tends to increase over time because each time a transition is made to a state the initial conditions for the filter are different, and filters with different initial conditions cannot be combined. This is not a problem for particle filter-based approaches because the particle filters can represent arbitrary distributions over the parameter values, so particles entering a state with two different sets of initial conditions will form a bi-modal distribution. As we said above, we used the model and data from this paper in our own work. We see fewer errors in the mode identification with our approach than in Washington's paper, although we are sometimes slower to identify the fault, and our computational requirements are somewhat higher.

As we said in the introduction, this paper is intended as a proof of concept. There is still much work to do on the problem of how to integrate a model from Livingstone with this system to act as an oracle. We have demonstrated that a simple look-ahead search performs quite well, but this is clearly inadequate for large diagnosis problems, particularly as most faults can occur at any time, so the one step lookahead is unlikely to scale to very large problem. We are also examining a number of other approaches to improving diagnosis with particle filters, such as backtracking when prediction is poor, and re-sampling past states based on observations that occurred more recently. Finally, we are investigating how a diagnosis system of this type would fit with the CLARAty rover architecture [12] being used for future NASA missions to Mars.

REFERENCES

[1] Dimitri P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs, 1987.

[2] A. Doucet, 'On sequential simulation-based methods for Bayesian filtering', Technical Report CUED/F-INFENG/TR.310, Department of Engineering, Cambridge University, (1998).

[3] *Sequential Monte Carlo in Practice*, eds., Arnaud Doucet, Nando De Freitas, and Neil Gordon, Springer-Verlag, 2001.

[4] Dieter Fox, Wolfram Burgard, and Sebastian Thrun, 'Markov localization for mobile robots in dynamic environments', *Journal of Artificial Intelligence Research*, **11**, 391–427, (1999).

[5] *Markov Chain Monte Carlo in Practice*, eds., W. R. Gilks, S. Richardson, and D. J. Spigelhalter, CRC Press, 1996.

[6] M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice*, Prentice Hall, 1993.

[7] M. Isard and A. Blake, 'CONDENSATION: Conditional density propagation for visual tracking', *International Journal of Computer Vision*, (1998).

[8] Uri Lerner, Ron Parr, Daphne Koller, and Gautam Biswas, 'Bayesian fault detection and diagnosis in dynamic systems', in *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, (2000).

[9] B. D. Ripley, *Stochastic Simulation*, Wiley, New York, 1987.

[10] Sebastian Thrun, John Langford, and Vandi Verma, 'Risk sensitive particle filters', in *Neural Information Processing Systems (NIPS)*, (December 2001).

[11] V. Verma, J. Langford, and R. Simmons, 'Non-parametric fault identification for space rovers', in *International Symposium on Artificial Intelligence and Robotics in Space (iSAIRAS)*, (2001).

[12] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, 'The CLARAty architecture for robotic autonomy', in *Proceedings of the 2001 IEEE Aerospace Conference*, Big Sky, Montana, (March 2001).

[13] Rich Washington, 'On-board real-time state and fault identification for rovers', in *Proceedings of the IEEE International Conference on Robotics and Automation*, (April 2000).

[14] Brian C. Williams and P. Pandurang Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 971–978, Portland, Oregon, (1996). AAAI Press / The MIT Press.

Hybrid Modeling and Diagnosis in the Real World: A Case Study

Sriram Narasimhan, Gautam Biswas, Gabor Karsai & Tivadar Szemethy
EECS Dept. & ISIS, Vanderbilt University
Box 1824, Station B, Nashville, TN 37235.
{nsriram, biswas, gabor, tiv}@vuse.vanderbilt.edu

Tim Bowman, Mark Kay & Kirby Keller
The Boeing Company
MC S111-1335, Box 516
St. Louis, MO 63166.
{timothy.e.bowman, mark.c.kay,kirby.j.keller}@boeing.com

Abstract

Applying model-based diagnosis techniques to systems that exhibit hybrid behavior presents an interesting set of challenges that mostly revolve around interactions of the continuous and discrete components of the system. In many real world systems, the overall physical plant is inherently continuous, but system control is performed by a supervisory controller that imposes discrete switching behaviors by reconfiguring the system components, or switching controllers. In this paper, we present a case study of an aircraft fuel system, and discuss methodologies for building system models for online tracking of system behavior and performing fault isolation and identification. Empirical studies are performed on detection and isolation for a set of pump and pipe failures.

1 Introduction

Most present-day systems that we use are designed to be repairable. Failures. either physical (hardware) or logical (software), and the resulting maintenance are a fundamental part of the economics of ownership. Fault diagnosis involves the detection of anomalous system behavior and the isolation and identification of the cause for the deviant behavior. When the system includes safety-critical components, failures or faults in the system must be diagnosed as quickly as possible, and their effects compensated for so that control and safety can be maintained. The term, *diagnostic capabilities*, refers to the abilities of a system to detect a failure and isolate it to a failed unit. Quick detection and isolation allows for quick corrective actions that may include reconfiguration of system functions to prevent damage and maintain control.

Fault accommodation requires tight integration of *online* fault detection, isolation, and identification with the system control loop that may be designed to take appropriate control actions to mitigate the effect of the faults and help maintain nominal system operation. Failure to detect faults reduces system availability, results in failed or incomplete missions, and, in some cases, may even lead to catastrophic failures that lead to loss and destruction of the system. Therefore, fault diagnosis is critical to achieving system performance and life-cycle cost objectives.

In general, systems are *dynamic*, i.e., their behavior changes over time. Faults impose additional *transients* on the dynamic behavior, but that may be difficult to detect and characterize, especially in the presence of model disturbances and noisy measurements. Moreover, in physical systems natural feedback from the system and controller actions may mask the transient behavior if they are not detected soon after they occur. This motivates the development and use of online model-based fault detection and isolation methods. Model-based techniques employ a model to predict nominal system behavior. The model must be constructed at a level of detail where system behavior can be mapped to system components and parameters. The relations in the model are employed to map observed deviations between measurements and values predicted by the model to possible faults in system components. Continued monitoring helps establish a unique fault or set of faults associated with the system.

Most real-life systems are equipped with a limited number of sensors to track system behavior, and analytic redundancy methods have to be applied to derive non-local interaction between potential faults and observations. These techniques have been applied to a variety of schemes used in the diagnosis of discrete [deKleer and Williams, 1987], discrete event [Lunze, 1999; Sampath *et al.*, 1996] and continuous systems [Gertler, 1997; Mosterman and Biswas, 1999]. The traditional approach to hybrid system diagnosis has been to use a single continuous model with complex non-linearities, or abstracting the continuous dynamics to a discrete event model. Complex non-linearities complicate the analysis and they may introduce numerical convergence problems. Discrete event abstractions lead to loss of critical information, such as fault transient characteristics. Further, methods to identify the set of events that describe both nominal and faulty behavior is often a computationally challenging task bringing to question the scalability of such approaches. Hybrid system analyses require the use of multiple models of the system. Recent approaches to hybrid system diagnosis have incorporated appropriate model selection and mode estimation techniques at run time to track faulty behavior and perform fault isolation [McIlraith *et al.*, 2000; Hofbaur and Williams, 2002; Narasimhan and Biswas, 2001; 2002].

Model-based diagnosis techniques can only be as good as the models upon which they are based. Incomplete and incor-

rect models cause problems with the tracking and fault isolation tasks. The tracking process may produce false alarms or worse missed alarms. In the first case, diagnosis is triggered when there is no fault in the system. In the second situation, diagnosis is not triggered and a fault may be missed. Fault isolation with incomplete and inaccurate models may also produce false candidates and miss true candidates. On the other hand, building models that include unnecessary detail may increase computational complexity making online processing infeasible. Therefore, a key task in model-based diagnosis is to build accurate models at the right level of detail. This paper focuses on the pragmatics of model building and fault isolation by performing a case study on the fuel transfer system of an aircraft.

2 Fuel System Description

High-performance aircraft require sophisticated control techniques to support all aspects of operation, such as flight control, mission management, and environmental control. An aircraft's fuel transfer system maintains the required flow of fuel to the engines through different modes of operation, while ensuring that imbalances are not created that affect center of gravity of the system. Fig. 1. illustrates a typical fuel system configuration. The fuel system geometry is symmetric and may be split into left side and right side arrangements. The overall system can be divided into two main sub-systems: (i) *the engine feed system*, and (ii) *the transfer system*. The feed system consists of a left and right engine feed tank. The tanks are connected through pipes with controlled valves so that fuel can be transferred between the tanks if a fault occurs in one of the tanks. A boost pump in each of the feed tanks controls the supply of fuel from the tank to its respective engine. The transfer system moves fuel from the two forward fuselage and the two wing tanks to the engine feed tanks. The intent is to keep the engine feed tanks near full at all times so that sufficient fuel is available on demand, and if failures occur in the transfer system there is still a significant amount of fuel available for emergency maneuvers. The fuel transfer sequence is set up in a way that maintains the aircraft's center of gravity. To achieve this, pumps located in the fuselage and wing tanks are are turned on in a pre-determined sequence to transfer their fuel to a common transfer manifold (set of tubes). The fuel exits the transfer manifold through level control valves into the feed tanks.

A wide variety of sensors may be included in the fuel transfer system. Fuel quantity gauging sensors determine the amount of fuel in a tank. Engine fuel flow meters determine engine fuel consumption. Pressure transducers measure the transfer and boost pump pressures. Position sensors determine the open and closed states of valves. Each sensor comes at a cost that is determined by its weight, reliability, complexity, and cost. Therefore, designers often try to minimize the number of sensors while ensuring that the required control can be achieved.

The transfer system control schedules the pump operation to match a pre-defined transfer sequence shown in Table 1. The unit of the amounts in the table is the *pound*. Initially one wing pump in each tank is turned on. When a feed tank

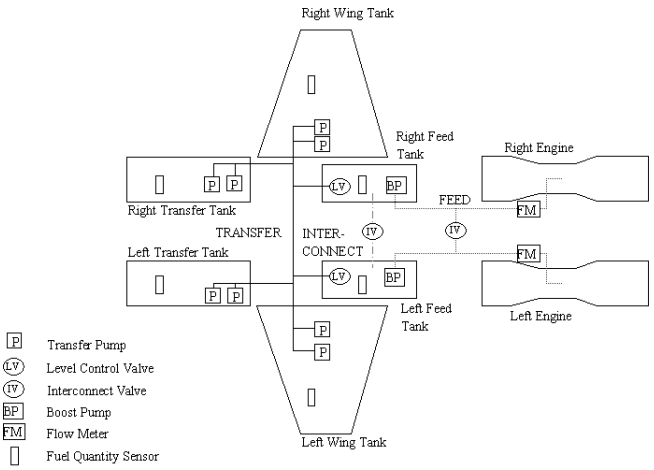


Figure 1: Fuel System Schematic

quantity decreases by *100 lbs*, the level control valve in that tank will be opened. The fuel then flows from the transfer manifold into the feed tank raising its level back to the full fuel quantity at which point the level control valve will be closed, stopping the fuel transfer.

Table 1: Fuel Transfer Sequence

Left Wing Tank	Right Wing Tank	Left Fuselage Tank	Right Fuselage Tank
2500	2500	3300	3000
2000	2000	3300	3000
2000	2000	3000	3000
1000	1000	2000	2000
0	0	1000	1000
0	0	0	0

The most common failures in this configuration are transfer and boost pump failures, and shutoff valve failures. The transfer pumps have two primary failure modes. One is a total loss of pressure caused by the impeller not turning. The other is a degraded state caused by mechanical wear, leakage, or electrical failure where the fuel flow rate falls below nominal values. The second failure can lead to the first condition over time. Faults in the boost pump mirror those in the transfer pumps. Valve failures are *stuck-at* conditions, i.e., their positions do not change even when they are commanded to do so. This can result from mechanical friction/jamming of the shaft or electrical failure of the motor or power source. In this work, we also consider partial failures of the valves. A third class of faults that we consider is leaks in the connecting pipes. Our goal is to develop an online diagnostic system for detection, isolation and identification of these faults.

3 Component-based Hierarchical Modeling for Diagnosis

Complex real-world systems are made up of a number of subsystems and components. Hierarchical component-based approaches, e.g., Statecharts [Harel, 1987], 20sim [van Amerongen, 2000], and Ptolemy [Buck *et al.*, 1994]. are a practical approach to constructing models of such systems, We have developed a new methodology for hierarchical component based modeling that customizes the graphical Generic Modeling Environment (*GME*) with a hybrid bond graph (*HBG*) approach for building hybrid models of physical systems with supervisory controllers. This section reviews our approach to hybrid bond graph modeling, then presents the *GME* methodology for building component-based models for the aircraft fuel transfer system.

3.1 Hybrid Bond Graphs

Our approach to modeling the fuel system is based on an extended form of bond graphs [Karnopp *et al.*, 1990], called *Hybrid Bond Graphs (HBG)* [Mosterman and Biswas, 1998]. Bond graphs present a methodology for energy-based modeling of physical systems. Generic bond graph components represent primitive processes, such as the energy storage elements, *inertias* and *capacitors*, and dissipative elements, *resistors*. Bonds represent the energy transfer pathways in the system. Junctions, which are of two types: 1 or *series*, and 0 or *parallel*, define the component interconnectivity structure, and impose energy conservation laws. Overall, the bond graph topology implies system behavior that combines individual component behaviors based on the principles of continuity and conservation of energy.

Extensions to hybrid systems require the introduction of discrete changes in the model configuration. In the *HBG* framework, discontinuities in behavior are dealt with at a meta level, where the energy model embodied in the bond graph scheme is suspended in time, and discontinuous model configuration changes are modeled to occur instantaneously. Therefore, the meta level describes a control structure that causes changes in bond graph topology using idealized switches that do not violate the principles of energy distribution in the system. Topology changes result in a new model configuration that defines future behavior evolution. To ensure physical principles are not violated, we have developed transformations that derive the initial system state in the new configuration from the old one. From this point on behavior evolution is continuous, till another discrete change is triggered at the meta level.

To keep the overall behavior generation consistent, the meta-model control mechanism and the energy-related bond graph models are kept distinct. The switching structure is implemented as localized *switched junctions* that provide a compact representation of the system model across all its nominal modes of operation. Instead of pre-enumerating the bond graph for each mode, the *HBG* uses individual junctions to model local mode transitions. The switched 0- and 1- junctions represent idealized discrete switching elements that can turn the corresponding energy connection on and off. A finite state machine determines the ON/OFF physical state of the junctions. The transitions in this automaton depend on both control signals and internal variable values.

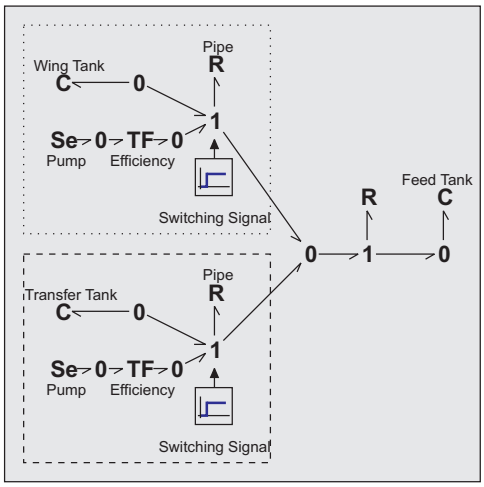


Figure 2: Hybrid Bond Graph Example

Fig. 2 shows the hybrid bond graph model of a portion of the fuel system. The dotted subsystem represents the wing tank, and the dashed subsystem represents the fuselage tank. In this simplified model, the tank system is modeled as a capacitor for storage of fuel, pump system as an effort source to boost the pressure and create an outflow, and pipes as conduits with resistive losses. For this configuration with two switched junctions, the system can be in four different modes. When the two junctions are off, there is no fuel supplied to the feed tank, one of the two tanks (wing or fuselage) can supply fuel to the feed tank, and both tanks may supply fuel to the feed tank at the same time. Switching of configurations is achieved by changing the switching signal values. Suppose the wing tank is supplying fuel, i.e., $signal_1 = 1$ (on) and $signal_2 = 0$ (off). To switch supplying tanks, we simply set $signal_1 = 0$ (off) and $signal_2 = 1$ (on). The state equation model for the new configuration can be easily derived online using a standard algorithm [Karnopp *et al.*, 1990].

3.2 GME

We have developed an approach for building component-based system models using a graphical modeling tool called Generic Modeling Environment(*GME*) [Ledeczi *et al.*, 2001]. *GME* is a configurable toolkit for creating domain-specific modeling and program synthesis environments. The configuration is accomplished through meta-models[1] specifying the modeling paradigm (modeling language) of the application domain. The modeling paradigm contains the syntactic, semantic, and visual presentation information of the domain, such as the concepts that form the building blocks for constructing models, the relationships among these concepts, how the concepts may be organized and viewed by the modeler, and the rules governing the composition of individual concepts and sets of concepts to form component and system

[1]The concept of meta-models in *GME* differs from the meta level switching models in *HBG*.

models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment.

The meta-models specifying the modeling paradigm are employed to automatically generate the target domain-modeling environment, e.g., the *HBG* environment. The generated modeling environment is then used to build domain models that are stored in a model database. The primarily graphical, domain models can be conveniently stored in standard formats including *XML* to be used by specific applications. We have developed a *GME* modeling paradigm that describes the *HBG* modeling environment.

3.3 Hierarchical and Compositional Modeling in the fluid domain

Real life systems with embedded control tend to be complex, and system designers and engineers typically have a lot of difficulty in generating flat models of the entire system. Hierarchical and compositional modeling are useful tools that allow the system to be constructed in a structured way by modeling subsystems independently and composing them to generate system models. The two main steps in this approach are: (i) decomposition into subsystems and building models of subsystems, and (ii) specifying interactions between subsystems and using composition operators to build system models. This approach provides a number of advantages, such as simplicity in model building, independence in building subsystem models, and modularity and reusability of the generated components.

To model the fuel transfer system, we develop models of typical components in the fluid domain, such as tanks, pipes, and pumps. Pipes may include valves that regulate flow. Pumps and valves can be turned on and off. We assume that their switching time constants are much faster than the time constants in the fluid domain. Therefore, pumps and pipes with valves are modeled as hybrid systems. In our *GME* paradigm, subsystems are modeled as *components*. Interactions between the components are specified as relations between their *in-ports* and *out-ports*. Components connected by ports define the system model. The ports can model: (i) energy transfer between components in the bond graph framework, and (ii) communication of information by signals. Signals are assumed to have no energy content.

For this work, we build first order linear models[2]. Tanks are modeled as a bond graph segment with a capacitor connected to a 0 junction. Each tank component can have multiple in-ports and out-ports. In-ports have energy connections (bonds) to the 0 junction and out-ports have bonds from the 0 junction. Ports may be marked as in and out based on a conventional direction for energy flow, but this does not mean that energy cannot flow in the opposite direction. In case there is an energy flow in the opposite direction, the corresponding variable takes on a negative value.

Pipes are modeled as resistors connected to a 1 junction. Pipes have exactly one in port and one out-port that can be connected to ports of other tanks and pipes. The switching on the pipes is achieved by specifying switching signals on the 1 junction connected to the resistor. As discussed earlier, pumps are modeled as an effort source connected to a transformer, which is connected to a 0 junction. Pumps have one out-port for representing the pressure delivered by the pump.

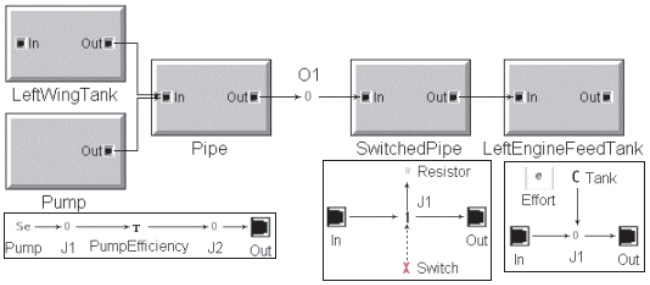


Figure 3: Hierarchical and Compositional Modeling

As an example, the left wing tank is connected to the left feed tank by instantiating two tank components, one pipe component, and one switched pipe component. The switched pipe controls the flow into the feed tank. The out port of the first tank (left wing tank) is connected to the in port of the pipe and out-port of the pipe is connected to the in-port of the second pipe. Since the pump is modeled to pull fuel out of the left wing tank, we connect the out port of a pump component to the in port of the pipe. Fig. 3 illustrates the component based and hierarchical model of this subsystem and the underlying model of the some of the components.

3.4 Modeling for diagnosis

Models form the core component of the tracking and diagnosis algorithms [Biswas and Yu, 1993; Narasimhan *et al.*, 2000]. The hybrid observer uses quantitative *state space models* for tracking nominal system behavior, the fault isolation and identification unit uses temporal causal graphs (*TCG*) for qualitative analysis and input output equation (*IOE*) models for quantitative parameter estimation. We have devised schemes to systematically derive these model representations from the HBG models created in GME.

Precise tracking of nominal system behavior requires the component parameter values in the bond graph model be accurately estimated. We describe our parameter estimation methodology in the next section. For fault isolation and identification, there has to be a a one to one correspondence between faults and parameters in the model. For example, if we abstract a pump model and represent it as an effort source, we cannot differentiate among faults in the electrical versus mechanical/fluid part of the pump. Including a transformer component that models the electrical to fluid energy transformation at an abstract level solves this problem. A partial fault or degradation in the mechanical part of the pump can be attributed to a change in the transformation parameter.

Once the model structure has specified and all parameters have been estimated, the hybrid bond graph model of the complete system is derived by composing the component models and flattening out the hierarchy. The designation of ports as in- and out-ports, and restricting connections to be from out-ports to in-ports only ensures the consistency of bond connections when the components are composed. The

[2]In reality the pressure flow relations are nonlinear.

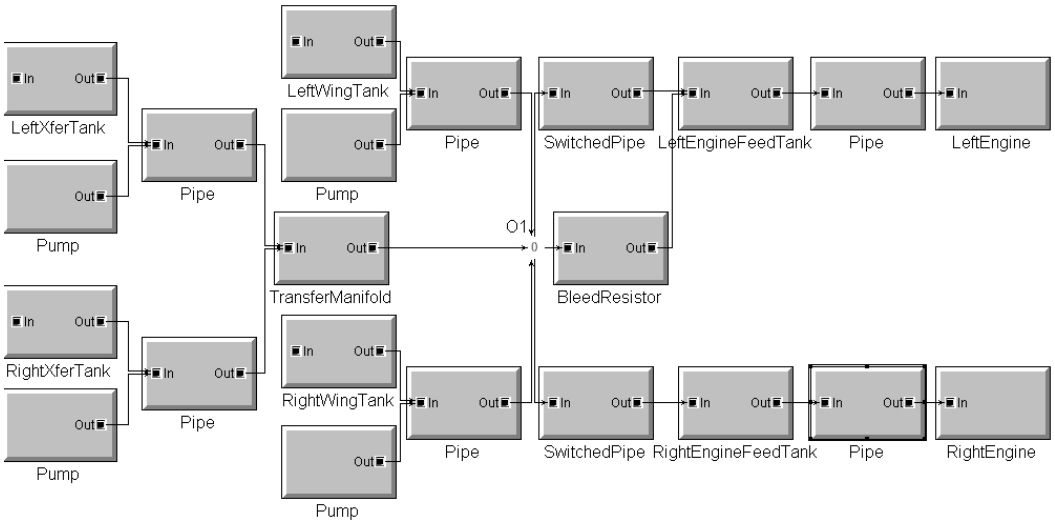


Figure 4: Component Model of Fuel System

resulting hybrid bond graph may be used to systematically derive the state space and temporal causal graph model of the system. In the bond graph framework, each element describes equations that need to be satisfied for that component. For example, for a 0 junction the pressures on all bonds incident is equal and net flow of all bonds is equal to zero. The procedure to convert to state space equations may be summarized as [Karnopp *et al.*, 1990]:

1. Identify state variables (efforts on capacitors and flow on inductors).
2. identify input variables (effort and flow sources).
3. Use constituent equations of the bond graph components to derive the relations between the effort and flow variables in the system.
4. Substitute for all non-state and non-input variables to derive the state equation model. This step is applied repeatedly till all unnecessary variables are eliminated.

The algorithm to derive the TCG from the bond graph is described in [Mosterman and Biswas, 1999].

3.5 Building Models of the Fuel System

From our discussion in earlier sections, the primary model building steps are: (i) identify subsystems and model them at the appropriate level of detail, (ii) compose system models by specifying interactions among the subsystems, and (iii) estimate parameters of the model.

As discussed earlier, tanks, pipes, and pumps are the main components of the fuel system model. In addition, we need to build models for the transfer manifold and the engines. For the scenarios we deal with, it was sufficient to model the engines as constant flow sources, i.e., a sink. Engines have one in-port that represents the flow into the engine from the feed tank. The transfer manifold is modeled as a single capacitor connected to the 0 junction. The transfer manifold has multiple in-ports representing flow into, and multiple out-ports representing flow out of the transfer manifold.

The next step is to determine the complete system configuration. For the fuel system we instantiate 6 tanks: 2 wing tanks, 2 fuselage tanks and 2 engine feed tanks. Each has a corresponding pump. Since the outlets of the wing and fuselage tanks and the inlet of feed tanks all have valves, we created switched pipe components for each of these components. Two instances of the engine are created, and the transfer manifold component is also included in the model. Fig. 4 depicts our component-based GME model of the entire fuel system.

The individual components are connected using bond graph junctions to build the energy model of the overall system. The fuselage tanks supply the transfer manifold, where the flows from the fuselage tanks sum up. This is modeled by connecting one out-port of the fuselage tank to the in-port of a pipe, and the out-port of the pipe to the in-port of the transfer manifold. The pump associated with each tank is also connected to the in port of the pipe. This develops a high pressure at the inlet of the pipe, and hence pulls fuel from the tank into the pipe. The flow from the wing tanks and the transfer manifold combine and distribute evenly to the left and right feed tanks. One out-port of the wing tank is connected to the in port of a pipe. The out-port from these pipes and the out-port from the transfer manifold are connected to a 0 junction to combine the flows. The 0 junction is connected to the in-port of switched pipes whose out-ports are connected to the in-ports of the feed tanks. In order to maintain stability when both feed tanks are closed, a bleed resistor is added to the piping. This resistor bleeds fuel into the left feed tank. The out-ports of the feed tanks are connected to the in-ports of the corresponding engines through pipes.

The next step is to estimate the model parameter values. For the scenarios we model, the engine fuel consumption rate is set at *g gpm* for both engines[3]. All other parameters are estimated from experimental data of an entire fuel burn curve, where all the fuel from the wing and fuselage tanks was con-

[3]In this discussion the actual numbers are not used to avoid any concerns about releasing sensitive data.

sumed by the engines. We used the rate of depletion of fuel in the tanks and the flow out of the tanks when the level control valves are closed to calculate the individual tank capacitances. For the left feed tank the fuel depletion rate is approximately *d lbs/s*, and hence we determine the capacitance of the left feed tank to be $c_l \frac{ft^5 sec^2}{lb}$. Similarly, the right feed tank capacitance is estimated to be $c_r \frac{ft^5 sec^2}{lb}$. We performed similar calculations to determine the wing and fuselage tank capacities (approximately $c_w \frac{ft^5 sec^2}{lb}$). To estimate the resistances, we used the pressure drop and flow through the pipe corresponding to the resistance to calculate the resistance value. The pump effort and efficiency values were given nominal, realistic values.

4 Diagnosis

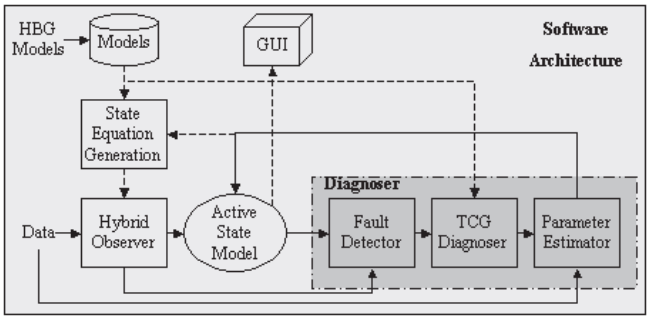


Figure 5: Software Architecture for Diagnosis

The diagnosis task involves tracking dynamic system behavior that includes continuous evolution plus discrete model changes till the fault detector signals a fault. At this point, the fault isolation unit is invoked. Discrete mode changes require dynamic switching of system models, and may also involve discontinuous changes in the system variables. The fault isolation unit also needs to consider change in modes when matching fault signatures with actual system behavior. This motivates the software architecture for diagnosis, illustrated in Fig. 5. The input to the diagnosis system is the model as an *XML* file and the experimental data as a text file. Each line of the data file represents one sample of the data. Although the current version uses a data file as input, replacing it with data from an actual system does not alter the rest of the architecture. Each sample of data includes all input values, all measured output values, and the values of all switching signals. The output of the diagnosis module is the set of diagnostic hypotheses that are consistent with the model and data. The diagnosis output at each time step can be observed in a GUI implemented in Python (*www.python.org*). The active state model (*ASM*) is an internal data structure that maintains information about the system including the current mode, current state estimates, and current diagnostic hypotheses. This structure is updated at each time step from information received from the observer and the diagnosis module. The hybrid bond graph (*HBG*) data structure contains the flattened *HBG* model of the system after composition of all active component subsystems. The *HBG* model also contains the switching conditions for mode changes. These are parsed and stored in the *ASM*. All the diagnosis algorithms modules were implemented in C++. The SWIG toolkit was used for Python-C++ interactions.

The parser reads in the model file, interprets it and creates the *HBG* data structure. The symbolic equation generator (SEG) takes the *HBG* and the current mode of the system and derives the state space equation (*SSE*) model of the system, which is stored in the *ASM*. When tracking system behavior, the hybrid observer reads in the data sample for the next time step from the data file, and checks to see if any controlled (specified in data file) or autonomous (stored in *ASM*) mode changes have occurred. When mode changes occur, the SEG is invoked to re-calculate the *SSE* model. To accommodate for model disturbances and measurement noise, a Kalman filter is built from the current SSE model to track system behavior. At each time step, the fault detector compares the system output with the observer estimates to determine if a fault has occurred in the system. When the fault detector triggers, the diagnosis module is activated. The diagnosis module uses propagation algorithms on the TCG to generate fault candidates that are consistent with the observed discrepancies. Continued tracking by matching the fault signatures generated for each candidate hypotheses helps refine the candidate set. For details on the hybrid observer and diagnosis algorithms, refer to [Mosterman and Biswas, 1999; Narasimhan and Biswas, 2002].

In subsequent sections we briefly describe the hybrid observer and the diagnosis modules and illustrate their functioning by applying them to a diagnosis problem on the fuel system. In the experimental setup, the fuel system is controlled by the sequence defined in Table 1. The data for the experiments was generated using a Matlab/Simulink simulation that was developed at Vanderbilt University. We assume pressure values are measured at the output of each of the six tanks of the fuel system. The fault introduced is an abrupt decrease in the left fuselage tank pump efficiency at time step 200. This occurs in the mode when only the left fuselage tank is supplying fuel, and only the left feed tank is receiving fuel.

4.1 Hybrid Observer and Fault Detector

The hybrid observer tracks the system behavior as it evolves and the fault detector compares the observer output to the system output to determine if a fault is present in the system. The hybrid observer performs the following tasks:

- Continuous tracking of system behavior in current mode,
- Determining if a mode change has occurred, and
- Initializing the observer in a new mode, with the new state and new model.

The discrete time form of the SSE models are derived to track system behavior. To account for model disturbances and noisy measurements, we use a Kalman filter to track system behavior. This requires computation of the R and Q matrices that model the disturbance and noise variances, and K, which

represents the Kalman gain matrix.

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - \hat{y})$$
$$\hat{y} = C\hat{x}$$
$$\dot{P} = AP + PA^T + BQB^T - KRK^T$$
$$K = PC^T R^{-1}$$

In our experiments, R and Q are diagonal matrices with values of 0.01 along the diagonal. The Kalman gain (K) is initialized to a diagonal matrix of arbitrarily high value (100 in our experiments). This gain matrix typically converges to its true value in a few time steps.

Mode changes may be of two types: controlled or autonomous. Controller issued switching commands need to be provided in the data file. These correspond to the controlled mode changes in the system. At each time step, the observer checks to see if the data set indicates a mode change. All autonomous change conditions are converted so that they contain only state and input variables. The observer uses input data and estimated state values to calculate if the conditions for any autonomous change evaluates to true. This is done at each time step also. For the fuel system, there are no autonomous changes and hence the data file provides sufficient information to determine if a mode change has occurred. If a controlled or autonomous mode change is indicated, the observer computes the new mode. The equation solver is invoked to derive the new *SSE* model. The observer re-initializes the state based on the reset function specified, and continues the tracking in the new mode with a new Kalman filter that is derived from the A and B matrices in the new mode.

Fig. 6 illustrates a sample run of the hybrid observer for the experimental setup described earlier. Gaussian noise with a 2% noise variance was generated using the Matlab models as described earlier. We illustrate the tracking of pressure in the left fuselage tank. The thick line represents the noisy system output (it is more like a waveform than a line due to the noise in the measurements) and the thin line represents the observer estimates. Until time step 200 (at which point the fault was introduced) we notice that this line is completely subsumed by the thick line indicating accurate tracking. However, after time step 200 the thin line deviates from the thick line indicating a fault. The fault detector (uses a 5% detection threshold) flags the fault. In the next section, we describe the fault isolation scheme.

4.2 Fault Isolation and Identification

Our fault isolation and identification methodology, described in greater detail in [Narasimhan and Biswas, 2002], for hybrid systems is broken down into three steps:

1. A fast *roll back process* using qualitative reasoning techniques to generate possible fault hypotheses. Since the fault could have occurred in a mode earlier than the current mode, fault hypotheses need to be characterized as a two-tuple *(mode, fault parameter)*, where mode indicates the mode in which the fault occurs, and fault parameter is the parameter of an implicated component whose deviation possibly explains the observed discrepancies in behavior.

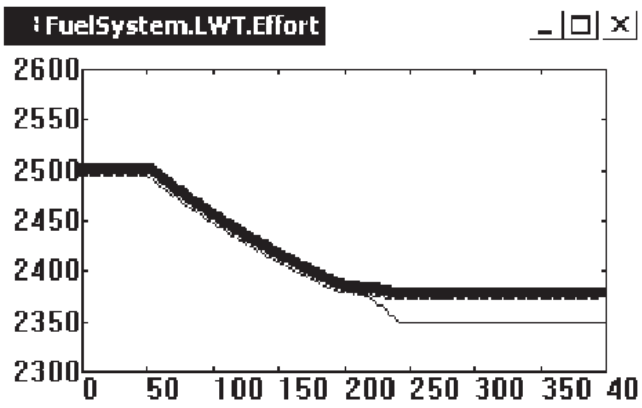


Figure 6: Hybrid Observer Sample Run

2. A quick *roll forward process* using progressive monitoring techniques to refine the possible fault candidates. The goal is to retain only those candidates whose fault signatures are consistent with the current sequence of measurements. After the occurrence of a fault, the observer's predictions of autonomous mode transitions may no longer be correct, therefore, determining the consistency of fault hypotheses also requires the fault isolation unit to roll forward to the correct current mode of system operation.
3. A *real-time parameter estimation process* using quantitative parameter estimation schemes. The qualitative reasoning schemes are inherently imprecise. As a result a number of fault hypotheses may still be active after Step 2. We employ least squares based estimation techniques on the input-output form of the system model to estimate consistent values of the fault parameter that is consistent with the sequence of measurements made on the system.

The models used in these three steps, temporal causal graph (*TCG*) and input output equations (*IOE*) model, are derived directly from the hybrid bond graph.

We illustrate the diagnosis algorithms for the experiment discussed in the previous section. As Fig. 5 illustrates, after time step 200 the actual pressure in the left fuselage tank is below the predicted value. The fault detector flags this and triggers the diagnosis process. We use the roll back procedure to propagate this discrepancy back through our models to generate the fault candidates. In the current mode, we get the following candidates: Left Fuselage Pump-, Left Fuselage Pipe+, Transfer Manifold+, Bleed Resistor+, Left Switched Pipe+, Left Feed Pump-. Since the left fuselage tank was not open in any of the previous modes, no candidates are generated in any previous modes.

The next step rolls forward to check for the consistency of the effects of the faults hypothesized against actual system measurements. Since no autonomous mode changes are present and we assume that all controller commands are known, we know exactly what mode the system is in. We generate signatures (effects of fault) in that mode for all the above candidates. In the current mode we cannot distinguish

between the candidates because they have similar signatures. However, when a mode change occurs (right feed tank is also opened), we regenerate signatures in the new mode and note that Left Switched Pipe+ and Left Feed Pump- do not affect the right feed tank pressure. However, we notice a discrepancy in the right feed tank pressure, hence we can drop these candidates. We cannot distinguish between the other candidates (Left Fuselage Pump-, Left Fuselage Pipe+, Transfer Manifold+) with the selected set of measurements. In order to distinguish between these candidates we need more measurements. For example, we could model the pump in more detail and add a sensor to measure the current drawn by the pump motor. This would let us identify faults in the pump as opposed to faults in pipes that the pump is connected to.

Table 2 lists the different fault classes in the fuel system. Each fault class represents multiple instances of the faults in the same component. The fault classes are numbered as follows: 1) Wing Tank Pump (WTP), 2) Wing Tank Pipe Resistance (WTR), 3) Fuselage/Transfer Tank Pump (TTP), 4) Fuselage/Transfer Tank Pipe Resistance (TTR), 5) Transfer Manifold Resistance (TMR), 6) Switched Pipe Resistance (SPR, 7) Feed Tank Pump (FTP), and 8) Feed Tank Pipe Resistance (FTR). The results of our diagnosis experiments for these sets of faults appear in the table. The $\surd$ mark in row i and column j indicates that if the roll back process generated candidates i and j, one of them will be dropped by the roll forward process. The $\times$ mark indicates that the current control sequence and set of measurements are not sufficient to distinguish between the pair in question. In general, the algorithm cannot distinguish between pump and pipe faults associated with the same tank. We need more detailed models and more measurements to do this. We also see that we cannot distinguish between the transfer manifold and fuselage pipe faults. We can distinguish between all other fault classes. The ability to distinguish between all fault classes is critical since the change in control strategy depends on the fault type.

Table 2: Fuel System Fault Diagnosability

	WTP	WTR	TTP	TTR	TMR	SPR	FTP	FTR
WTP	-	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
WTR	$\times$	-	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
TTP	$\surd$	$\surd$	-	$\times$	$\times$	$\surd$	$\surd$	$\surd$
TTR	$\surd$	$\surd$	$\times$	-	$\times$	$\surd$	$\surd$	$\surd$
TMR	$\surd$	$\surd$	$\times$	$\times$	-	$\surd$	$\surd$	$\surd$
SPR	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	-	$\surd$	$\surd$
FTP	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	-	$\times$
FTR	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	-

5 Conclusions

We have presented a case study on modeling a real system and building a diagnosis engine for the system. The key to successful tracking and diagnosis is to have models that are topologically correct, with parameter value estimates that match the nominal observed behavior so as not to generate false alarms. This can be a difficult and time consuming task, with a lot of experimental data being required to build accurate models. The presence of noise in the data complicates the tracking and fault detection task. For the given set of measurements, our tracking mechanisms worked well with fault-free data provided the variance of the added Gaussian noise was limited to 2%. Part of the reason for such low noise thresholds was the use of a naive threshold-based fault detector in these experiments. The diagnosis system always generated the true fault hypothesis, but in a number of cases the hypothesis set contained more than one fault candidate. This could be attributed to lack of detail in the models and the need for more measurements in the analysis. Also, parameter estimation was not included as part of the experiment. In previous work [Narasimhan and Biswas, 2002], we have shown that parameter estimation often helps to isolate the true fault.

In future work, we would like to build more detailed models of the different components of the fuel system in an attempt to diagnose a larger set of faults. The experiments need to be extended to run with real data provided from Boeing, as opposed to simulated Matlab data that we generated at Vanderbilt University. We would also like to run sensitivity analysis tests to the diagnosis system performance under varying noise and disturbance conditions. Finally we would like to build more robust techniques for fault detection and parameter estimation to combat the effects of noise and disturbance.

6 Acknowledgments

This project was conducted as part of ISIS and the EHS lab at Vanderbilt University. The DARPA/ITO SEC program (F30602-96-2-0227), NASA-IS grant (NCC2-1238), and the Boeing Company, St. Louis have supported the activities described in this paper. We would also like to thank Brian Olson and John Ramirez for helping to build the initial fuel system models.

References

[Biswas and Yu, 1993] Gautam Biswas and Xudong W. Yu. A formal modeling scheme for continuous systems: Focus on diagnosis. In *IJCAI 93*, pages 1474–1479, Chamberey, France, 1993.

[Buck *et al.*, 1994] J.T. Buck, S. Ha, E.A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogenous systems. *International Journal of Computer Simulation, Special Issue on "Simulation Software Development"*, 4:155–182, April 1994.

[deKleer and Williams, 1987] Johan deKleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Gertler, 1997] Janos Gertler. Fault detection and isolation using parity relations. *Control Engineering Practice*, 5(5):653–661, 1997.

[Harel, 1987] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hofbaur and Williams, 2002] Michael W. Hofbaur and Brian Williams. Mode estimation of probabilistic hybrid

systems. In *Fifth International Workshop on Hybrid Systems: Computation and Control*, page To Appear, Stanford, California, USA, March 2002.

[Karnopp *et al.*, 1990] D.C. Karnopp, D.L. Margolis, and R.C. Rosenberg. *Systems Dynamics: A Unified Approach*. John Wiley and Sons, New York, 2 edition, 1990.

[Ledeczi *et al.*, 2001] A. Ledeczi, M.Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstorm, J.Sprinkle, and P.Volgyesi. The generic modeling environment. *Workshop on Intelligent Signal Processing*, 2001.

[Lunze, 1999] J. Lunze. A timed discrete-event abstraction of continuous-variable systems. *International Journal of Control*, 72(13):1147–64, 1999.

[McIlraith *et al.*, 2000] Sheila McIlraith, Gautam Biswas, Dan Clancy, and Vineet Gupta. Hybrid systems diagnsois. In *Third International Workshop on Hybrid Systems*, pages 282–295, 2000.

[Mosterman and Biswas, 1998] Pieter J. Mosterman and Gautam Biswas. A theory of discontinuities in physical system models. *Journal of the Franklin Institute : Engineering and Applied Mathematics*, 1(3):401–439, 1998.

[Mosterman and Biswas, 1999] Pieter J. Mosterman and Gautam Biswas. Diagnosis of continuous valued systems in transient operating regions. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(6):554–565, 1999.

[Narasimhan and Biswas, 2001] Sriram Narasimhan and Gautam Biswas. Efficient diagnosis of hybrid systems using model of supervisory controller. In *12th International Workshop on Principles of Diagnosis (DX '01)*, pages 127–134, via Lattea, Italy, March 2001.

[Narasimhan and Biswas, 2002] Sriram Narasimhan and Gautam Biswas. An approach to model-based diagnosis of hybrid systems. In *Fifth International Workshop on Hybrid Systems: Computation and Control*, Stanford , CA, USA, vol. LNCS 2289, pp. 308-322, March 2002.

[Narasimhan *et al.*, 2000] Sriram Narasimhan, Gautam Biswas, Gabor Karsai, Tal Pasternak, and Feng Zhao. Building observers to address fault isolation and control problems in hybrid dynamic systems. In *The 2000 IEEE International Conference on Systems, Man, and Cybernetics (SMC '00')*, pages 2393–2398, Nashville, TN, USA, October 2000.

[Sampath *et al.*, 1996] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[van Amerongen, 2000] Job van Amerongen. Modeling, simulation and controller design of mechatronic systems with 20sim. *IFAC Mechatronics*, 2000.

A Model-Based Diagnosis Framework for Distributed Systems*

Gregory Provan
Rockwell Scientific Company,
1049 Camino Dos Rios, Thousand Oaks, CA 91360
gprovan@rwsc.com

Abstract

We present a distributed model-based diagnostics architecture for embedded diagnostics. We extend the traditional model-based definition of diagnosis to a distributed diagnosis definition, in which we have a collection of distributed components whose interconnectivity is described by a directed graph. Assuming that each component can compute a local minimal diagnosis based only on sensors internal to that component and knowledge only of its own system description, we describe an algorithm that guarantees a globally sound, complete and minimal diagnosis for the complete system. By compiling diagnoses for groups of components based on the interconnectivey graph, the algorithm efficiently synthesizes the local diagnoses computed in distributed components into a globally-sound system diagnosis using a graph-based message-passing approach.

1 INTRODUCTION

This article proposes a new technique for diagnosing distributed systems using a model-based approach. We assume that we have a system consisting of a set of inter-connected components, each of which computes a local (component) diagnosis.[1] We adopt the structure-based diagnosis framework of Darwiche [8] for synthesizing component diagnoses into globally-sound diagnoses, where we obtain the structure from the component connectivity. Unlike previous approaches that compute diagnoses using the system observations and a system description [8; 10], we transform the component diagnosis synthesis into the space of minimal diagnoses. Assuming that each component can compute a local minimal diagnosis based only on sensors internal to that component and knowledge only of the component system description, we describe an algorithm that guarantees a globally sound, complete and minimal diagnosis for the complete system. This algorithm uses as input the directed graph (digraph) describing the connectivity of distributed components,with arc directionality derived from the causal relation between the the components. Given that real-world graphs of this type are either tree-structured or can be converted to tree-structured graphs, we propose a graph-based message-passing algorithm which passes diagnoses as messages and synthesizes local diagnoses into a globally minimal diagnosis in a two-phase process. By compiling diagnoses for collections of components (as determined by the graph's topology), we can significantly improve the performance of distributed embedded systems. We show how this approach can be used for the distributed diagnosis of systems with arbitrary topologies by transforming such topologies into trees.

One important point to stress is that this approach synthesizes diagnoses computed locally, and places no restriction on the technique used to compute each local diagnosis (e.g., neural network, Bayesian network, etc.), provided that each local diagnosis is a least-cost or most-likely diagnosis. The synthesis approach takes this set of self-diagnosing sub-systems, together with the connectivity of these sub-systems, to compute globally-consistent diagnoses.

The approach presented in this article assumes that all faults are diagnosable (i.e., can be isolated) through a centralized algorithm. We examine whether a distributed approach can diagnose all faults, since a distributed algorithm can isolate faults no better than a centralized algorithm. Issues relating to restricted diagnosability of both centralized and distributed algorithms due to insufficient observable data (e.g., when the suite of sensors is insufficient to guarantee complete diagnosability) are examined in [21].

This article is organized as follows. Section 2 introduces the application model that we use to demonstrate our approach. Section 3 introduces our modeling formalism, and specifies our notion of centralized and distributed model. Section 4 describes how we diagnose distributed models. Section 5 surveys some related work on this topic. We summarize our conclusions in Section 6.

2 IN-FLIGHT ENTERTAINMENT EXAMPLE

Throughout this article we use a simplified example of an

*Research supported in part by The Office of Naval Research under contract number N00014-98-3-0012.

[1]Note that one can compute component diagnoses using any method which returns a minimal diagnosis (with respect to a specified minimality criterion).

In-Flight Entertainment (IFE) system. Figure 1 shows the schematic for an IFE system fragment where we have (1) a transmitter module (Tx) that generates 10 movie channels (consisting of both video and audio signals) and 10 audio channels; (2) two area distribution boxes (ADB); and (3) attached to each ADB_i we have two passenger units, P_{i1} and P_{i2}. For ADB j, passenger $i,\ i = 1,\ 2$ has a controller C_{ji} for selecting a video or audio channel, plus an audio unit α_i and video display υ_i. Control signal C_{ji} is sent by passenger i to ADB_j and then to the transmitter, which in turn sends an RF signal (RF) to each passenger.

We adopt a notion of causal influence for describing how different components affect the value of a signal as it propagates through the system. For example, the RF signal causally influences the passenger audio and video outputs. In this model the observables are the control signals, plus for passenger i downstream of ADB_j sound (S_{ji}) and video-display (VD_{ji}). We assign a fault-mode to the transmitter and to each ADB and passenger unit.

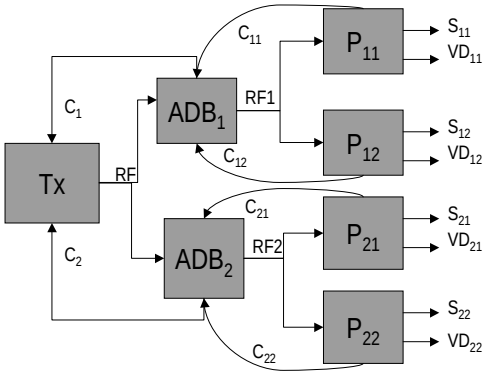


Figure 1: Schematic of IFE fragment, showing the main modules and the directed arcs of data-flows.

Our modeling approach makes the following assumptions. First, we can specify a system using an object-oriented approach. In other words, a system can be defined as a collection of components, which are connected together, e.g., physically, as in an HVAC system, or in terms of data transmission/reception, as in the IFE example. Our primary component consists of a *block*, which has properties: input set, output set, fault-mode, and equations. Given the fault-mode and input set, the equations provide a mapping to the output set. In other words, the inputs are the only nodes with causal arcs into the block, and the outputs are the only nodes with causal arcs out of the block. Typically, we have causal dependence of block outputs ω_i on inputs ℓ_i, *i.e.* $\omega_i \propto \ell_i$.[2]

This distributed model consists of a set of sub-models, or blocks, which may be connected together. In our IFE example, the transmitter block has inputs of control signals C_1 and C_2, and output an RF signal.

Second, we assume that each component computes diagnoses based on data local to the component. We do not place any restrictions on the type of algorithm used to compute the diagnosis, except that the diagnosis be a least-cost diagnosis. We will describe the cost function used by our synthesis algorithm in the following section.

[2]The causal function $\propto$ can be be generalized to include propositions, relations, probabilistic functions, qualitative differential equations, etc. We don't address such a generalization here.

3 MODEL-BASED DIAGNOSTICS USING CAUSAL NETWORKS

This section formalizes our modeling and inference approach to diagnostics and control reconfiguration. We first introduce the model-based formalism, and then extend these notions to capture a distributed model-based formalism.

3.1 FLAT (CENTRALIZED) MODELS

We adopt and extend the model-based representation for diagnosis of Darwiche [8]. We model the system using a causal network:

Definition 1 *A system description is a four-tuple* $\Phi = (\mathcal{V}, \mathcal{G}, \Sigma)$*, where*

- $\mathcal{V}$ *is a set of variables comprising two variable types:* $\mathcal{A}$ *is a set of variables (called assumables) representing the failure modes of the components,* $\mathbf{V}$ *is a set of non-assumable variables (*$\mathbf{V} \cap \mathcal{A} = \emptyset$*) representing system properties other than failure modes;*
- $\mathcal{G}$ *is a directed acyclic graph (DAG) called a causal structure whose nodes are members in* $\mathbf{V} \cup \mathcal{A}$ *and whose directed arcs represent causal relations between pairs of nodes;*
- *and* Σ *is a set of propositional sentences (called the domain axioms) constructed from members in* $\mathbf{V} \cup \mathcal{A}$ *based on the topological structure of* $\mathcal{G}$*.*

This definition of system description differs from the standard definition (called SD in [22]) only in that we include a graph $\mathcal{G}$ to complement the domain axioms set of failure modes (commonly called COMPS) and non-assumable variables.

The set of non-assumable variables consists of two exclusive subsets: $\mathbf{V}_{obs}$ (the set of observables) and $\mathbf{V}_{unobs}$ (the set of unobservables).

We can capture structural properties of the system description using the directed acyclic graph, or DAG, $\mathcal{G}$.[3] For example, if an actuator determines if a motor is on or not, we say that the actuator causally influences the motor. More generally, A may directly causally influence B if A is a predecessor of B in $\mathcal{G}$. We use $B \propto A$ to denote the direct causal influence of the value of B by the value of A.[4] Through transitivity, we can deduce *indirect* causal influence. For example, if $B \propto A$ and $C \propto B$, then A indirectly influences C.

This captures the notion of direct causal influence, i.e., a node N and those nodes that are directly causally affected by N, using a *clan*. We define the notion of the clan of a node N of a DAG $\mathcal{G}$ in terms of graphical relationships as follows:

[3]In other system description specifications, e.g. [12], these structural relations are captured using logical sentences.

[4]This notion of causal influence does not guarantee that A influences B, but that A *may* influence B.

Definition 2 (Clan) *: Given a DAG $\mathcal{G}$, the clan $Y(N_i)$ of a node $N_i \in \mathcal{G}$ consists of the node N_i together with its children in $\mathcal{G}$.*

We adopt the notion of clan because we are interested in synthesizing diagnoses computed at a set of distributed nodes organized in a tree structure. The intuition behind the algorithm is as follows: given local diagnoses, we start at the parents of leaves in the decomposition tree and move up the tree to the root, identifying if any node's diagnosis is affected by the diagnoses of its children, and if so, synthesizing those diagnoses. To perform each synthesis operation, we use a clan.

A clan is dual to the well-known notion of *family*, which is typically defined as a node together with its parents in $\mathcal{G}$. This notion is important because we need to synthesize local diagnostics within tree-structured systems, and the clan provides a more efficient means for doing so than the family for tree-structured systems. For simplicity of notation, we will denote the clan for node N_i, $Y(N_i)$, as Y_i.

It is also important to define restrictions of subsets of observables:

Definition 3 (Restriction) *We denote by θ_i the restriction of an instantiation θ of variables V to the instantiation of a subset V_i of V. We denote the restriction of variable set T to variables in sub-system description Φ_i by T^{Φ_i}.*

One of the key elements of diagnosing a system is the instantiation of observables, since a diagnosis is computed for abnormal observable instantiations.

Definition 4 (Instantiation) *θ^{Φ_i} is an instantiation of observables ${V_{obs}}^{\Phi_i}$ for system description Φ_i. Θ^{Φ_i} denotes the set of all instantiations of observables ${V_{obs}}^{\Phi_i}$.*

We specify failure-mode instantiations and partition the possible states into normal states and faulty states as follows:

Definition 5 (Mode-Instantiation) *$\mathcal{A}^*$ is an instantiation of behavior modes for mode-set $\mathcal{A}$. Further, we decomposition $\mathcal{A}^*$ such that $\mathcal{A}^* = \mathcal{A}^F \cup \mathcal{A}^\emptyset$, where $\mathcal{A}^\emptyset$ denotes normal system behaviour, i.e. all modes are normal, and $\mathcal{A}^F$ denotes a system fault, which may consist of simultaneous faults in multiple components.*

An assumable (behavior-mode variable) specifies the discrete set of behavior-states that a component can have, e.g., and AND-gate can be either *OK*, *stuck-at-0*, or *stuck-at-1*. Our IFE-system, with component-set $\{Tx, ABD_1, ADB_2, P_{11}, P_{12}, P_{21}, P_{22}\}$, can have a mode-instantiation in which all components are OK except P_{11}, which is in *audio-fail* mode. In this case we have $\mathcal{A}^\emptyset = \{Tx - mode = OK, ABD_1 - mode = OK, ADB_2 - mode = OK, P_{12} - mode = OK, P_{21} - mode = OK, P_{22} - mode = OK\}$ and $\mathcal{A}^F = \{P_{11} - mode =$*audio-fail*$\}$.

3.2 DISTRIBUTED SYSTEM DESCRIPTIONS

This section describes our distributed formalism, which applies to collections of interconnected components, or blocks. We assume that a distributed system description is provided either by the user or is deduced from the physical constraints of available local diagnostic agents and physical connectivity. For example, many engineering systems, such as commercial aircraft, are subdivided into Line-Replaceable Units (LRUs), based on a number of factors, such as fault-isolation capabilities, physical constraints, and ease of repair. An LRU typically consists of a number of connected sub-systems, as in the Passenger Unit of the IFE example, which consists of circuit-cards to select audio/video channels and to drive the audio and video output devices. It is standard practice in commercial aircraft to isolate faults only to the LRU-level, and replace faulty components only at the LRU-level.

Definition 6 (Decomposition Function) *a decomposition function is a mapping $\psi(\Phi) = \Phi_{dist}$ that decomposes a centralized system description Φ into a distributed system description $\Phi_{dist} = \{\Phi_1, ..., \Phi_m\}$. The distributed system description induced by a decomposition function ψ is defined by a decomposition Π over the system variables $\mathcal{V}$, i.e. a collection $\mathcal{X} = \{X_1, ..., X_m\}$ of nonempty subsets of $\mathcal{V}$ such that (1) $\forall i = 1, ..., m,\ X_i \in 2^{\mathcal{V}}$; (2) $\mathcal{V} = \cup_i (X_i | X_i \in \Pi)$. When $\xi_{ij} = X_i \cap X_j \neq \emptyset$, we call ξ_{ij} the separating set, or sepset, of variables between Φ_i and Φ_j.*

We can describe a distributed system description in terms of a *decomposition graph*. A decomposition graph is a graphical representation of the system model, when viewed as a collection of connected blocks. In this graph each vertex corresponds to a block, and each directed edge corresponds to a directed (causal) link between two blocks. Figure 2 shows the decomposition graph for the extended IFE example. [5]

A *decomposition graph* is a directed tree, or D-tree, which is defined as follows:

Definition 7 *A D-tree $\mathcal{T}_\mathcal{D}$ is a directed graph with vertices $V(\mathcal{T}_\mathcal{D})$ with a vertex r_0, called the root, with the property that for every vertex $r \in V(\mathcal{T}_\mathcal{D})$ there is a unique directed walk from r_0 to r.*

Definition 8 *A decomposition graph $G_\mathcal{X}$ is an edge-labeled D-tree $G(\mathcal{X}, \mathcal{E}, \xi)$ with (1) vertices $\mathcal{X} = \{X_1, ..., X_m\}$, where each vertex consists of a collection of variables of $\mathcal{G}$, (2) directed edges join pairs of vertices with non-empty intersections, and arc direction is specified by the causal direction of the arcs between blocks in the decomposition graph, i.e., $\mathcal{E} = \{(X_j, X_k) | X_i \cap X_j \neq \emptyset,\ X_k \propto X_j\}$, and (3) edge labels (or separators) defined by the edge intersections, $\xi = \{\xi_{ij} | X_i \cap X_j \neq \emptyset\}$.*

We assume that in a distributed system description, for any block all sensor data is local, and all equations describing distributed subsystems refer to local sensor data and local conditions.

3.3 DIAGNOSIS SPECIFICATION

We define the notion of diagnosis as follows:

Definition 9 (Diagnosis) *Given a system description Φ with domain axioms Σ and an instantiation θ of $\mathbf{V}_{obs}$, a diagnosis $D(\theta)$ is an instantiation of behavior modes $\mathcal{A}^F \cup \mathcal{A}^\emptyset$ such that $\Sigma \cup \theta \cup \mathcal{A}^F \cup \mathcal{A}^\emptyset \not\models \bot$.*

[5] We do not show the feedback loops of control requests $(C_1, C_2, C_{11} ..., C_{22})$ since all edges concerning observables can be cut [7].

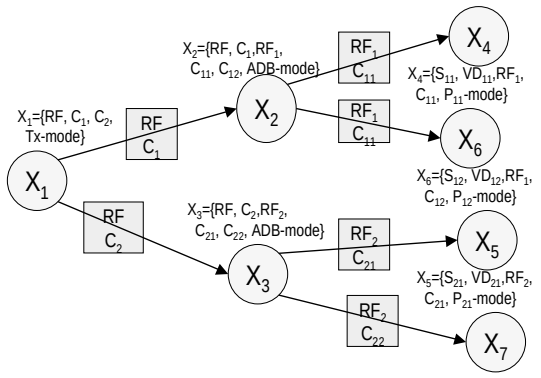


Figure 2: Decomposition graph of extended IFE system description. Here an oval corresponds to a vertex, and a block corresponds to a sepset. We specify the variables associated with each vertex in the graph.

This diagnostic framework provides the capability to rank diagnoses using a likelihood weight κ_i assigned to each assumable $\mathcal{A}_i,\ i = 1, ..., m$. Using the likelihood algebra defined in [8], we can compute the likelihood assigned to each diagnosis for observation θ. We refer to a (diagnosis, weight) pair using $(D(\theta), \kappa)$. We use the weights to rank diagnoses, i.e., least-weight diagnoses are the most-likely. This provides a notion of *minimal diagnosis*, i.e. a diagnosis of weight κ such that there exists no lesser-weight diagnosis.

3.4 LOCAL/GLOBAL DIAGNOSTICS

Our methodology rests on the determination of when component diagnoses are independent, in which case the global diagnosis is just the conjunction of the component diagnoses. We apply the decomposition theorem of [8] to this case of distributed diagnostics:

Theorem 1 *If we have a system description Φ consisting of two component system descriptions Φ_1 and Φ_2, and a system observation θ, if the variables shared by Φ_1 and Φ_2 all appear in θ, then*

$$D^{\Phi}(\theta) \equiv D^{\Phi_1}(\theta_1) \wedge D^{\Phi_2}(\theta_2).$$

This theorem states that a diagnosis is decomposable provided that the system observation contains the variables shared between Φ_1 and Φ_2. However, what happens when the observation θ does not contain all variables shared between Φ_1 and Φ_2? One solution [8] is to decompose the computation of D^{Φ} by performing a case-analysis of all shared variables ξ_{12}. However, this case-analysis approach is exponential in $|\xi_{12}|$, the number of variables on which we do case-analysis. Hence if we wanted to embed the diagnostics code, such a case-analysis might be too time-consuming when performed on a system-level model.

In the following we assume that each component computes a *local diagnosis*, i.e., a diagnosis based only on local observables and on equations containing only local variables. In contrast a *global diagnosis* is one based on global observables and on equations describing all system variables. Our task is to integrate these local component diagnoses into a globally sound, minimal and consistent diagnosis, since for many systems the diagnostics generated locally are either incomplete or not minimal.

Note that we can obtain global diagnostics for a modular system by composing local blocks and diagnosing the entire system model. However, it is true in many cases that global and local diagnostics may differ. We now define a notion of correspondence between local and global diagnoses.

The conjunction of the set of distributed system descriptions is defined as $D_{dist}(\theta) = \bigwedge_{\Phi_k \in B} D^{\Phi_k}(\theta)$, and we know that $D_{dist}(\theta) = D(\theta)$ only when $\theta \equiv \bigcup i, j \xi_{ij}$.

We can compute the diagnoses for this set of distributed system descriptions either using an on-line algorithm, or by pre-computing the set of diagnoses for $D_{dist}(\theta)$. In the following, we outline the compiled method of diagnosis.

We define a table, called a clan table, to specify local and global diagnoses for collections of blocks. This table compiles the local case-analysis required by Theorem 1. We will show later how to use this table for our diagnosis synthesis algorithm.

Definition 10 *A* **clan (or local/global diagnosis) table** *for block-set $B = \{\Phi_i, ...\Phi_j\}$ is a table consisting of tuples (observable-intantiation, global diagnosis, weight) for all abnormal instantiations of observables θ in B.*

Note that we can use the compositionality of blocks to show that any time we compose a system description from multiple blocks, we obtain "global" diagnostics for that composed system description when we compute diagnoses over the composed system description. Hence the "global" diagnosis for each collection of blocks is computed from a system description generated from the composition of the system descriptions of the blocks in B, using the observables from B.

Example 1 Table 1 contrasts the local and global diagnoses for a set of scenarios where the set B of blocks is an ADB with downstream passenger units. In these scenarios, we compute the (probabilistically) most-likely diagnosis, assuming that all faults are equally likely, i.e., have weight 1. Moreover, in defining a local diagnosis in Table 1, we report the conjunction of all local diagnoses, i.e. the local diagnosis is *ADB-diagnosis* $\wedge$ P_1*-diagnosis* $\wedge$ P_1*-diagnosis*. In scenarios 1, 2 and 4, the local and global diagnoses are identical. However, in scenarios 3, 5 and 6, they differ: the passenger units each assume a local fault, whereas the transmitter unit is the faulty one (since a single transmitter fault is much more likely the two simultaneous faults, one in each passenger unit).[6]

Given this potential for discrepancy between local and global diagnoses, we map the decomposition graph into a representation, the clan graph, from which we can synthesize globally sound and complete minimal diagnoses from local minimal diagnoses. Figure 3 shows the clan graph for the extended IFE example.

[6]These differences arise due to different instantiations of the RF signal in the local and global diagnosis. We hide the details of the case-analysis of shared variables for simplicity of presentation.

Scenario	ADB_1 Unit		Pass. $Unit_{11}$		Pass. $Unit_{12}$		Diagnosis	
	C_{11}	C_{12}	S_{11}	VD_{11}	S_{12}	VD_{12}	LOCAL	GLOBAL
1	audio	audio	nom.	none	nom.	none	–	–
2	audio	audio	**none**	none	nom.	none	P_{11}-*audio-fail*	P_{11}-*audio-fail*
3	audio	audio	**none**	none	**none**	none	P_{11}-*audio-fail*$\wedge$ P_{12}-*audio-fail*	*Xaudio*
4	video	video	nom.	nom.	nom.	**none**	P_{12}-*video-fail*	P_{12}-*video-fail*
5	video	video	nom.	**none**	nom.	**none**	P_{11}-*video-fail*$\wedge$ P_{12}-*video-fail*	*Xvideo*
6	audio	video	**none**	none	**none**.	**none**	P_{11}-*audio-fail*$\wedge$ P_{12}-*video-fail*	ADB_1-*fail*

Table 1: Diagnostic Scenarios. We denote a nominal passenger output of nominal using nom., and abnormal observable data in bold-face. *Xaudio* denotes degraded audio, and *Xvideo* denotes degrated video.

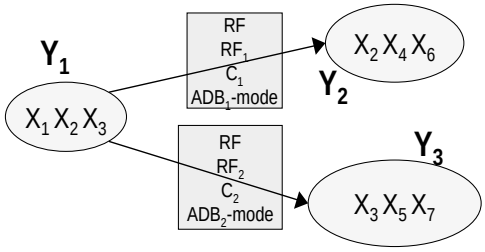


Figure 3: Clan graph of extended IFE system description.

Definition 11 (Clan graph) *: A clan graph $G_{\mathcal{Y}}$ of a DAG $\mathcal{G}(V, E)$ of vertices V and edges E is an edge-labeled D-tree $G(\mathcal{Y}, \mathcal{E}, \xi)$ defined as follows: (1) vertices $\mathcal{Y} = \{Y_1, ..., Y_m\}$, where each node Y_i consists of a clan of $\mathcal{G}$; (2) edges defined by non-empty intersections between pairs of vertices $\mathcal{E} = \{(Y_j, Y_k)|Y_i \cap Y_j \neq \emptyset\}$; and (3) separators defined by the edge intersections $\xi = \{\xi_{ij} = Y_i \cap Y_j\}$.*

The following section shows how we use the clan graph for distributed diagnosis.

4 DISTRIBUTED MODEL-BASED DIAGNOSIS

This section describes our distributed model-based diagnosis algorithm. We first map the directed graph of the system into a tree using tree-decomposition techniques, and then employ a message-passing algorithm on the tree.

4.1 TREE-DECOMPOSITION

The work on tree-decomposition stems from work on treewidth and graph minors [23]. A good review of the literature can be found in [5]. We define the basic notions below.

Definition 12 *A* tree decomposition *of an undirected graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$ with $T = (I, F)$ a tree, and $\mathcal{X} = \{X_i | i \in I\}$ is a family of subsets of V, one for each node of T, such that*

1. $\bigcup_{i \in I} X_i = V$*;*
2. *for all edges $\{v, w\} \in E$ there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$, and*
3. *for all $i, j, k \in I$ if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The last property is known as the running-intersection property within the BN community. The clique-tree algorithm computes a tree-decomposition in which each node of the tree is a clique, and undirected edges correspond to shared variables between cliques.

Given a tree-decomposition, inference complexity is based on the treewidth, defined as follows. The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. The treewidth bears close relations to the maximal vertex degree and maximal clique of a graph, so it provides a measure of the complexity of diagnostic inference, among other things. If a graph has a low treewidth then inference on the graph is guaranteed to be easy. The task of computing treewidth is NP-hard [2]. Many algorithms exist that, given a graph with n variables, will compute an optimal treewidth in time polynomial in n but exponential in the treewidth k; see, for example, [4].

Directed Tree-Decomposition

The difference between the standard literature on tree-decompositions and the task addressed here is that the standard literature focuses on undirected graphs, and we focus on directed graphs. We capture and exploit the directionality of causal relations during all phases of diagnostic inference. For example, if we have an abstract hierarchical specification of a system and compute diagnostics for each abstract hierarchical block, we still preserve the directionality of causality among the abstract blocks. We exploit this directionality using a diagnostic synthesis algorithm operating on a *directed* tree.

Definition 13 *A D-tree $\mathcal{T}_{\mathcal{D}}$ is a directed graph with vertices $\mathcal{V}_{\mathcal{T}_{\mathcal{D}}}$ and a vertex V_0, called the root, with the property that for every vertex $V \in \mathcal{V}_{\mathcal{T}_{\mathcal{D}}}$ there is a unique directed walk from V_0 to V.*

The tree-decomposition results have been generalized to directed graphs in [16], and we make use of some of those results here. The key change is that we need to preserve ordering of edges during the decomposition process. To capture such properties, we first need to define a notion of variable ordering, called Z-normality.

Definition 14 *Let $\mathcal{G}$ be a digraph and let $Z \subseteq \mathcal{V}$. A set S is Z-normal if and only if the vertex-sets of the strong components of $\mathcal{G} \setminus Z$ can be numbered $S_1, S_2, ..., S_d$ such that*

1. *if $1 \leq i \leq j \leq d$, then no edge of $\mathcal{G}$ has a head in S_i and tail in S_j, and*
2. *either $S = \emptyset$ or $S = S_i \cup S_{i+1} \cdots \cup S_j$ for some integers i, j with $1 \leq i \leq j \leq d$.*

Definition 15 *A* D-tree decomposition *of a digraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a pair* $(\mathcal{X}, \mathcal{T}_\mathcal{D})$ *with* $\mathcal{T}_\mathcal{D} = (\mathcal{I}, \mathcal{F})$ *a D-tree, and* $\mathcal{X} = \{X_i | i \in \mathcal{I}\}$ *is a family of subsets of* $\mathcal{V}$*, one for each node of* $\mathcal{T}_\mathcal{D}$*, and the edges are numbered* $\mathcal{J} = \{1, ..., l\}$ *with* $\mathcal{F} = \{F_j : j \in \mathcal{J}\}$*, such that*

1. $\bigcup_{i \in \mathcal{I}} X_i = \mathcal{V}$*;*
2. *for all edges* $\{v, w\} \in \mathcal{E}$ *there exists an* $i \in \mathcal{I}$ *with* $v \in X_i$ *and* $w \in X_i$*, and*
3. *for all* $i, j, k \in \mathcal{I}$ *if* j *is on the path from* i *to* k *in* $\mathcal{T}_\mathcal{D}$*, then* $X_i \cap X_k \subseteq X_j$*;*
4. *if* $j \in \mathcal{J}$*, then* $\bigcup_i \{X_i : i \in \mathcal{I}, i > j\}$ *is* X_j*-normal.*

The *width* of a tree decomposition is the least integer w such that for all $i \in \mathcal{I}$, $|X_i \cup \bigcup X_j| \leq w + 1$, where the union is taken over all edges $j \in \mathcal{J}$ incident with i. $\max_{i \in \mathcal{I}} |X_i| - 1$. The *treewidth* of a graph $\mathcal{G}$ is the least integer w such that $\mathcal{G}$ has a D-tree-decomposition of width w.

For the class of applications addressed in this article, the input graphs $\mathcal{G}$ for the system description are digraphs, and the decomposition graph and clan graph are both D-tree decompositions of $\mathcal{G}$. For more general digraph topologies, by applying an algorithm for generating D-tree decompositions, we can convert the digraphs into a decomposition graph, and apply the diagnostic synthesis approach. Many of the properties of undirected tree-decompositions hold for the directed case [16].

4.2 DIAGNOSIS OF SYSTEMS WITH TREE-STRUCTURED GRAPHS

We now describe an approach to diagnosing systems with tree-structured decomposition graphs.

We assume that:

- We are provided with the component system descriptions and their connectivity;
- There is a single root in the decomposition graph (which is a component with no parent-components), and each leaf is a component with no child-component;
- Nodes have indices starting at the root (X_1), increasing based on a breadth-first expansion from the root and ending at the leaves, labeled $X_{n-s}, ..., X_n$;
- Each component computes a local diagnosis based on local observables.

We base our approach on synthesizing diagnoses, starting from the leaf components and ending up at the root of the tree. We first decompose the decomposition graph into a clan graph. Based on the clan graph we construct a clan table for each node in the graph.

This algorithm is inspired by the Bayesian network clique-tree approach of [17], but replaces the clique-tree with an analogous clan-tree, and passes diagnoses as messages. Analogous to the clique-tree method's clique-table pre-computation, this approach requires pre-computing clan-tables, but for embedded systems this results in computationally simpler algorithms than those adopted in the past.

Under this scheme, we pre-compute clan tables for each clan in $\mathcal{G}_\mathcal{Y}$. Given an observation θ for blocks $X_i, ..., X_k$, where $X_i, ..., X_k$ are members of a clan $Y \in \mathcal{G}_\mathcal{Y}$, each block computes diagnostics locally. We then compute the most likely fault-mode assignment for Y through a process we call *diagnostics synthesis*, which entails table-lookup in the clan table of the minimal diagnosis given θ. The algorithm synthesizes final diagnoses, going from the leaves to the root. This guarantees a sound, complete and globally minimum system diagnosis.

In this approach we first need to pre-compute the clan table, and then use that table for diagnostic synthesis. We can pre-compute the clan table from a set of blocks $\{\Phi_1, ..., \Phi_k\}$ as follows:

1. Generate the decomposition graph $G_\mathcal{X}$ from $\{\Phi_1, ..., \Phi_k\}$, with indices increasing in a breadth-first manner from the root.
2. Generate the clan graph $G_\mathcal{Y}$ of $G_\mathcal{X}$.
3. Compute the clan table for each clan Y_i in $G_\mathcal{Y}$.

Given an observation θ, the diagnostic synthesis algorithm is as follows:

1. Given observation θ, each block B_i computes its local diagnosis $D^{\Phi_i}(\theta)$ and likelihood $\kappa(D^{\Phi_i})$.
2. Mark all nodes $X_i,\ i = 1, ..., n$ with flag=0;
3. Loop for $j = n$ to 1:
   (a) If flag=0 for X_j do:
   For each node X_i in the clan $Y(X_j)$, look up corresponding clan diagnosis $D^{\Phi_Y}(\theta)$ and weight $\kappa(D^{\Phi_Y}(\theta))$ in the clan-table;
   $$\text{If } \kappa(D^{\Phi_Y}(\theta)) < \sum_{k:\Phi_k \in Y} \kappa(D^{\Phi_k}),$$
   - revise fault-mode assignment to nodes in $Y(N_j)$, by (a) setting the minimum-weight diagnosis mode-variable; (b) if any local diagnosis D' is synthesized, update D'
   - reassign values to variables in Y based on D and θ
   - if reassignment is sound pass message with fault report $D^{\Phi_Y}(\theta)$.
   - Set flag for all $X_i \in Y(X_j)$ to 1;

Theorem 2 *Given a tree-structured decomposition graph* $\mathcal{G}_\mathcal{X}$ *and local component diagnoses, diagnostics synthesis will compute a sound and globally consistent set of fault mode assignments for components* $\mathcal{X} \in \mathcal{G}_\mathcal{X}$ *within* $O(|\mathcal{Y}|)$ *message-passing steps, where* $\mathcal{G}_\mathcal{Y}$ *is the clan graph generated from* $\mathcal{G}_\mathcal{X}$*.*

Example 2 Diagnosis Synthesis in a Clan: Consider Scenario 3 of Table 1. For this observation θ, the total set of possible clan diagnoses is: (P_{11}, *audio-fail*) $\wedge$ (P_{12}, *audio-fail*) $\vee$ (ADB_1, *Xaudio*). The weights of the diagnoses are 2 and 1, respectively.

In computing diagnoses on a purely local basis, the resulting diagnosis is (P_{11}, *audio-fail*) $\wedge$ (P_{12}, *audio-fail*), with weight 2. Note however there is a family diagnosis of weight 1, (ADB_1, *Xaudio*), which is selected since it is of lower weight than the distributed diagnosis. We now instantiate each local component with θ, and set diagnoses as follows: (P_{11}, $\emptyset$), (P_{12}, $\emptyset$), (ADB_1, *Xaudio*). There exists a consistent set of local variable instantantiations for this assignment, so no further message-passing is necessary.

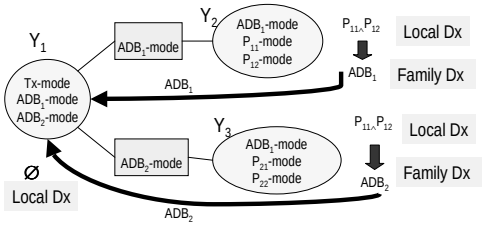


Figure 4: Diagnosis synthesis procedure, Step 1: (a) local diagnoses synthesized at clans, and (b) clan diagnoses are passed between families, as noted by dark arrows.

Example 3 Message-Passing: Figure 4 shows the first stage of this procedure. In the graph we show nodes where the variables are restricted to fault mode variables, to simplify the description of message-passing of instantations of mode variables. First, the local diagnoses are computed at each node in the decomposition graph: all four passenger units register a fault, and no other nodes in the decomposition graph register faults. As a shorthand, we denote a fault-weight pair using variable-names for faults, with $\emptyset$ denoting a nominal mode. Then, these faults are synthesized at each clan using the clan-table: fault-weight pair $(P_{11} \wedge P_{12},\ 2)$ is synthesized into $(ADB_1,\ 1)$, and fault $(P_{21} \wedge P_{22},\ 2)$ is synthesized into $(ADB_2,\ 1)$. Second, the synthesized faults $(ADB_1,\ 1)$ and $(ADB_2,\ 1)$ are sent to the adjacent node in the clan graph, Y_1.

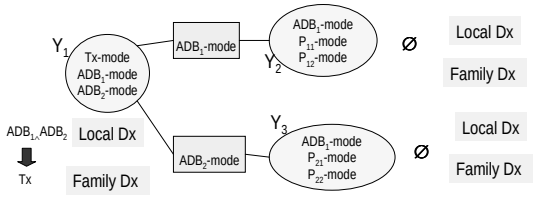


Figure 5: Diagnosis synthesis procedure, Step 2: global diagnoses computed following family diagnosis message-passing.

Figure 5 shows the second stage of this procedure. Fault-weight pair $(ADB_1 \wedge ADB_2,\ 2)$ is synthesized into $(Tx,\ 1)$ at clan Y_1, and all other fault-modes are set to nominal. This is the global minimum-weight fault.

4.3 COMPLEXITY ISSUES

The complexity of logical resolution within a distributed framework have been discussed in [1]. Here, our task is model-based diagnosis within a tree-structured topology.

This approach is based on computing diagnoses for the clans of $\mathcal{G}$. Hence, it never needs to diagnose a system description for the entire graph $\mathcal{G}$, but only for the clans of $\mathcal{G}$. As noted in Theorem 2, once the clan tables are computed, given any local component diagnoses, the algorithm is linear in the number of nodes in the clan-graph.

The worst-case complexity of computing a clan table is exponential in the number of variables in the clan table. The memory requirements for storing the clan tables are defined as follows. In the worst case, for a clan with mode variables $\mathcal{A}_1, ..., \mathcal{A}_m$, where each mode variable has $|\omega_{A_i}|$ faulty values, a clan table stores an entry for each of the $\times_i |\omega_{A_i}|$ multiple-fault combinations. For single-fault scenarios, a clan table must store only $\sum_i |\omega_{A_i}|$ entries.

The main issue is the time-complexity of generating the clan tables. For tree-structured systems the complexity of diagnosing $\mathcal{G}$ is exponential in the clan size, and the complexity is bounded by the largest clan of $\mathcal{G}$. Hence the complexity of initially computing diagnoses is the same for the centralized and distributed approaches. However, for embedded applications, the distributed approach has a complexity advantage, since only clan-table lookup and simple message-passing are required.

5 RELATED WORK

Our approach to distributed diagnosis has been preceded by many pieces of related work, and we review several here. Note that this review examines the most relevant work, and does not claim to be exhaustive.

One of the most closely-related pieces of work describes techniques for distributed logical inference [1; 20]. This work focuses on how to perform logical reasoning and query answering, proposing sound and complete message passing algorithms, by exploiting the tree structure of distributed theories. They examine the complexity of computation, propose specialized algorithms for first-order resolution and focused consequence finding, and propose algorithms for optimally partitioning a theory that is not already distributed. In some ways, our task can be considered a special case of the general problem that Amir and McIlraith examine. Logical inference computes a model, whereas diagnostic inference computes a *minimal* model in the assumables, a subset of the language of the theory. We leverage many aspects of the specific diagnosis problem in our work, aspects that serve to distinguish both our approach and our results. These include the notion of causality, which imposes a directionality on the tree structure and the inference, and the notion of preference. In addition, the task of diagnostic inference depends critically on two classes of distinguished variables, assumables (the literals of interest) and observables (the inputs), and distributed diagnosability depends on how assumables and observables are distributed among the collection of blocks. In addition, if the variables common between two blocks are observable, then from a distributed diagnostics point of view those blocks are independent [7].

The approach presented here bears some relation to diagnostic approaches on trees. Stumptner and Wotawa [25] have an algorithm for diagnosing tree-structured systems. This approach assumes a centralized system defined at the component level whereas our approach deals with distributed systems that can be defined at any level of abstraction. In addition, our assumption of sub-systems computing their own diagnoses means that our diagnostic synthesis process is a single-pass algorithm from the leaves of the tree to the root,

whereas Stumptner and Wotawa need a two-pass approach since they must first enumerate all component diagnoses. A second major tree-based method uses a clique-tree decomposition of a system, e.g., the diagnostic method of [13]. A clique-tree is a representation that is used for many kinds of inference in addition to diagnosis, including probabilistic inference and constraint satisfaction. The tree we generate is a directed tree with a fixed root, and the nodes of the tree are generated based on the clan property; a clique-tree is undirected (with an arbitrary root), and the nodes of the tree are generated based on the family property. One can think of the D-tree as a directed variant of a clique-tree, which is optimized for diagnostic inference. In addition, our approach uses the ordering of the D-tree to require message-passing in a single direction only; in contrast, message propagation in clique trees is bi-directional.

Our work also bears some relation to papers describing distributed solutions to Constraint Satisfaction Problems (CSPs) [26; 15]. As with the work on distributed logical inference [1], the task of distributed CSPs is finding a satisfying assignment to the variables, when constraints are distributed in a collection of subsets of constraints. Hence the underlying tasks of distributed diagnosis and CSP satisfiability are different. One issue in this work that is similar to diagnostic reasoning is the recording of minimal sets of unsatisfiable clauses as nogoods [15]. The computation of nogoods is a key step to computing diagnoses [10].

There have been several proposals for using the ATMS [9] in a distributed manner, e.g., [11; 19; 3; 18]. Our approach differs from this work in that our approach uses system topology explicitly, whereas these other approaches do not make as extensive a use of topology.

The compilation approach proposed in this article bears some relation to prior work.[7] [24] presents an empirical comparison of centralized compilation techniques as applied to several areas, of which diagnosis is one. Our future work includes examining the applicability of these compilation techniques within our distributed framework. Compilation is also examined in [20], but (as mentioned earlier) as applied to a different task, logical resolution.

There has been some prior work on distributed model-based diagnosis. For example, the approach in [14] assumes that the diagnosis computed by each distributed agent is globally correct, and examine the case where agents must cooperate to diagnose components whose status is unknown. Our approach makes the more realistic assumption that diagnoses are not necessarily globally sound, and derives a very different global synthesis algorithm.

6 SUMMARY AND CONCLUSIONS

This document has described a mechanism for computing distributed diagnoses using system topology and observability properties. This algorithm takes as input minimal diagnoses computed within distributed components, and uses system topology to integrate these diagnoses into a globally sound and minimal system diagnosis.

[7] A review of compilation can be found in [6].

We are in the process of applying this approach to two real-world domains, that of In-Flight Entertainment and diagnosis of HVAC systems.

The approach presented here provides a mechanism for designing systems with predictable distributed diagnostics properties. A given decomposition graph can be rated according to its diagnosability and efficiency. Additionally, given a system description, we can apply D-tree decomposition algorithms to the system DAG to assist in identifying small-treewidth decompositions, if any exist. Further, if a system has no small treewidth decomposition, one can then recommend system re-design to be facilitate efficiently computing distributed diagnoses.

References

[1] E. Amir and S. McIlraith. Paritition-based logical reasoning. In *Proc. KR '2000*, pages 389–400. Morgan Kaufmann, 2000.

[2] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Meth.*, 8:277–284, 1987.

[3] C. Beckstein, R. Fuhge, and G. Kraetzschmar. Supporting assumption-based reasoning in a distributed environment. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 3–17, Hidden Valley, Pennsylvania, 1993.

[4] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[5] H. Bodlander. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97*, volume 1295 of Lecture Notes in Computer Science, pages 29–36. Springer-Verlag, 1997.

[6] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

[7] A. Darwiche and G. Provan. Exploiting system structure in model-based diagnosis of discrete-event systems. In *Proc. 7th Intl. Workshop on Principles of Diagnosis*, pages 95–105, 1996.

[8] Adnan Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.

[9] J. de Kleer. An Assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[10] J. de Kleer and B. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97–130, 1987.

[11] A. Dragoni. Distributed belief revision versus distributed truth maintenance: preliminary report. In *Atti del 3zo Incontro del Gruppo AI*IA di Interesse Speciale su Inteligenza Artificiale Distribuita*, pages 64–73, Rome, Italy, 1993.

[12] O. Dressler and Peter Struss. The consistency-based approach to the automated diagnosis of devices. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 267–311. CSLI Publications, Stanford, CA, USA, 1996.

[13] Yousri El Fattah and Rina Dechter. Diagnosing tree-decomposable circuits. In *IJCAI*, pages 1742–1749, 1995.

[14] Peter Frohlich, Iara de Almeida Mora, Wolfgang Nejdl, and Michael Schroeder. Diagnostic agents for distributed systems. In *ModelAge Workshop*, pages 173–186, 1997.

[15] K. Hirayama and M. Yokoo. The effect of nogood learning in distributed constraint satisfaction. In *Proceedings of the 20th IEEE International Conf. on Distributed Computing Systems*, pages 169–177, 2000.

[16] T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Directed tree-width. *to appear in J. Combin. Theory Ser. B.*

[17] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Royal Statistical Society*, 50:154–227, 1988.

[18] Benedita Malheiro and Eugenio Oliveira. Solving conflicting beliefs with a distributed belief revision approach. In *IBERAMIA-SBIA*, pages 146–155, 2000.

[19] Cindy L. Mason and Rowland R. Johnson. DATMS: A framework for distributed assumption based reasoning. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence*, volume 2, pages 293–317. Pitman, 1989.

[20] Sheila A. McIlraith and Eyal Amir. Theorem proving with structured theories. In *Proc. IJCAI*, pages 624–634. Morgan Kaufmann, 2001.

[21] G. Provan. Distributed Diagnosability Properties of Discrete Event Systems. In *Proc. American Control Conference*, Anchorage, AK, May 2002.

[22] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–96, 1987.

[23] N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *J. Algorithms*, 7:309–322, 1986.

[24] Laurent Simon and Alvaro del Val. Efficient consequence finding. In *IJCAI*, pages 359–370, 2001.

[25] Markus Stumptner and Franz Wotawa. Diagnosing tree-structured systems. *Artificial Intelligence*, 127(1):1–29, 2001.

[26] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

Model-based Tools for the Integration of Design and Diagnosis into a Common Process - A Project Report*

P. Struss[1,8], B. Rehfus[2], R. Brignolo[3], F. Cascio[4], L. Console[5], P. Dague[6], P. Dubois[7], O. Dressler[8], and D. Millet[9]

[1]Technical Univ. of Munich, [2]DaimlerChrysler AG, Stuttgart, [3]Magneti-Marelli Spa, [4]Centro Ricerche Fiat; Torino, [5]Universita di Torino, [6]Université de Paris Nord, [7]Renault, Paris, [8]OCC'M Software, Munich, [9]PSA Peugeot Citroën, Paris
struss@in.tum.de, struss@occm.de

Abstract

The growing importance of on-board diagnosis for automobiles demands for a close integration of diagnostic tasks in the entire design process. This report describes work carried out to date within the European project „Integrated Design Process for onboard Diagnosis„ (IDD). It presents an analysis of the current design process and the model of a new process which allows for a better integration of diagnosis related tasks, such as diagnosability analysis, failure-modes-and-effects analysis (FMEA), on-board diagnosis design, in the overall design process of mechatronic subsystems. We then discuss in what way model-based technology can provide tools to support the actual integration and, in particular, present an approach to model-based diagnosability analysis..

Introduction

The importance of diagnosis in onboard automotive systems is constantly growing together with the complexity of the systems. The average dimension of the diagnostic code inside a modern electronic control unit (ECU) is now more than 50% of the whole code. At present, there is no correspondence between such an important role of diagnosis in onboard systems and a similar role that diagnosis could play in the design process chain.

The correct way of dealing with this situation is **to re-organize the design and development chain so that the diagnosis is no longer the last task in the design chain.** This goal provides an opportunity and challenge to model-based systems technology for several reasons. First, in early design stages, when physical prototypes of the designed system are not existing, diagnostic reasoning can only be based on a model. Second, since the design is subject to revisions, the adaptation of diagnostics and fault analysis to such revisions has to happen automatically or, at least, without major efforts. Finally, the existence and use of (simulation) models for the development and validation of control design can provide a basis for the application model-based diagnosis technology.

The European Fifth Framework project „Integrated Design Process for onboard Diagnosis" (IDD) pursues the goal to formalize and standardize the diagnostic design process, and to enable the introduction of diagnosis early in the chain. This methodological goal has to be combined with another important objective: *giving to the designers a set of model-based tools that can help them in evaluating and understanding the effects of each choice on the system being designed.* The IDD project was started February 2000 with a duration of three years and involves both industrial and academic partners: Fiat CRF (Torino), Magneti-Marelli SpA (Torino), PSA, Peugeot Citroen (Paris), Renault (Paris), DaimlerChrysler AG (Stuttgart), OCC'M Software GmbH (München), Universita di Torino, Université de Paris Nord, XIII, and Technische Universität München.

Except for the approach to diagnosability analysis, this paper does not aim at presenting new model-based theories or techniques, but rather focuses on describing the work and intermediate results of this project in order to increase the awareness of this challenge in the field of model-based reasoning. Therefore, we start with a description of the current design process and its deficiencies. Based on this, a new design process is proposed in section 3 that introduces the exchange of models as the major medium for a closer interaction between control design on the one hand and failure-modes-and-effects analysis (FMEA) and diagnostic design on the other hand. Section 4 outlines the technological and software basis chosen by IDD to develop the tools that are required to realize this integrated process. We then present our approach to model-based diagnosability analysis. Finally, we outline the remaining work in the project and list the guiding applications which will be used in the project for validation of the tools.

* This work is supported by the Commission of the European Union (Project no. G3RD - CT199-00058)

Analysis of the Current Process of Design and Generation of Diagnostics

The current processes of each industrial partners have been investigated with a focus on the integration of the diagnostic process and diagnosis-related processes into the whole design process of mechatronic subsystems.
Starting from these results a „merged process“ has been developed that is based on the similarities recognized, ignoring details and small differences. The abstraction of this process will be used as a comprehensive reference for the current design processes. This analysis and its consequences are presented in more detail in [Brignolo et. al. 01].
In the framework presented here we consider especially processes related to **mechatronic** *subsystems*, such as air conditioning or engine control systems. These subsystems involve ECUs as centers of control and diagnostic functions and the physical system, comprising mechanic, hydraulic, electric components. Following [Bortolazzi-Steinhauer 00], Fig. 1 summarizes the overall design, isolating the different phases and showing in which way the process for a subsystem, which is the most interesting one in this project, is related to the entire process.

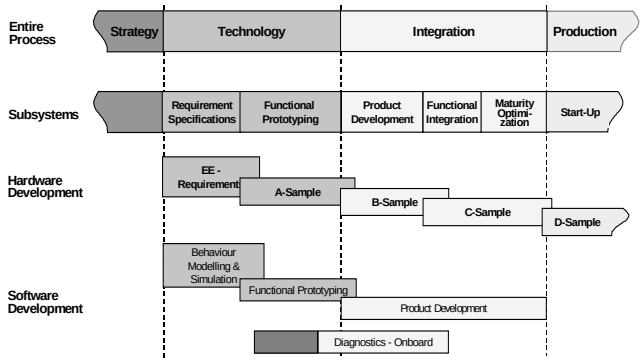


Figure 1 Entire Process and subsystem process, overview

During the ‚strategy phase‘ a first conceptual framework for the new product is worked out, the ‚technology phase‘ targets the concept approval, the ‚integration phase‘ focuses on the realization of the new product by taking into consideration technical feasibility and manufacturing aspects, and, finally, the ‚production phase‘ ensures the industrial mass production with the correct requirements of quality.
The IDD approach focuses primarily on the Technology phase which leads to the first almost complete prototype, but takes into account that a good amount of diagnostic development is performed at present in the Integration phase, as illustrated in Figure 1.

From an abstract point of view, the reference process, which is focussed on the functional prototyping within the technology phase, can be modeled as a set of nested loops:

- **Specifications loop**: Definition of requirements, specifications and implementation of the validated result. In this phase also feedback from after-sales and customers may be involved. Further requirements may be added depending on mock-up observations.
- **Outer design loop**: Design of the whole system prototype, involving the definition of the overall structure of the system, i.e. the selection of the physical (mechanic, hydraulic, electric) components and decisions about the overall layout of the system. This loop terminates when the prototype meets all the requirements and specifications. The core activities are design of the system including its control and diagnosis, comprising a series of inner design loops, and the hardware development of the physical system, which runs in parallel.
- **Inner design loop**: Design of the ECU-based control system and components. Each iteration involves the design of the control algorithms, FMEA, diagnostic development, implementation of the ECU (HW and SW) and verification of the algorithms, as shown in Figure 2. The verification step at the end of the first iterations is performed using models (software/ hardware in the loop), whereas, later, the physical system is used. Depending on the achieved results, there are several iterations, each one of them producing an advanced prototype.

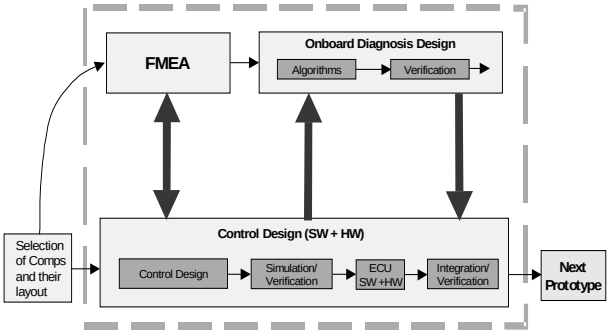


Figure 2 The reference process, one iteration of the inner design loop

Three problem areas in the reference design process have been identified as the essential ones with respect to a better integration of the diagnostic tasks, mainly in the inner and the outer design loops.
The first problem concerns the interaction between the diagnosis design process and the FMEA generation (cf. upper part of Figure 2).

- FMEA and generation of onboard diagnosis are separated and sequential tasks.
- Only few tools support the information extraction process needed for the FMEA, e.g. simulating the consequences of faults or studying interactions between faults. Thus, a lot of work is left to the experience and sensibility of the people that perform FMEA.

The second problem area concerns the interaction between FMEA and the development of diagnostics, and the development and design of control algorithms of the system (cf. Figure 2).

Currently, these are two substantially separate tasks, despite the fact that there are important interdependencies. Examples for possible interactions are:

- a change of the control algorithm may turn a physical component, that was not very essential before, into a critical one and, hence require additional diagnostics,
- a change of the control algorithm promotes the masking of certain faults that were detectable more easily before. Again, additional diagnostics have to take this into account,
- a change of the diagnostics aiming at enhancing diagnosability may exploit additional signals, which may possibly improve control, as well.

As a consequence, requirements and constraints arising from one of these tasks can be dealt with by the other ones only in the next inner design loop, i.e. changes in the design of control algorithms can have impact on FMEA/ diagnosis only during the next inner design loop and vice versa, thus causing additional iterations and time delay.

The third problem area concerns the relation between the design of diagnosis and component selection and layout definition (cf. left-hand part of Figure 2).

The problem here is, that currently the component selection task is external to the inner design loop. As a consequence, for instance the choice or placement of sensor is often not optimized with respect to diagnosis purposes, or, if later changes are made, additional (outer and inner) design loops are needed that cause delays.

An improvement could be reached by performing a comparative analysis (‚what-if-analysis') inside the inner design step and the integration in the early phases of control and diagnostic development. Thus, part of the component selection task is moved inside the inner design process, and, in particular in the early phases of the inner design loop, it is possible and cheap to modify component choices, e.g. sensors, regarding type, sensitivity or placement and to immediately explore the impact on control generation, FMEA, diagnosability analysis, and diagnosis generation.

The New Process

Based on this analysis of the reference process and the outlined improvements, we propose a frame for a new process which is closely connected to a new tool architecture.

In summary, the framework for a new process has to satisfy the requirement that in the inner design loop of the process, the designers (the different experts involved in the design) should be supported in performing different activities in an interleaved way:

- design of the physical system,
- design of control algorithms, and their simulation (for quantitative analysis),
- generation of the FMEA of the designed system
- analysis of the diagnosability, i.e. investigation which faults are detectable and discriminable from each other,
- derivation of on-board diagnosis (OBD) software for the system,
- comparative analysis on the current design (physical system and control), i.e., analysis of the consequences of applying changes to the design both from the control and diagnosability point of view,
- comparative analysis of different design alternatives.

Thus, designers and decision makers are supported in the process of evaluating different designs and in making choices about the best design of a system.

- Such a tight integration of different activities and the aim to perform them concurrently require the fast and reliable exchange of information about any changes in the design introduced by any of the activities. This is why we propose that the **model of the system being designed must play a central role in the new process**, as indicated by Figure 3.
- The aims to update FMEA, diagnosability analysis and OBD generation quickly after a change and to consider different design alternatives in parallel establishes the requirement that these tasks can be effectively supported or automated by computer tools based on the model, i.e. they have to be **model-based tools**.

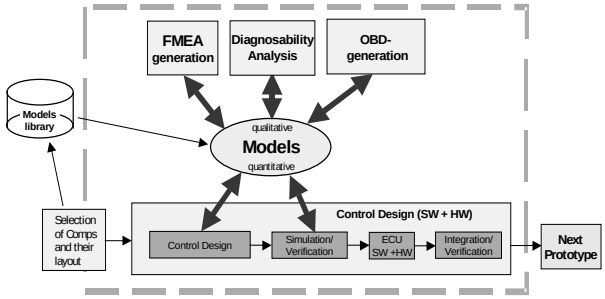


Figure 3 Frame for the new design process

Software Support for the New Process

Accordingly, the actual goal is to provide a new set of functions for supporting the designer, which are realized as ‚software plug-ins' added to the existing software tools for design. Within the scope of IDD, we are considering three plug-ins:

- tools for diagnosability analysis
- tools for supporting the FMEA generation (cf. [Price 98])
- tools for supporting the generation of onboard diagnostics (see e.g. [Bidian et al. 99], [Cascio et al. 99], [Sachenbacher-Struss-Weber 00]).

These tools rely on model-based systems and will be based on a common set of models and a common model-based diagnostic system core.

The new process and the respective tools should be integrated or combined with the simulation tools, that are currently used for the design of control strategies and typically based on quantitative models. In IDD, this is Matlab/Simulink. This requires software that transforms the models created in these environments into qualitative diagnostic models that form the basis for the model-based tools.

Figure 4 summarizes the overall architecture of the new design support system .

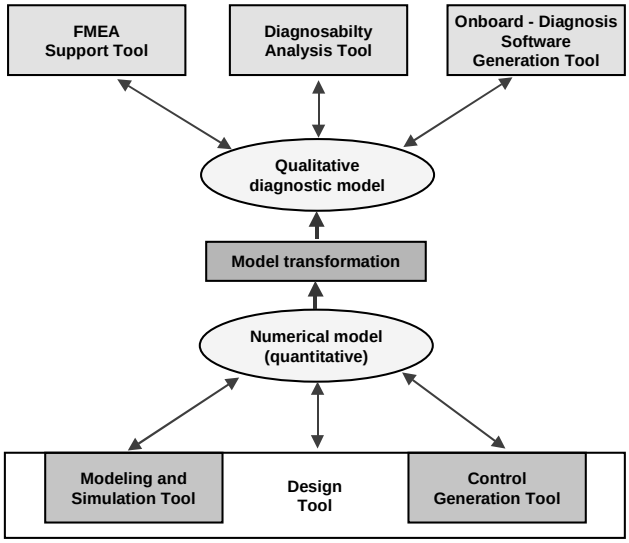


Figure 4 Tools architecture for the new process

A challenge lies in providing

- a common software platform with components that are re-usable in different contexts, and
- the harmonization of models used for different tasks.

The latter is ideally to be achieved by automated transformation routines. In particular the automated transfer of traditional quantitative models (used e.g. for simulation and control design) to qualitative models allowing for automated FMEA and fast, i.e. real-time, on-board diagnosis, is a central target. If indeed successful, the re-use of existing model fragments for **different** tasks will reduce life cycle costs by a significant amount.

IDD envisions three types of application settings:

- an **integrated toolbox** with its own graphical user interface and storage of models. A component-oriented ontology has been chosen to best address modeling requirements in the automotive domain.
- a variety of **plug-ins** to industry-adopted existing tools. In IDD, we have chosen MatLab/Simulink. Models will possibly be stored with these tools and a specific graphical user interface will be limited, if existent at all. The plug-ins provide additional functionality, namely diagnosability analysis, FMEA, and the transformation of design information captured by the Matlab/Simulink model.
- the (on-board) processing scenario for **dedicated applications** such as diagnosis and monitoring. They are dedicated to a particular variant of a device. A diagnosis and monitoring application on a ECU is a typical example.

The IDD toolbox and plug-ins will be running on Microsoft Windows. Therefore, COM (component object model) was chosen as a protocol for the interaction of (binary) components. All the engines, transformers, etc are implemented obeying this standard. This allows for the re-use of functionality in different contexts, and, in particular, the three different application settings. The second cornerstone is given by the use of XML (extended markup language) for describing data in a uniform and exchangeable way. Many of our software components take XML documents as input and produce such documents as output.

COM and XML allow us to build task-related applications that are constructed from components which themselves are aggregated from even more basic components. The components in the layer directly under the application level we call **engines**, our third cornerstone. So, there are (re-usable COM) components that encapsulate a diagnosis engine, an FMEA engine, a predictive engine, a transformation engine, etc. An important consequence of the choice of COM, XML, and engines is that the resulting architecture is an open one, open at any desired degree down to the level of individual methods of low level objects.

At the component level, the IDD consortium has chosen OCC'M's Raz'r [RAZ'R 02] as a basis for implementation. It provides state of the art model-based systems software packaged into COM-components and supplied with XML-interfaces. This allows for further extensions as needed by the consortium requirements. These components include

- an **ATMS** (Assumption Truth Maintenance System) which provides fast consistency checking and handling of time. While still adhering to the basic

framework of assumption-based truth maintenance [de Kleer 86], the employed technology has changed substantially making possible the implementation of on-board systems meeting real-time requirements ([Sachenbacher-Struss-Weber 00]).

- a c**onstraint-based predictive engine** which allows to limit the computational efforts by specifying appropriate foci of attention.
- a **model compiler** which produces system descriptions (**XML** documents) suitable for processing by various engines. For representing constraints, a data structure similar to ordered binary decision diagrams (**OBDD**), but also suitable for direct constraint processing is used as a compact representation [Bryant 92].
- a **diagnosis engine** which accepts a system description and a continuous stream of observations (measurements) as the input and produces an assessment of the current situation by listing the best candidates for diagnosis.
- The **model transformation engine** is central and touches on still open research questions. Therefore, it is a main subject of the consortium's current activities. As already pointed out, automated model transformation is required to obtain qualitative models. Behavioral and structural descriptions are extracted from numerical models (developed in Matlab/Simulink), converted to qualitative models represented in XML form and possibly transformed into more abstract descriptions through a process called task-dependent model abstraction ([Sachenbacher-Struss 01]). The foundations of one of the implementations and a critical discussion of the practical experiences are presented in [Struss 02].

In the following, we discuss the foundations for the diagnosability analysis engine, that forms a specific contribution of the project, in a little more detail.

Diagnosability Analysis Engine

Diagnosability analysis is expected to answer two different types of questions:

"*For a particular design and a chosen set of sensors, determine:*

- **Fault detectability**, *i.e. whether and under which circumstances the possible faults considered can be detected (by the ECU)*
- **Fault** *(class)* **discriminability**, *i.e. whether and under which circumstances the ECU is able to distinguish different classes of faults.*"

The second question is a generalization of the fault identification task ("*Determine the present fault mode unambiguously*"). This generalization is motivated by on-board diagnosis requirements: full fault identification is usually not possible and also not required for on-board purposes, since there is a limited set of possible recovery actions that can be performed by the control unit and which are to be selected dependent on the general type of fault and its severity rather than the individual fault. For instance, only certain critical faults may require immediate shut-off of the engine while others allow continued operation possibly under certain limitations.

Also off-board diagnosis is appropriately characterized as fault class discrimination where the classes comprise the faults of the various smallest replaceable units. More generally, diagnosis is usually a discrimination task whose goal is defined by the available "therapy" actions.

Discriminability is the fundamental task, because detectability can be formulated as discriminability from the normal behavior.

Although the ultimate goal is to discriminate **classes** of behavior modes from each other, the analysis has to based on the discriminability of each pair of **individual** faults taken from any pair of classes, which is unfortunate from a computational point of view.

In our framework, (fault) behavior modes are represented as finite relations, and discriminability analysis becomes the task of computing the observable distinctions between two relations. So, let V_{obs} be the set of observable variables. In an on-board situation, this corresponds to the set of actuator and sensor signals. Since we want to characterize the situations under which detection or discrimination is possible, we introduce a set of variables V_{cause} that are exogenous or "causal " variables w.r.t. the physical system (i.e. the subsystem excluding the ECU). This set includes the actuator signals but also other quantities that influence the behavior of the physical system. Some of the latter may be observables, e.g. the atmospheric pressure, while other are not (directly) measurable, such as the load. Since on-board diagnosis can rely only on what is observable to the ECU, we define:

$$V_{o\text{-}cause} = V_{obs} \cap V_{cause}$$

and

$$V_{obs\backslash cause} = V_{obs} \setminus V_{cause}$$

as well as the respective projections, $PROJ_{obs}$, $PROJ_{o\text{-}cause}$.

The abstract example in Figure 5 will provide an intuition about possible answers to the discriminability question. The vertical axis represents the observable causal variables and the horizontal axis the remaining observables. There may be many unobservable variables, but the shown projection to the space of observables is all that matters.

Two different fault modes (or, more generally, behavior modes) are represented by two relations. As illustrated by the figure, we can distinguish three different cases:

- In the upper section the relations cover each other, i.e. for any causal stimulus in the projection of this intersection area, the observable set of consistent tuples for the two behavior modes are the same, and, hence, they **cannot be discriminated** from each other.

- In the lower section, they are totally disjoint, i.e. any of the respective causal inputs always leads to different system behavior and, thus, **deterministically discriminates** between the two modes.
- For all other causal inputs, the two modes can **possibly be discriminated**, because the actual response of the system may be outside one of the relations, but is not guaranteed to.

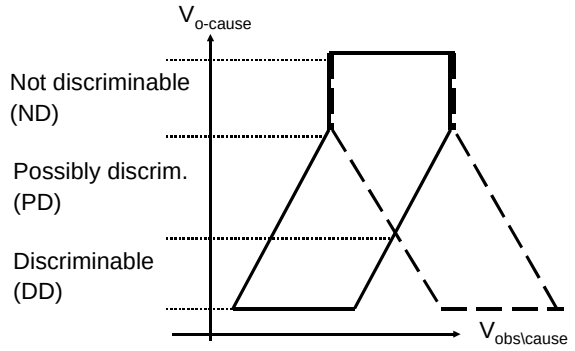


Figure 5 Three categories of discriminability of two behavior modes

With this translation of the task to the analysis of relations, we can also support our previous claim, that, in general, a pairwise comparison of individual modes of required to determine the discriminability of classes of modes. Consider the trivial example of one inverter with two mode classes:

C_1 ={output-stuck-0, output-stuck-1},

C_2 ={shorted, ok}.

Figure 6 a and b display the four faults in the observable space i, o, grouped in the two classes.

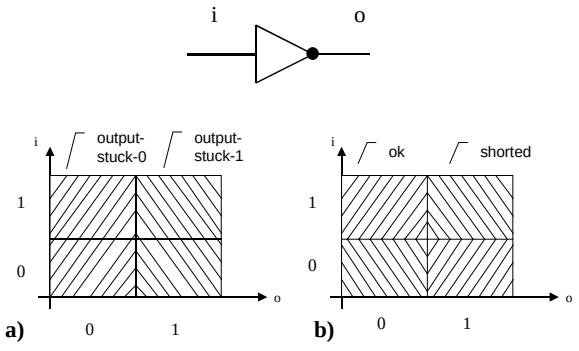


Figure 6 Behavior classes of the inverter for fault classes C_1 (a) and C_2 (b)

Obviously, the faults are pairwise discriminable, and, hence, so are the two classes of faults. However, if we would try to represent each class as the disjunction of its modes and associate with it the union of the respective relations, then both of these class relations cover the entire behavior space and are not distinguishable. The deeper reason is that a fault class represents more than a (exclusive) disjunction of modes. We also make a **persistence assumption**, namely that one particular mode occurs in all inspected situations (i.e. for all inputs).

Before we give formal definitions and computable expressions for the concepts, we introduce one last element: **operating conditions**. This reflects the common practice of distinguishing between ranges of internal or external quantities that result in qualitatively different behaviors and are often reflected by different states of the system and its control. Examples are *engine idle*, *clutch engaged*, *cold engine*, *brake pedal pushed*.

Often, the analysis of fault effects and diagnosability can be restricted to certain operating conditions and is futile for others. For instance, one may not be extremely interested in the detectability of a fault in the air intake system under conditions where the engine is not running (one has to be cautious with such restrictions, though, because firstly, there may be a requirement to perform fault detection beforehand, such as checking the operability of the airbag system or the ABS, and secondly, a broken component could affect operating modes in which it is not intended to be active).

In our approach, an operating condition has to be expressed as a constraint on a subset of model variables. Often, but not always, they will refer to exogenous variables such as the angle of the accelerator pedal or air temperature, and typically, but not exclusively, they are observables (the load, for instance, is not directly observable).

In most cases, the constraint that defines an operating condition will be a conjunction of restrictions on variable values to some interval or state like *temperature>120°C* or *ignition = ON*.

Restricting the analysis to certain operating conditions then boils down to computing the intersection of a behavior relation with their respective constraints.

Definition 1 (Discriminability of behavior modes)

Let $MODEL_{fault1}$, $MODEL_{fault2}$ be the behavior relations of two modes,

OPC_i an operating condition,

and

$SIT \subset DOM(V_{O\text{-}CAUSE})$

a non-empty relation on the observable causal variables.

For OPC_i and SIT, two faults are called

- **not discriminable**, written
  $ND(fault_1, fault_2, OPC_i, SIT)$,

iff

(i) $SIT \subseteq PROJ_{o\text{-}cause}(OPC_i) \setminus PROJ_{o\text{-}cause}$
$(PROJ_{obs}(MODEL_{fault1} \cap OPC_i) \setminus$
$PROJ_{obs}(MODEL_{fault2} \cap OPC_i)$
$\cup PROJ_{obs}(MODEL_{fault2} \cap OPC_i) \setminus$
$PROJ_{obs}(MODEL_{fault1} \cap OPC_i))$

- **deterministically discriminable** , written $DD(fault_1, fault_2, OPC_i, SIT)$,

iff

(ii) $SIT \subseteq PROJ_{o\text{-}cause}(OPC_i) \setminus$
$PROJ_{o\text{-}cause}(PROJ_{obs}(MODEL_{fault1} \cap OPC_i)$
$\cap PROJ_{obs}(MODEL_{fault2} \cap OPC_i))$

- **possibly discriminable** , written $PD(fault_1, fault_2, OPC_i, SIT)$,

iff

$SIT \subseteq PROJ_{o\text{-}cause}(OPC_i) \setminus (SIT_{ND} \cup SIT_{DD})$,

where SIT_{ND} and SIT_{DD} are the maximal relations that satisfy (i) and (ii), respectively.

These definitions characterize the three cases discussed above w.r.t. Figure 6 in a way that can be computed by operations on the extensional constraint representation generated by the model compiler.
Based on the discriminability of modes, discriminability of fault classes can be defined and computed.

Definition 2 (Discriminability of mode classes)

Let $FC_j = \{fault_{i,j}\}$, j =1,2 be two fault classes and OPC_i an operating condition. Let furthermore
$SIT\text{-}SET = \{SIT_{kl}\} \subset P(DOM(V_{o\text{-}cause}))$
be a set of non-empty relations of observable causal variables. FC_1, FC_2 are called

- **not discriminable**, written $ND(FC_1, FC_2, OPC_i)$

iff there exists a pair of modes that is completely non-discriminable:

$\exists fault_{1k} \in FC_1 \quad \exists fault_{2l} \in FC_2$
$ND(fault_{lk}, fault_{2l}, OPC_i, PROJ_{o\text{-}cause}(OPC_i))$

- **deterministically discriminable**, written $ND(FC_1, FC_2, OPC_i, SIT\text{-}SET)$,

iff each pair of modes is deterministically dicriminable for some element of SIT-SET:

$\forall fault_{1k} \in FC_1 \quad \forall fault_{2l} \in FC_2 \; \exists SIT_{kl} \in SIT\text{-}SET$
$DD(fault_{lk}, fault_{2l}, OPC_i, SIT_{kl})$

- **Possibly discriminable**, written $PD(FC_1, FC_2, OPC_i, SIT\text{-}SET)$,

otherwise, iff all SIT_{kl} are in the complement of the non-discriminable situations:

$\forall_{kl} SIT_{kl} \cap SIT_{ND,kl} = \varnothing$

Status and Future Work

As of now, two different alternatives have been implemented to generate the qualitative diagnosis models from existing numerical models which both use Matlab itself to compute the tuples of the modeling relation. In addition, a library of qualitative models will be created manually that allows to configure the model based on the structural description only. Based on a use case analysis, the core of the diagnosability analysis tool and the model-based on-board diagnosis engine have been developed.
IDD will use a number of guiding applications with the goal to demonstrate how the diagnostic tasks described can be performed by using the new process and the new tools architecture. Furthermore, we aim to demonstrate how additional advantages of the new method can be achieved, e.g. optimization of sensor placement or deeper diagnostic performance. Thereby, the guiding applications serve, on the one hand, as case studies for the application of the new techniques and, on the other hand, as test cases and demonstrators of the results of the project.
The guiding applications chosen cover on the one hand different mechatronic systems with central ECU-functions, and on the other hand the general application of diagnostic tasks to multiplexed architecture systems. They include

- The **air delivery system** for diesel engines (Figure 7), comprising the exhaust gas turbocharging system and the exhaust gas recirculation system (EGR. and the Common Rail Injection System (Fiat and Magneti-Marelli).

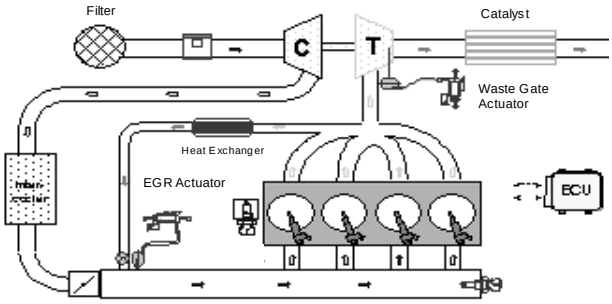


Figure 7 Guiding application: Air delivery system

- The **cooling system** (DaimlerChrysler AG), including an intercooler, which on the one hand increases the efficiency of the engine by cooling the compressed air and, hence, increasing the air charge rate, and on the other hand decreases NOx emissions by keeping the combustion at lower temperature (Figure 8).
- The a**ir conditioning system** (Peugeot Citroën PSA) which consists of two loops that supply a cold heat exchanger and a hot heat exchanger (Figure 9).

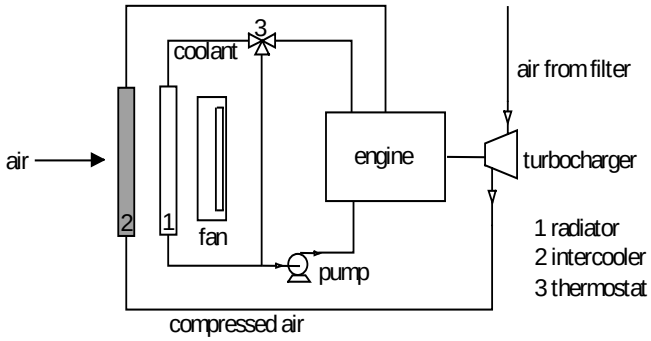


Figure 8 Guiding application: Cooling system

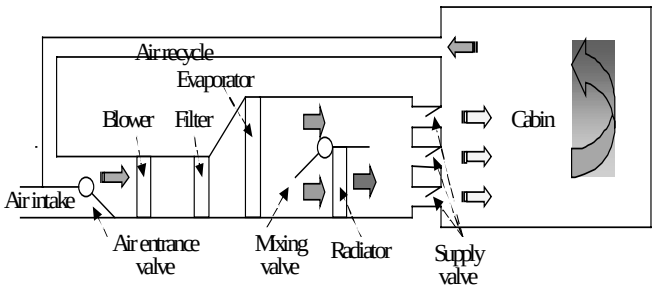


Figure 9 Guiding application: Air conditioning system

- The **multiplexed architecture** (Renault) involving ECUs, sensors, actuators, functions (EF = elementary functions), busses and data frames (Figure 10). The design engineer will be enabled to run a program directly on the representation of a designed architecture and receive the results of an analysis of the interdependency of faults and functions in this architecture.

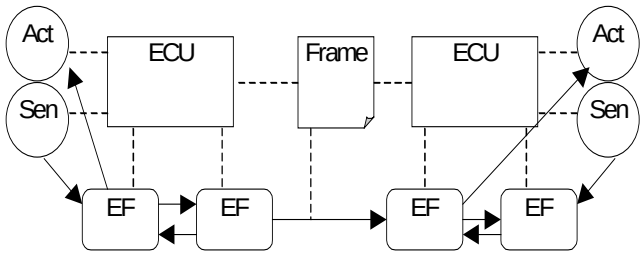


Figure 10 Guiding application: Multiplexed architecture

A first version of models for these guiding applications has been developed and will be used to validate and improve the model abstraction module and to evaluate the tools. By the end of the project in January 2003, we hope to demonstrate the utility of the tools and the benefits of the modified design process based on examples that are close to reality.

References

[Bidian et al. 99] P. Bidian, M. Tatar, F. Cascio, D. Theseider-Dupré, M. Sachenbacher, R. Weber, C. Carlén: Powertrain Diagnostics: A Model-Based Approach, Proceedings of ERA Technology Vehicle Electronic, Systems Conference '99, Coventry, UK, 1999

[Bortolazzi-Steinhauer 00] J. Bortolazzi, St. Steinhauer, Th. Weber: Development and Quality Management of In-Vehicle Software. In: Electronic Systems for Vehicles (VDI – Berichte 1547), VDI Verlag , Duesseldorf 2000

[Brignolo et a. 01] R. Brignolo, F. Cascio, L. Console, P. Dague, P. Dubois, O. Dressler, D. Millet, B. Rehfus, P. Struss. Integration of Design and Diagnosis into a Common Process. In: Electronic Systems for Vehicles, pp. 53-73. VDI Verlag, Duesseldorf, 2001.

[Bryant 92] R. Bryant: Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams ACM Computing Surveys, Vol. 24, No. September 1992

[Cascio et al. 99] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, D. Theseider-Dupré: Strategies for on-board diagnostics of dynamic automotive systems using qualitative models, AI Communications, June 1999.

[de Kleer 86] J. de Kleer: An assumption-based truth maintenance system, Artificial Intelligence 28, 1986

[Price 98] C. Price: Function-directed Electrical Design Analysis, AI in Engineering 12(4), pp. 445-456, 1998.

[RAZ'R 02] Raz'r Version 1.6, Occ'm Software GmbH, see http://www.occm.de

[Sachenbacher-Struss-Weber 00] M. Sachenbacher, P. Struss, R. Weber: Advances in Design and Implementation of OBD Functions for Diesel Injection Systems based on a Qualitative Approach to Diagnosis, SAE 2000 World Congress, Detroit, USA, 2000.

[Sachenbacher-Struss 01] M. Sachenbacher, P. Struss: AQUA: A Framework for Automated Qualitative Abstraction. In: Working Papers of the 15th International Workshop on Qualitative Reasoning (QR-01), San Antonio, USA, 2001

[Struss 02] P. Struss: Automated Abstraction of Numerical Simulations Models - Theory and Practical Experience. In: Sixteenth International Workshop on Qualitative Reasoning, Sitges, Catalonia, Spain, 2002.

Suggestions from the software engineering practice for applying consistency-based diagnosis to configuration knowledge bases

Gerhard Fleischanderl

Siemens AG Österreich, Program and System Engineering, CES Design Services
Erdberger Laende 26, A-1030 Vienna, Austria
gerhard.fleischanderl@siemens.com

Abstract

A configuration knowledge base is software that needs debugging during maintenance and can benefit from consistency-based diagnosis. The paper describes suggestions and practical experience from the introduction of this diagnosis technique in the work flow for maintaining configuration knowledge bases. Consistency-based diagnosis is suitable for detecting bugs in knowledge bases, but needs tailoring to fit in the work flow of the knowledge engineers.

1 Introduction

Configurators have already been applied to different industry domains. For instance, telecommunication systems are among the products successfully handled with configurators. The crucial information is in the knowledge bases of the configurators.

Configurators using declarative constraints [Mittal and Frayman, 1989] are in everyday use and can generate and modify configurations with more than 50,000 objects [Fleischanderl *et al.*, 1998]. Declarative constraints offer easier maintenance compared to procedural specifications, but also benefit from effective debugging methods. Consistency-based diagnosis [Reiter, 1987] [Greiner *et al.*, 1988] is applicable to fault detection in configuration knowledge bases [Felfernig *et al.*, 2000], which is the topic of this paper. The extensions towards hierarchical models [Felfernig *et al.*, 2001] are not discussed here because the author did not apply this yet.

This paper discusses suggestions and practical experience from applying diagnosis techniques to the debugging of declarative knowledge bases for configurators. The experience ranges from the planning of an engineering process including diagnosis to the early adoption of diagnosis for the debugging of knowledge bases. The requirements of the development process for knowledge bases are compared with the specification of the diagnosis method.

2 Maintaining knowledge bases

Creating and maintaining knowledge bases is essentially a software engineering process.

After collecting and analyzing new requirements, the knowledge base is modified and tested. Regression tests are essential for long-term maintenance. So the results from replaying regression tests should be fed into a diagnosis tool if the new output differs from the expected output of a regression test.

In an ideal world the discrepancies from regression tests would be analyzed with a diagnosis tool and suggestions be made which constraints in the knowledge base are responsible for the discrepancies. Unfortunately this is not that easy.

3 Preconditions for consistency-based diagnosis

Consistency-based diagnosis needs a consistency checker, i.e. a solver that yields conflict sets when a knowledge base is in contradiction to a positive example. The configurator kernel COCOS [Stumptner *et al.*, 1998] applied by the author is a solver that uses declarative constraints for statically checking or expanding a partial configuration. The kernel was extended to also yield conflict sets. So a sufficiently powerful consistency checker is available.

The elements that can be faulty have to be identifiable parts of a knowledge base. In our case the constraints can be faulty with respect to positive examples and are the "components" for model-based diagnosis.

4 Requirements and consequences of consistency-based diagnosis

4.1 Definition of a CKB-diagnosis

A CKB-diagnosis (i.e. diagnosis of configuration knowledge bases) uses the model-based diagnosis paradigm and is defined as follows [Felfernig *et al.*, 2000].

Definition (CKB-Diagnosis Problem): A CKB-Diagnosis Problem is a triple (DD,E+,E-) where DD is a configuration knowledge base, E+ is a set of positive and E- of negative configuration examples. The examples are given as sets of logical sentences. It is assumed that each example on its own does not contain inconsistencies.

Definition: A CKB-diagnosis for a CKB-Diagnosis Problem (DD,E+,E-) is a set $S \subseteq DD$ of sentences such that there exists an extension EX, where EX is a set of logical sentences, such that

$DD - S \cup EX \cup e+$ consistent $\forall e+ \in E+$

$DD - S \cup EX \cup e-$ inconsistent $\forall e- \in E-$

Let NE be the conjunction of all negated negative examples. This is the most easily found EX.

Proposition: Given a CKB-Diagnosis Problem (DD,E+,E-), a diagnosis S for (DD,E+,E-) exists iff

$\forall e+ \in E+ : e+ \cup NE$ is consistent.

Corollary: S is a diagnosis iff

$\forall e+ \in E+ : DD - S \cup e+ \cup NE$ is consistent.

4.2 Representation of examples

The definition of a CKB-Diagnosis Problem says that the examples are given as sets of logical sentences. This is usually not the case in configurator implementations. Yet, databases or other data representations can easily be transformed into facts, i.e. logical sentences. This transformation need not be done for the implementation of diagnosis for configurator knowledge bases, but is a precondition for the applicability of CKB-diagnosis.

With logical sentences one can define a configuration as a set of fragments. In configurator applications, configurations are based on an object model, which is usually defined with UML. All objects usually are reachable from one entry object. So the positive or negative examples cannot just be isolated sub-configurations, but must be connected objects. This is a slight restriction that does not limit the diagnosis.

This property of configurations ensures that trivial inconsistencies are avoided, e.g. there cannot be two modules in the same slot. Therefore each example (i.e. its structure of objects and connections) does not contain inconsistencies among its elements.

4.3 Conjunction of negated negative examples

The definition of a CKB-diagnosis requires an extension EX. The question is: Where does EX come from?

The simplest EX would be the negation of all negative examples, i.e. NE as defined above. This is not a useful solution for maintaining configurator knowledge bases in real life. This would reduce the advantages of declarative constraints, namely that knowledge bases contain little redundant information and can be understood easily by domain experts. Furthermore, the constraints should be sufficiently general to be applicable to similar situations in the future. The negation of configurations (i.e. negative examples) would clutter the knowledge base with facts that may overlap and would not prevent examples that are slightly different.

4.4 Diagnosis is part of the existing knowledge base

According to the definition of CKB-diagnosis, a diagnosis is a subset of the knowledge base. That means faults are found among the constraints in the existing knowledge base. This is useful in real-life projects and makes the consistency-based diagnosis worthwhile. Yet, defining new constraints (thus extending the knowledge base) has to be accomplished with other approaches.

5 Integrating consistency-based diagnosis in the software engineering process

The definitions for consistency-based diagnosis of configuration knowledge bases do not tell a lot about how to proceed (step by step) to reach a correct knowledge base. However, the conditions for the correctness check for knowledge bases are specified.

This section describes how to use diagnosis in the software engineering process for knowledge bases.

5.1 Use the examples one by one

Examples, i.e. stored configurations, may be partial or complete. Due to restrictions coming from the usual object models in software development, each example is a network of objects that can be reached from an entry object. Therefore, only one example can be loaded at one time. This holds for positive and negative examples.

5.2 Negative examples are outsiders

In the diagnosis process discussed here, negative examples do not yield hints for mistakes in a knowledge base.

We expect that negative examples lead to inconsistencies. If a negative example is consistent with the knowledge base, the consistency-based diagnosis has no discrepancy to start from. The practical suggestion then is to analyze the consistent negative examples "by hand" and modify the knowledge base to rule out those examples. This corresponds to finding the mysterious EX in the definition of CKB-diagnosis.

The good news, however, is that negative examples usually are modifications of positive examples or previously positive examples that became negative after a modification to the knowledge base. Our experience from maintenance over many years shows that these negative examples will mostly remain negative examples after more modifications to the knowledge base.

Help also comes from good practice in software engineering. When knowledge bases are stored in a version control (configuration management) system, we can find the latest previous version where some negative example was still rejected by the knowledge base. Comparing that older version with the current knowledge base shows the constraints that were modified or removed in the meantime. This is of course an excellent starting point for modifying

the current knowledge base such that it again rejects the negative example.

When all negative examples are rejected by the knowledge base, start looking at the positive examples. So the negative examples are treated outside the diagnosis step.

5.3 Use the results from regression tests

Like any software, knowledge bases can be maintained more efficiently by using regression tests and checking them after a modification.

When a regression test produces an output different from its reference, find out whether the new output is expected (after a modification to the knowledge base). Only if the new output is different from what is expected, feed this output into diagnosis.

5.4 Do diagnosis and repeat the cycle

Finally, we use consistency-based diagnosis to detect faults in the knowledge base. This follows the definition of CKB-diagnosis as described above. The well-defined preconditions and semantics of the method make it particularly valuable.

After the knowledge base was modified, we must repeat the cycle of testing and diagnosis until all negative examples are inconsistent and all positive ones are consistent.

The cycle described here starts with the negative examples (by modifying or extending the knowledge base) and continues with the positive examples (by modifying or reducing the knowledge base). This could be done the other way round. The "optimal" sequence, however, depends on the structure of the knowledge base and the expert's experience and point of view. The objective is to modify the knowledge base such that it remains easy to maintain and easy to understand. We are confident that the steps described above help us get close to this objective.

6 Beyond diagnosis

Beyond the scope of CKB-diagnosis, other methods can be useful for maintaining knowledge bases.

Automatic generation of test cases would be helpful for producing a large set of regression test cases. This would assure the quality of knowledge bases that are maintained over several years.

If a negative example is consistent, automatic generalization of the negated negative example could yield a non-redundant modification to the knowledge base. Here the optimum between introducing too many new constraints and over-generalization has to be found. For this purpose the methods for automatic learning of concepts have to be analyzed with respect to the semantics of the configuration knowledge base.

7 Summary and conclusion

Consistency-based diagnosis is applicable to the debugging of configuration knowledge bases. The method is particularly valuable because of its well-defined preconditions and semantics.

Integrating CKB-diagnosis in the software engineering process for knowledge bases can be done efficiently and effectively. There are minor limitations where CKB-diagnosis cannot be fully applied, i.e. with respect to automatic suggestions from negative examples. Altogether the experience from the planning of a debugging process with diagnosis and from the early adoption is encouraging. Results from wide usage will follow.

Acknowledgement

I want to thank Gerhard Friedrich and Dietmar Jannach for their valuable contributions to our discussions.

References

[Felfernig *et al.*, 2000] Alexander Felfernig, Gerhard E. Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based Diagnosis of Configuration Knowledge Bases. *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000),* pp. 146-150, Berlin, Aug. 2000, IOS Press.

[Felfernig *et al.*, 2001] Alexander Felfernig, Gerhard E. Friedrich, Dietmar Jannach, and Markus Stumptner. Hierarchical diagnosis of large configurator knowledge bases. *Working Notes of the 12th Intl. Workshop on Principles of Diagnosis (DX-2001),* Via Lattea, Italy, March 2001.

[Fleischanderl *et al.*, 1998] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems & their applications,* 13(4):59-68, July/Aug. 1998.

[Greiner *et al.*, 1988] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A Correction to the Algorithm in Reiter's Theory of Diagnosis. *Artificial Intelligence*, 41(1):79-88, Nov. 1989.

[Mittal and Frayman, 1989] Sanjay Mittal and Felix Frayman. Towards a generic model of configuration tasks. *Proceedings of the 11th Intl. Joint Conference on Artificial Intelligence (IJCAI-1989),* pp. 1395-1401, Detroit, Aug. 1989, Morgan Kaufman Publishers.

[Reiter, 1987] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57-95, Apr. 1987.

[Stumptner *et al.*, 1998] Markus Stumptner, Gerhard E. Friedrich, and Alois Haselböck. Generative Constraint-Based Configuration of Large Technical Systems. *AI-EDAM (Artificial Intelligence for Engineering, Design, Analysis and Manufacturing)*, 12(4):307-320, Special Issue on Configuration, Sep. 1998.

Consistency-Based Fault Isolation for Uncertain Systems with Applications to Quantitative Dynamic Models

Colin N. Jones [1] and **Gregory W. Bond** [2] and **Peter D. Lawrence**[3]

Abstract. This paper presents the Probabilistic General Diagnostic Engine (PGDE), a novel method of offline consistency-based fault isolation. Many existing proposals require qualitative logic models for consistency-based diagnosis due to their ability to speed the search for conflict sets through the use of an ATMS. However, for many applications, quantitative dynamic models are preferred or already available. The key strength of the PGDE is that it allows the use of any modelling language for which an appropriate calculation engine can be written. It also offers graceful degradation in the presence of uncertainty, commonly caused by noise or modelling errors. Finally, given perfect knowledge, it can be shown that the PGDE computes the same result as existing consistency-based diagnosis methods. To demonstrate the performance of the algorithm, we have used a quantitative dynamic model of the fluid power circuit of a single-degree of freedom hydraulic test bench and developed an appropriate calculation engine for computing consistency between measured values and predicted results. Various failures were generated on the physical test bench and the PGDE isolated the faults with approximately 85% accuracy.

1 INTRODUCTION

Consistency-based diagnosis has at its heart the search for a subset of the full model such that predictions made using the subset are consistent with sensor measurements. This search space is exponential in the number of model components and so a great deal of attention has been given to developing efficient algorithms. Much progress has been made by utilizing the properties of propositional logic and qualitative models ([10, 8, 1] to name a few) but the problems associated with more complex dynamic systems have still to be solved in general. The Probabilistic General Diagnostic Engine (PGDE) addresses some of these issues in a general framework that applies to any model for which an appropriate "consistency measure" can be formulated.

There are many devices for which quantitative dynamic models either already exist or whose behavior can best be described by a set of differential equations. The cost of developing qualitative models exclusively for the purpose of diagnosis is prohibitive, thus making the adaptation of qualitative methods to quantitative dynamic models an important topic. Models of this type present two new challenges to the diagnostician: First, quantitative dynamic models require the comparison of sets of signals to determine consistency. Due to noise and modelling errors, it can be difficult to represent the results of these comparisons by the discrete values typically used in qualitative methods. Second, the nature of dynamic systems is that they often have states which are not directly measurable. When the model is simulated using only the equations from a few components, it is often the case that many of the states will become unknown. If no conflict is observed, we reason that a possible diagnosis has been identified, however, it is impossible to know if there would have been a conflict if these states had been known. As a result, the underconstrained nature of dynamic systems reduces the resolution of fault isolation procedures and this must be taken into account in any diagnostic method dealing with these models.

The PGDE algorithm attempts to deal with these difficulties by maintaining a belief distribution for each possible diagnosis. Since these distributions are not limited to discrete-valued consistency measures, the PGDE is able to more accurately interpret intermediate non-boolean consistency assessments. They are also updated throughout the duration of the diagnostic procedure, and conclusions about the consistency of sets of components with observations are not drawn until sufficient information has been processed. In Section 2, the proposed algorithm is laid out in a step-by-step fashion, including consideration of its computational complexity in Section 2.5. Next, Section 3 presents a non-trivial example hydraulic circuit and summarizes some diagnostic results obtained by the PGDE. Finally, the paper closes with a discussion of conclusions and future directions of research in Section 4.

2 PGDE ALGORITHM

The model used in a consistency-based algorithm is a set of constraints on the signals passing through the system. A failure can be declared when these signals are inconsistent with the constraints. The goal of the algorithm is then to locate a subset of these constraints, which when removed from the model, restore consistency between the predicted and observed behavior. This process can proceed in an iterative manner, selecting a set of constraints to remove and simulating the system until a feasible set is found.

We begin by defining the system as in [7]:

Definition 1 *A system is a triple (SD,COMPS,OBS) where:*

1. *the components (COMPS) are a finite set of constants*
2. *the system description (SD) is a set of constraints*
3. *the observations (OBS) are measurements of the physical device*

There is no requirement that there be a one-to-one mapping from components to constraints and so a partition $\{SD_c\}_{c\in COMPS}$ is defined covering SD such that $\bigcup_{c\in COMPS} SD_c = SD$ and $SD_{c_i} \bigcap SD_{c_j} = \emptyset \ \forall c_i \neq c_j$. The set of all possible failures is

[1] Cambridge University, Engineering Department, Trumpington St., Cambridge, CB2 1PZ, U.K.
[2] AT&T Labs - Research, 180 Park Avenue, Rm. D273, Bldg. 103, Florham Park, NJ, P.O. Box 971, U.S.A
[3] University of British Columbia, EECE Department, 2356 Main Mall Vancouver, BC, V6T 1Z4, Canada

given by the power set of $COMPS$ and for each element $\Delta \subseteq P(COMPS)$, define $SD_\Delta = \bigcup_{c\in\Delta} SD_c$. This allows the definition of components which contain large numbers of constraints or complex behaviors as well as hierarchies of components. The cardinality of a set of constraints $X \subseteq SD$ is written as $|X|$; it is a system-dependent real number, representing the notion of how "large" the set X is when compared to SD.

Reiter's original work [7] relies on a 'theorem prover', $\texttt{TP}(SD, \mathcal{D}(\Delta, COMPS\backslash\Delta), OBS)$, which returns true if the partial model containing only the constraints in the complement of SD_Δ, $(SD_\Delta)^c$, is consistent with the observations OBS and false otherwise; consistency implying that the components Δ are a possible diagnosis. Here the theorem prover is redefined to return a continuous measure of how consistent the constraints $(SD_\Delta)^c$ are with the observations OBS. It is possible that the system defined by $(SD_\Delta)^c$ with OBS as inputs may be underconstrained. Thus, for some of the constraints in $(SD_\Delta)^c$, it is impossible to verify if they have, or have not, been violated. If this system is consistent then it is not valid to say that Δ is a diagnosis as the faults might have been in the constraints that could not be tested. This situation is very common in dynamic systems with state as they are inherently underconstrained [4]. To deal with this, the constraints which were used during the simulation of $(SD_\Delta)^c$ are returned by $\texttt{TP}(\cdot)$ as defined below.

Definition 2 *Let $\Delta \in P(COMPS)$. Define the function $\texttt{TP}(\cdot,\cdot) : SD \times OBS \rightarrow \mathbb{R} \times SD$ as:*

$$(\mu_\Delta, A_\Delta) = \texttt{TP}((SD_\Delta)^c, OBS)$$

Where:

- *$\mu_\Delta \in [0,1]$, 1 implies constraints $(SD_\Delta)^c$ are consistent with the observations OBS, and 0 implies inconsistency*
- *$A_\Delta \subseteq (SD_\Delta)^c$ are the constraints which $\texttt{TP}(\cdot)$ had sufficient information to apply during the calculation of μ_Δ*

Two belief distributions over the states $\{true, false, unknown\}$ are maintained for each element $\Delta \in P(COMPS)$. These are represented by the probability mass functions $B_{\text{D},\Delta}(x)$ and $B_{\text{IC},\Delta}(x)$ with domains $\{true, false, unknown\}$. $B_{\text{D},\Delta}(true)$ is the belief that the evidence, provided by calls to $\texttt{TP}(\cdot)$, shows that Δ is a diagnosis. $B_{\text{D},\Delta}(false)$ is the belief that the evidence does not show that Δ is a diagnosis. It does not mean that the evidence *does* show that Δ is *not* a diagnosis as consistency can only incriminate components, it cannot exonerate them [7]. Finally, $B_{\text{D},\Delta}(unknown)$ is the probability that it is unknown what the evidence shows, or that there is no evidence. If $\mu_\Delta = 0$ then at least one component of Δ^c must be faulty and we call Δ^c a conflict set [7] and Δ an inverse conflict. $B_{\text{IC},\Delta}(true)$ is the belief that the evidence shows that Δ is an inverse conflict, $B_{\text{IC},\Delta}(false)$ that it doesn't and $B_{\text{IC},\Delta}(unknown)$ that the evidence is unclear.

Initially, all the beliefs are 100% *unknown* ($B_{\text{D},\Delta}(x) = B_{\text{IC},\Delta}(x) = \{0.0, 0.0, 1.0\}$). In each iteration, a call is made to $\texttt{TP}(\cdot)$ to check if a new set of constraints $(SD_\Delta)^c$, is consistent with the observations, OBS. The distributions are then updated to reflect the simulator's certainty in the consistency of each set of components, again with the observations. In this way, the diagnostic engine determines the components that are most likely to be faulty, as well as a measure of its confidence in these decisions.

A block diagram of the PGDE is shown in Figure 1. The following sections deal with each stage of the algorithm in detail in the order: updating the beliefs (steps 3 and 4), choosing a new set to test for consistency via $\texttt{TP}(\cdot)$ (step 1), deciding when to stop and interpreting the final belief distributions (steps 5 and 6).

2.1 Belief update

Once a possible diagnosis, Δ, has been selected, $\texttt{TP}(\cdot)$ is used to find the consistency measure, μ_Δ, and the constraints which were used to compute it, A_Δ. The goal is to determine what the consistency measure has shown about each of the subsets of $COMPS$, using A_Δ as a guide. Assuming no fault models, two properties of constraint systems allow the consistency measure of the set Δ to affect the beliefs of other sets: supersets of diagnoses are diagnoses (removing more constraints will not make the system inconsistent) and subsets of inverse conflicts are inverse conflicts (adding constraints will not make the system consistent). Using these facts, the supersets of Δ are first considered and the information derived from μ_Δ and A_Δ is used to update the beliefs that they are diagnoses ($B_{\text{D},\Delta_P}(x)\ \forall\Delta_P \supseteq \Delta$). Similarly, the beliefs that the subsets are inverse conflicts are also updated ($B_{\text{IC},\Delta_C}(x)\ \forall\Delta_C \subseteq \Delta$).

2.1.1 *Update belief in diagnosis*

We begin by assuming that $\mu_\Delta = 1$, indicating that the observations are consistent with the constraints $(SD_\Delta)^c$. The goal is to determine to what degree this evidence shows that each set is a diagnosis. The first step is to locate the *base set*, Δ_B, for the set $(SD_\Delta)^c$ as defined below in Definition 3. This is the set with the most components of which none have had any of their constraints used during the calculation of μ_Δ. Referring to Figure 2, in which $\texttt{TP}((SD_{\{1,2,3\}})^c, OBS)$ was called, the base node is $\Delta_B = \{1,2,3,4\}$. If $\Delta \neq \Delta_B$, then the constraints of at least one component have not been considered due to the assumption that the components in Δ were faulty (in Figure 2 this would be component 3). In essence, $\texttt{TP}(\cdot)$ cannot distinguish between any set Δ' such that $\Delta \subseteq \Delta' \subseteq \Delta_B$, since whenever the constraints associated with the components in Δ are not considered, neither are those of Δ_B, which implies that $\mu_\Delta = \mu_{\Delta'} = \mu_{\Delta_B}$. This is a limitation of the model and the placement of the sensors; as a result the best the algorithm can do is incriminate Δ_B and inform the user of this sensor deficiency. Because the consistency measure would be the same for all of the sets Δ', such that $\Delta \subseteq \Delta' \subseteq \Delta_B$, the sets are marked and ignored in subsequent calls to $\texttt{TP}(\cdot)$. For certain model types these families of sets can be identified *a priori* and grouped into single components to speed the algorithm [1, 2].

Definition 3 *Let $\Delta \subseteq \Delta_B \subseteq COMPS$. Then Δ_B is the base set for Δ iff*

$$SD_{\Delta_B} \bigcap A_\Delta = \emptyset$$
$$\forall\Delta' \supset \Delta_B,\ SD_{\Delta'} \bigcap A_\Delta \neq \emptyset$$

If the constraints associated with Δ_B are not considered during the call to $\texttt{TP}(\cdot)$, those in $(A_\Delta)^c \setminus SD_{\Delta_B}$ are not either (in Figure 2 this would be the unshaded sections of components 5 and 6). These are the constraints which were *not* considered that do *not* make up a full component. The question is: Is the lack of conflict during the computation of μ_Δ due to the constraints in SD_{Δ_B}, those in $(A_\Delta)^c \setminus SD_{\Delta_B}$, or some combination of the two? The safest approach would be to say that this evidence can only increase the belief that some set $\Delta' \supseteq \Delta_B$ which covers all of $(A_\Delta)^c$ is a diagnosis ($\Delta' = \{1,2,3,4,5,6\}$ in the example). However, if

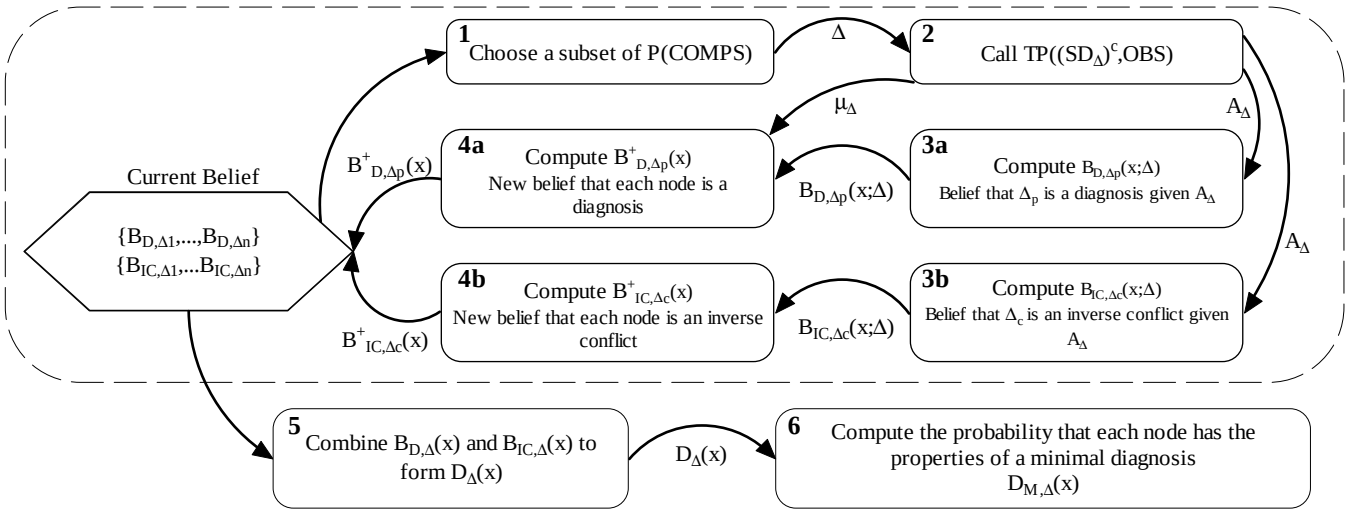


Figure 1. The PGDE Algorithm

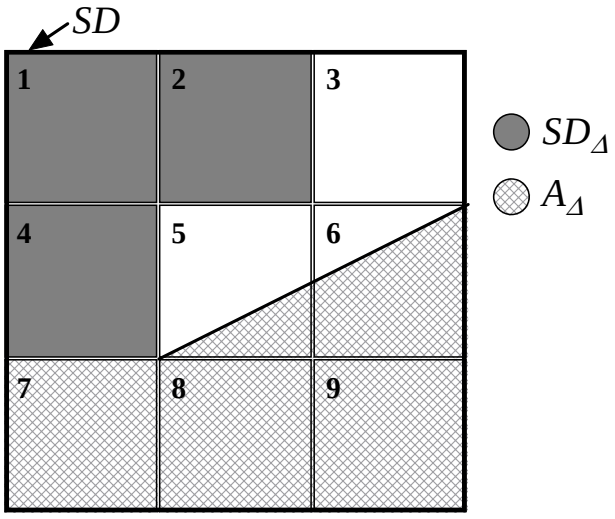


Figure 2. Example nine component system

$|(A_\Delta)^c \setminus SD_{\Delta_B}| \ll |SD_{\Delta_B}|$, this would be a very conservative approach, in the sense that a set will never be called a diagnosis if it cannot completely explain the observed behavior, and multiple component failures would be returned more often than they should. In most cases, designing models which reduce the size of $(A_\Delta)^c \setminus SD_{\Delta_B}$ will increase the precision of the diagnosis and so we make the assumption that most modelers will aim for this characteristic and as a result assume that $|(A_\Delta)^c \setminus SD_{\Delta_B}|$ is small compared to $|SD_{\Delta_B}|$.

Under the assumption that the majority of the constraints which were not considered during the computation of μ_Δ belong to Δ_B, this evidence increases the belief that Δ_B is a diagnosis. However, because every superset of a diagnosis is a diagnosis, this evidence also increases the belief that all of the supersets of Δ_B are diagnoses. Therefore for each set $\Delta_P \supseteq \Delta_B$ the probability that the constraints in SD_{Δ_P} can account for the lack of conflict during the computation of μ_Δ is:

$$P(\Delta_P \text{ is a diagnosis} \mid A_\Delta \wedge \mu_\Delta = 1) = \frac{|(A_\Delta)^c \bigcap SD_{\Delta_P}|}{|(A_\Delta)^c|} \quad (4)$$

Assuming that faults are equally likely to be anywhere in $(A_\Delta)^c$, the probability that they are in SD_{Δ_P} is given by Equation 4, as the proportion of $(A_\Delta)^c$ that is covered by SD_{Δ_P}. If all of, or more than, $(A_\Delta)^c$ is covered, then the probability that the system will be consistent is 100%, by the assumption that $\mu_\Delta = 1.0$.

This probability is computed assuming $\mu_\Delta = 1$, when in fact it may well be less than one. The consistency measure describes our ability to measure how consistent the observations are with the constraints A_Δ. The real components Δ are either consistent or inconsistent with observations and it is only the inability of the model and sensors to perfectly determine which one is true that causes $\mu_\Delta < 1$. Therefore the consistency measure can be interpreted as a probability that the real artifact is consistent or inconsistent and we assume a mapping $\mathcal{PC}(\mu_\Delta)$ to $[0, 1]$ defined by the modeler which represents how probable it is that the real artifact is consistent given μ_Δ.

For each $\Delta_P \supseteq \Delta_B$ we define a belief distribution $B_{\text{D},\Delta_P}(x; \Delta)$ over the states $\{true, false, unknown\}$ which represents the belief that Δ_P is a diagnosis given only the information from calling $\texttt{TP}(\cdot)$ on Δ. The distribution is defined as follows:

$$\begin{aligned} &B_{\text{D},\Delta_P}(true; \Delta) \\ &\quad = \quad P(\Delta_P \text{ is a diagnosis} \mid A_\Delta \wedge \mu_\Delta = 1) \cdot \mathcal{PC}(\mu_\Delta) \\ &B_{\text{D},\Delta_P}(false; \Delta) \\ &\quad = \quad (1 - P(\Delta_P \text{ is a diagnosis} \mid A_\Delta \wedge \mu_\Delta = 1)) \cdot \mathcal{PC}(\mu_\Delta) \\ &B_{\text{D},\Delta_P}(unknown; \Delta) \\ &\quad = \quad 1 - \mathcal{PC}(\mu_\Delta) \end{aligned} \quad (5)$$

Equation 5 takes the probability that a set is a diagnosis given A_Δ and that the measure is consistent, and then scales this probability by the certainty that the call to $\texttt{TP}(\cdot)$ returned consistent. This distribution is now combined with the current beliefs using Bayes' Theorem and the Total Probability Theorem.

Let F be the set $\{true, false, unknown\}$. Then the current belief distribution, $B_{\text{D},\Delta_P}(x)$, is updated by the evidence $B_{\text{D},\Delta_P}(x;\Delta)$ to the new belief distribution $B^+_{\text{D},\Delta_P}(x)$:

$$B^+_{\text{D},\Delta_P}(x) = \sum_{f_1,f_2 \in F} P(B^+_{\text{D},\Delta_P}(x) \mid B_{\text{D},\Delta_P}(f_1) = 1 \wedge B_{\text{D},\Delta_P}(f_2;\Delta) = 1) \cdot B_{\text{D},\Delta_P}(f_1) \cdot B_{\text{D},\Delta_P}(f_2;\Delta) \quad (6)$$

The probabilities $P(B^+_{\text{D},\Delta_P}(x) \mid B_{\text{D},\Delta_P}(f_1) = 1 \wedge B_{\text{D},\Delta_P}(f_2;\Delta) = 1)$ in Equation 6 can be represented by a conditional probability table as shown in Table 1. The first two columns represent f_1 and f_2 respectively and the last three represent x. The values in Table 1 are chosen such that if the current belief is very certain, as defined by the weight of the *unknown* state, then a new distribution which is very uncertain, will not strongly influence the belief, and vice versa. If the new evidence agrees with our current belief, then this belief is strengthened, and if it does not then it is weakened.

Table 1. Conditional Probability Table used to update $B_{\text{D},\Delta_P}(x)$ given $B_{\text{D},\Delta_P}(x;\Delta)$

$P(B^+_{\text{D},\Delta_P}(x) \mid B_{\text{D},\Delta_P}(f_1) = 1 \wedge B_{\text{D},\Delta_P}(f_2;\Delta) = 1)$

f_1	f_2	x: $True$	$False$	$Unknown$
True	True	1.0	0.0	0.0
True	False	0.5	0.5	0.0
True	Unknown	1.0	0.0	0.0
False	True	0.5	0.5	0.0
False	False	0.0	1.0	0.0
False	Unknown	0.0	1.0	0.0
Unknown	True	1.0	0.0	0.0
Unknown	False	0.0	1.0	0.0
Unknown	Unknown	0.0	0.0	1.0

2.1.2 *Update belief in inverse conflict*

To update the beliefs $B_{\text{IC},\Delta}(x)$, much the same procedure is followed as in the case where the system is consistent, only now the evidence suggests that the considered sets are inverse conflicts rather than diagnoses. As before, the first step is to locate the set Δ_B, but now it is the base set of $(A_\Delta)^c$ ($\Delta_B = \{7, 8, 9\}$ in Figure 2). $(\Delta_B)^c$ is the largest set of components such that all of $(SD_{\Delta_B})^c$ was used to compute μ_Δ and we again assume that $|(SD_{\Delta_B})^c| \gg |A_\Delta \setminus (SD_{\Delta_B})^c|$. The evidence provided by μ_Δ suggests that some of the constraints in $(SD_{\Delta_B})^c$ have been violated. Since adding constraints will not take away the fact that some of these have not been met, every superset of $(SD_{\Delta_B})^c$ also contains broken constraints indicating that every subset, Δ_C, of Δ_B is an inverse conflict. As before, the probability that the set Δ_C is an inverse conflict is:

$$P(\Delta_C \text{ is an inverse conflict} \mid A_\Delta \wedge \mu_\Delta = 0) = \frac{|A_\Delta \bigcap (SD_{\Delta_C})^c|}{|A_\Delta|}$$

We assume a mapping $\mathcal{PIC}(\mu_\Delta) \in [0, 1]$, defined by the modeler, which represents the probability that the real artifact is inconsistent given μ_Δ. This mapping is then used to compute a distribution, $B_{\text{IC},\Delta_C}(x;\Delta)$, over the states $\{true, false, unknown\}$ which represents the belief that the set Δ_C is an inverse conflict given only the information from calling $\texttt{TP}(\cdot)$ on Δ.

$$\begin{aligned} &B_{\text{IC},\Delta_C}(true;\Delta) \\ &\quad = P(\Delta_C \text{ is an inverse conflict} \mid A_\Delta \wedge \mu_\Delta = 0) \cdot \mathcal{PIC}(\mu_\Delta) \\ &B_{\text{IC},\Delta_C}(false;\Delta) \\ &\quad = (1 - P(\Delta_C \text{ is an inverse conflict} \mid A_\Delta \wedge \mu_\Delta = 0)) \cdot \mathcal{PIC}(\mu_\Delta) \\ &B_{\text{IC},\Delta_C}(unknown;\Delta) \\ &\quad = 1 - \mathcal{PIC}(\mu_\Delta) \end{aligned}$$

This belief distribution is incorporated into our current belief $B_{\text{IC},\Delta_C}(x)$ in the same manner as discussed in the previous section. The total probability theorem is again used as in Equation 6 to compute the new belief distribution $B^+_{\text{IC},\Delta_C}(x)$ from the old one $B_{\text{IC},\Delta_C}(x)$ and the new evidence $B_{\text{IC},\Delta_C}(x;\Delta)$ using the conditional probabilities in Table 1.

The new evidence provided by the call to $\texttt{TP}((SD_\Delta)^c, OBS)$ has now been incorporated into the belief distributions $B_{\text{IC},\Delta}(x)$ and $B_{\text{D},\Delta}(x)$ for all subsets Δ of $COMPS$. The next section looks at how to use these belief distributions to choose the next component to pass to $\texttt{TP}(\cdot)$.

2.2 Next best set

The order in which the subsets of $COMPS$ are tested is crucial to the speed at which the algorithm will find the diagnoses. There are, however, several choices which will produce varying results and so the choice depends largely on knowledge of the system. The following properties can be taken into account when developing a heuristic search strategy:

- Failure rates: choose sets of components with a history of failure
- Expected knowledge gain: choose sets of components which are expected to reduce the unknown portions of the belief distributions the most. (i.e. $B_{\text{D},\Delta}(unknown)$ and $B_{\text{IC},\Delta}(unknown)$). See [5] for a derivation.
- Current belief: choose the supersets and subsets of the set currently most likely to be a minimal diagnosis to isolate a single diagnosis as quickly as possible.
- Principle of Parsimony: choose the sets with the fewest components as they are more likely to be diagnoses.
- Execution time: choose the sets with the most components, as $\texttt{TP}(\cdot)$ will likely take less time to evaluate systems with fewer constraints.

2.3 Stop conditions

The certainties in the potential diagnoses returned by the PGDE increase monotonically with each iteration [5]. Thus, the maximum certainties are achieved when all subsets of $P(COMPS)$ have been passed to $\texttt{TP}(\cdot)$ for testing. Since this is likely to take too long, a decision needs to be made about when to stop. As it is when choosing a search algorithm, this decision is mostly heuristic and entirely up to the modeler. Some examples of criteria are listed here:

- A time limit has been reached
- The sum of all of the subsets of $P(COMPS)$'s knowledge has risen above some limit
- The knowledge gained per call to $\texttt{TP}(\cdot)$ has fallen below some level

- A percentage of the subsets of $COMPS$ have been tested
- At least one minimal diagnosis has been found with some minimum certainty

2.4 Most likely minimal diagnoses

A minimal diagnosis is a diagnosis such that no proper subset of it is also a diagnosis. They are of interest as the Principle of Parsimony [7] states that the diagnoses with the fewest components are the most likely. The minimal diagnoses will have the properties that all of their supersets will be diagnoses and all of their proper subsets will be inverse conflicts. The goal is to determine which sets are most likely to have these properties given the belief distributions $B_{\text{IC},\Delta}(x)$ and $B_{\text{D},\Delta}(x)$.

2.4.1 Combining $B_{D}(x)$ and $B_{IC}(x)$

The two belief distributions $B_{\text{D}}(x)$ and $B_{\text{IC}}(x)$ have been kept separate, as they represent different types of information. In order to compute the most likely minimal diagnoses, all of the information needs to be taken into account and as a result they need to be combined. This is done using the conditional probability table shown as Table 2 to compute the combined belief distribution $D(x)$. $D_\Delta(true)$ represents the probability that Δ is a diagnosis, while $D_\Delta(false)$ represents the probability that it is not. Note that this is different from $B_{\text{D},\Delta}(false)$ as $B_{\text{D},\Delta}(false)$ represents the belief that the evidence does *not* show that Δ is a diagnosis, whereas $D_\Delta(false)$ represents the belief that the evidence *does* show that Δ is *not* a diagnosis. $D_\Delta(unknown)$, represents the belief that we don't know what the evidence shows. The values in Table 2 are chosen such that if $B_{\text{D},\Delta}(x)$ and $B_{\text{IC},\Delta}(x)$ agree that Δ is a diagnosis and not a inverse conflict then $D_\Delta(true) = 1$. However, if they do not agree, then we are confused about what the evidence has shown and $D_\Delta(unknown) = 1$. If neither $B_{\text{D},\Delta}(x)$ nor $B_{\text{IC},\Delta}(x)$ have any information then $D_\Delta(unknown) = 1$.

Table 2. Conditional Probability Table used to combine $B_{\text{D}}(x)$ and $B_{\text{IC}}(x)$ into $D(x)$

$P(D_\Delta(x) \mid B_{\text{D},\Delta}(f_1) = 1 \wedge B_{\text{IC},\Delta}(f_2) = 1)$

f_1	f_2	x $True$	$False$	$Unknown$
True	True	0.0	0.0	1.0
True	False	1.0	0.0	0.0
True	Unknown	1.0	0.0	0.0
False	True	0.0	1.0	0.0
False	False	0.0	0.0	1.0
False	Unknown	0.0	0.0	1.0
Unknown	True	0.0	1.0	0.0
Unknown	False	0.0	0.0	1.0
Unknown	Unknown	0.0	0.0	1.0

2.4.2 Finding the minimal diagnoses

Definition 7 below, defines a distribution $D_{M_\Delta}(x)$ for each $\Delta \in P(COMPS)$ which represents the belief that the set Δ has the properties of a minimal diagnosis.

Definition 7 *Let* $\Delta \in P(COMPS)$.
Let $\Delta_{C_i} \subset \Delta, i = 1, \ldots, m,\ \forall i \neq j\ \Delta_{C_i} \neq \Delta_{C_j}$
Let $\Delta_{P_i} \supset \Delta, i = 1, \ldots, n,\ \forall i \neq j\ \Delta_{P_i} \neq \Delta_{P_j}$.
Define the distribution $\neg D(x)$ *such that:*

$$\begin{aligned} \neg D(true) &= D(false) \\ \neg D(false) &= D(true) \\ \neg D(unknown) &= D(unknown) \end{aligned}$$

Define the operator $\odot$ *such that* $A \odot B$ *equals the result of combining* A *and* B *using the conditional probability table 3, then:*

$$\begin{aligned} D_{M_\Delta}(x) = \; & D_\Delta(x) \\ & \odot D_{\Delta_{P_1}}(x) \odot \ldots \odot D_{\Delta_{P_n}}(x) \\ & \odot \neg D_{\Delta_{C_1}}(x) \odot \ldots \odot \neg D_{\Delta_{C_m}}(x) \end{aligned}$$

Table 3. Conditional Probability Table used to compute $C = A \odot B$

$P(C(x) \mid A(f_1) = 1 \wedge B(f_2) = 1)$

f_1	f_2	x $True$	$False$	$Unknown$
True	True	1.0	0.0	0.0
True	False	0.0	1.0	0.0
True	Unknown	1.0	0.0	0.0
False	True	0.0	1.0	0.0
False	False	0.0	1.0	0.0
False	Unknown	0.0	1.0	0.0
Unknown	True	0.0	0.0	1.0
Unknown	False	0.0	0.0	1.0
Unknown	Unknown	0.0	0.0	1.0

The result is that $D_{M_\Delta}(x)$ is true for sets which have all properties that a minimal diagnosis should have and false or unknown for all other sets. Because $D_M(x)$ is a continuous distribution over the states $\{true, false, unknown\}$, a function is needed which allows the possible diagnoses to be returned to the diagnostician in order from most likely to least, along with a measure of the algorithm's certainty in the result. The following sorting function is suggested as a good balance between certainty in the result and the belief that the set is a minimal diagnosis:

$$D_{M_\Delta}(true) \cdot \big(1 - D_{M_\Delta}(unknown)\big) \tag{8}$$

Minimal diagnoses can now be returned to the diagnostician in order from the one with the largest value for Equation 8 to the smallest. The probability that a set is a minimal diagnosis is equal to $D_{M_\Delta}(true)/(1 - D_{M_\Delta}(unknown))$ and the certainty in the result defined by $1 - D_{M_\Delta}(unknown)$.

2.5 Complexity considerations

Calling $\texttt{TP}(\cdot)$ on every subset of $COMPS$ is an exponential undertaking. If the PGDE is run so that the maximum certainty is achieved in the result, every subset of $COMPS$ would need to be tested and the algorithm would indeed be exponential in time. However, a trade-off can be made between certainty and execution time by using some of the criteria listed in Section 2.3.

Maintaining the distributions $B_{\text{D}}(x)$ and $B_{\text{IC}}(x)$ is exponential in space if the entire set $P(COMPS)$ is considered. However, for example, we assume that the likelihood of 40 components failing simultaneously in a system of 50 components is negligible. Therefore, the algorithm does not require that the distributions $B_{\text{D}}(x)$ and $B_{\text{IC}}(x)$

cover all of $P(COMPS)$, but only up to the level where a reasonable number of simultaneous faults are considered.

As seen in Figure 1 there are four steps to the algorithm which are performed in an iterative fashion: choose next set, call $\texttt{TP}(\cdot)$, interpret the results and update the beliefs $B_{\text{D}}(x)$ and $B_{\text{IC}}(x)$. This algorithm is primarily intended for the diagnosis of complex dynamic systems for which $\texttt{TP}(\cdot)$ will require a period of simulation in order to test for consistency and so it is assumed that this call will take a significant period of time. Computing the next set to test can be a function of $P(COMPS)$, but it is assumed that the $\texttt{TP}(\cdot)$ will take the majority of the time. Both the interpretation of the results and the updating of the belief states involve only the supersets and/or subsets of the set under test, which is a relatively small number when compared to the size of $P(COMPS)$. The final two steps of the algorithm do involve the entire set $P(COMPS)$, but as they are not part of the iterative procedure, their effect on the speed of the algorithm is not significant.

3 DIAGNOSIS OF A HYDRAULIC CIRCUIT

Figure 4 shows a schematic for a single degree of freedom hydraulic manipulator used to test the algorithm presented in this paper. The model is made of eight components as seen in Figure 3: the head-side port of the main valve, the rod-side port of the main valve, the cylinder, the manipulator, the rod-side anti-cavitation valve, the head-side anti-cavitation valve, the exit filter and the check valve. The behavior of the components is described by sets of hybrid dynamic equations which can be found in [6] and [5].

The function $\texttt{TP}((SD_{\Delta})^{c}, OBS)$ was implemented using a modified version of Hybrid Concurrent Constraint programming, or hcc [3]. The set of hybrid dynamic equations $(SD_{\Delta})^{c}$ is passed to the modified hcc, along with OBS which are the time sequences of the sensor values. The system made of $(SD_{\Delta})^{c}$ and OBS will likely be over-constrained and the resulting simulation will contain several discrepancies between measured and simulated values. These residuals (simulated outputs less measured) will also be time sequences which can be compared to a set of residuals recorded during normal operation to generate a consistency measure, μ_{Δ}. During the experiments, the system was setup in a position control loop with a sinusoidal input signal at a frequency of 0.25Hz. A period of six seconds is recorded, encompassing a single extension and retraction of the manipulator arm. Six experiments were run, each with the arm under a different failure condition which is common in a system such as this [6, 9]. The failures were caused by manual adjustment of the three valves and one friction plate shown in Figure 4. The faults are assumed to be permanent and to have occurred before the measurements are taken. At each iteration the set to be passed to $\texttt{TP}(\cdot)$ is selected to maximize the expected decrease in $U = \sum_{\Delta \in P(COMPS)} B_{\text{IC},\Delta}(unknown) + B_{\text{D},\Delta}(unknown)$ and the algorithm is stopped when the change in U is less than 1% for more than 10 iterations.

The six failures and the results of fault isolation using the PGDE are as follows. On average, 99.90% of the time taken is spent in simulation during the calls to $\texttt{TP}(\cdot)$, while only 0.10% is required for the PGDE calculations. For details refer to [5].

- Leak in the hose connecting the valve to the head-side of the cylinder.

  This failure was correctly isolated in all 10 sample runs taking an average of 54.5 seconds.
- Leak in the hose connecting the valve to the rod-side of the cylinder.

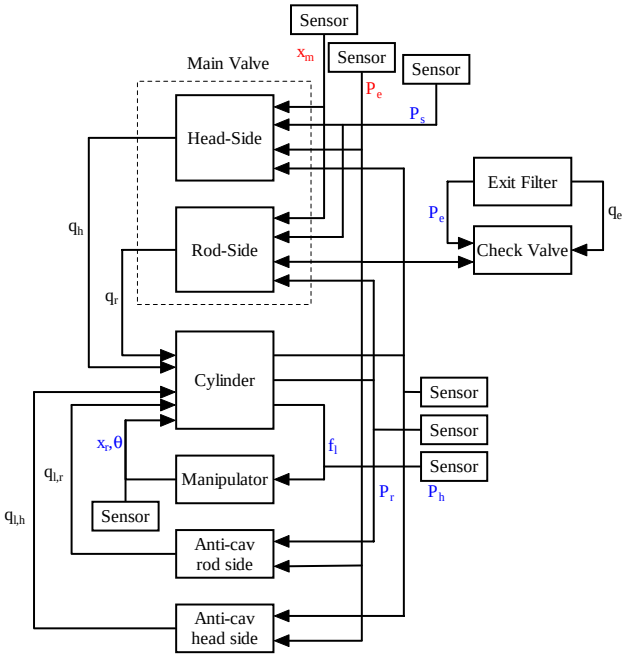


Figure 3. Component model of the hydraulic test bench

This failure was correctly isolated in all 10 sample runs taking an average of 53.1 seconds.
- Partially clogged return filter.

  For two of the five tests run, the filter was returned as the most likely diagnosis, with the rod-side port of the main valve and the rod-side anti-cavitation valves together forming a close second. In the remaining three tests the filter was not returned as a diagnosis by itself, but five diagnoses containing the filter and another component were returned as all being very likely. The average calculation time was 167 seconds.
- Increased friction in manipulator bearing.

  For two of the five tests run, the manipulator was returned as the only likely diagnosis with very high certainty (96%, 100%). In two more of the tests it was returned as one-half of a double fault and in the fifth test the algorithm did not get the correct solution. These calculations took on average 82 seconds to complete.
- Leaks in both hoses connecting the valve to the cylinder.

  In all five tests the four double faults: {rod-side anti-cavitation valve, head-side anti-cavitation valve}, {rod-side anti-cavitation valve, head-side port}, {head-side anti-cavitation valve, rod-side port} and {head-side port, rod-side port} were returned as being equally likely with a high degree of certainty ($\sim 85\%$). For this situation, these are the correct diagnoses as one component on the rod-side and one on the head-side that can account for the leaks is needed to explain this failure. The average calculation time was 140 seconds.
- Partially clogged return filter and a leak in the head-side hose.

  In all five tests the algorithm returned the head-side anti-cavitation valve or port as the only explanation. The filter causes a much smaller effect on the system and so it is difficult to recognize it as faulty when other components are misbehaving. The average calculation time was 61 seconds.

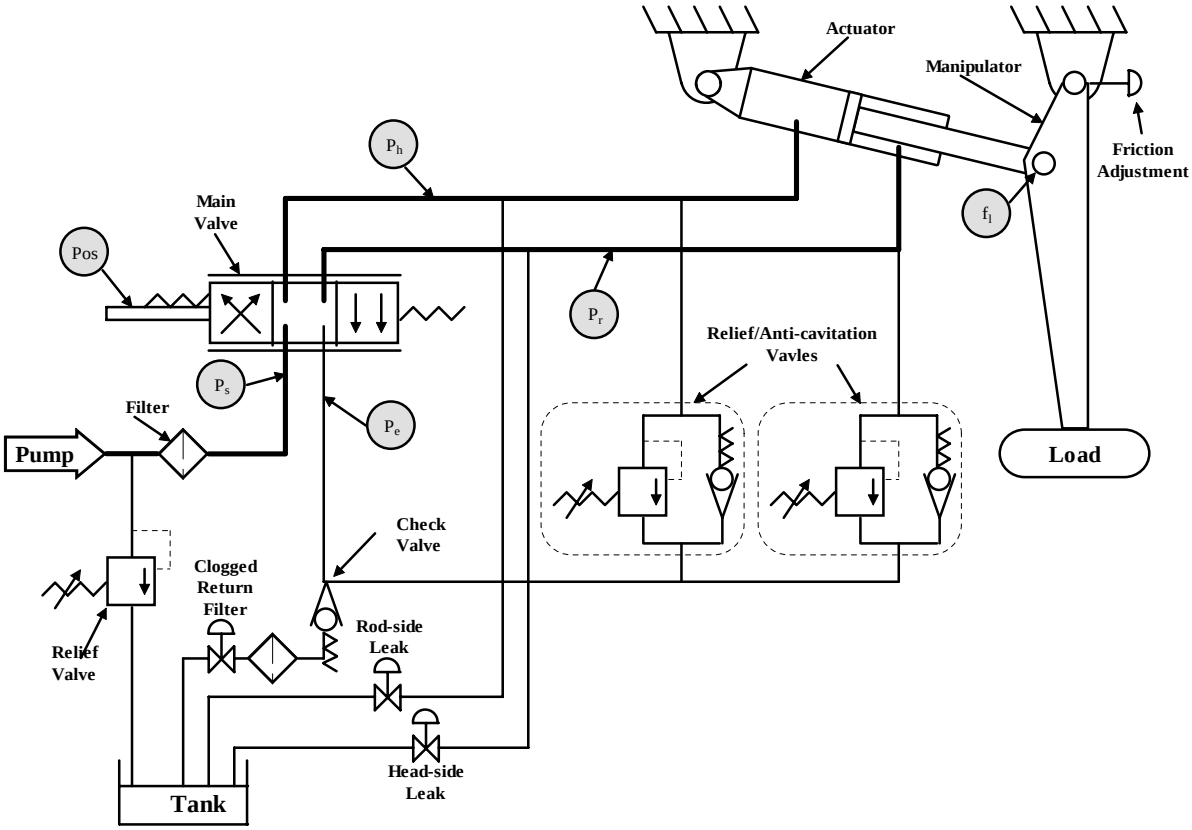


Figure 4. Schematic of experimental test bench

4 CONCLUSIONS

This paper has presented a novel approach to consistency-based diagnosis which allows for the use of any modelling language. The use of continuous distributions representing the belief that each set of components is a diagnosis allows the determination of consistency or inconsistency to be delayed until supporting evidence has been collected and for noise in the simulator, $\mathtt{TP}(\cdot)$, to be handled. The demonstration of this algorithm on a non-trivial physical test bench shows that it can be applied effectively to isolate realistic faults in real artifacts.

ACKNOWLEDGEMENTS

The first author would like to thank the Natural Sciences and Engineering Research Council (NSERC) of Canada for partial funding of this work.

REFERENCES

[1] B.Pulido, C.Alonso, C.Llamas, and F.Acebes, 'Consistency-based diagnosis using possible conflicts', in *Proc. of the twelfth Workshop on Principles of Diagnosis*, (2001).

[2] B.Pulido, F.Acebes, and C.Alonso, 'Exploiting knowledge about structure and behaviour in consistency-based diagnosis with fault modes in dynamic systems', in *Proc. of the Ninth Intl. Workshop on Principles of Diagnosis*, (1998).

[3] V. Gupta, R. Jagadeesan, V. Saraswat, and D. Bobrow, 'Computing with continuous change', *Science of Computer Programming*, (1997).

[4] W.C. Hamscher and R. Davis, 'Diagnosing circuits with state: An inherently underconstrained problem', in *Proceedings of the 4^{th} National Conference on Artificial Intelligence*, (1984).

[5] Colin N. Jones, *Consistency-Based Fault Isolation for Hybrid Dynamic Models*, Master's thesis, The University of British Columbia, August 2001.

[6] Masoud Khoshzaban-Zavarehi, *On-Line Condition Monitoring and Fault Diagnosis in Hydraulic System Components using Parameter Estimation and Pattern Classification*, Ph.D. dissertation, University of British Columbia, 1997.

[7] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–96, (1987).

[8] S.Narasimhan, F.Zhao, G.Biswas, and E.Hung, 'Fault isolation in hybrid systems combining model based diagnosis and signal processing', in *Proc. of the Intl. Workshop on Hybrid Systems*, (2000).

[9] Xiaodan Sun, *Real-Time Performance Monitoring and Fault Diagnosis of Hydraulic Manipulators*, Master's thesis, The University of British Columbia, March 1995.

[10] Andrew Watkins, *Consistency-Based Diagnosis Using Dynamic Models*, Master's thesis, The University of British Columbia, 1999.

Merging Indiscriminable Diagnoses: an Approach Based on Automatic Domains Abstraction

Pietro Torasso, Gianluca Torta
Dipartimento di Informatica
Università di Torino
Torino (Italy)
e-mail: {torasso,torta}@di.unito.it

Abstract. The paper presents an approach suitable for on-line diagnosis, which aims at automatically abstracting the domains of discrete variables in the model (i.e. behavioral modes of system components) in order to keep only those distinctions that are relevant given the available observations and their granularity.
In particular the paper describes an algorithm which identifies indistinguishable behavioral modes by taking into account specific classes of available observations and derives an abstract model where such modes are merged and the domain model is revised accordingly.
By considering increasingly restricted classes of available observations (and/or granularity of observations), a set of abstract models can be derived that can be exploited through model selection each time a new diagnostic problem has to be solved.
The approach has been tested within the framework of a diagnostic agent for a space robotic arm, and experimental results showing the reduction in the number of diagnoses are reported.

1 Introduction

Model based diagnosis has been applied successfully to automatic on-board diagnosis problems in a variety of domains, including automotive and space missions ([1], [10]).

While many problems are common to off-line and on-line diagnosis, the latter presents some peculiar challenges, the most apparent of which concerns the tough constraints on computational resources and time ([3]).

Another difficult problem both on-line and off-line diagnosis have to deal with is the potentially large number of alternative diagnoses returned by a diagnostic system when a specific problem has to be solved.
One classical way of addressing this problem consists in defining preference criteria among diagnoses, usually based on some form of minimality (see e.g. [6]) or probability, so that a number of admissible diagnoses can be discarded because of their implausibility .
We can also approach the problem not at diagnosis time, but at earlier time (i.e. during system design and modeling): there exist guidelines for creating models suitable for troubleshooting (see e.g. [8]) as well as methods for suggesting the placement of enough sensors in the system to guarantee that only one or a few admissible diagnoses will be returned in each situation (see for example [15]); sensors failures can be handled by an adequate level of redundancy.
Finally, the encoding of large sets of diagnoses in a compact way can at least alleviate the explosion of time and space required to compute and handle such large sets (see [11]).

Unfortunately, all these approaches only provide a partial solution; while preference criteria, cleverly written models and compact encoding do not guarantee that the reduced set of diagnoses is small enough in all situations, exhaustive sensor placement may be too expensive or just impossible because the device design is already frozen.

In off-line diagnosis, there's an additional possibility: when the number of diagnoses returned on the basis of available observations is too high, further discriminant measures can be automatically suggested and manually taken until a satisfactory level of discrimination is reached. Effective techniques based on information theory and probability have been devised to support this process (e.g. [7]). However, for on-board diagnosis, this approach is inadequate since in most cases the only available measures are provided by sensors and taking further measures manually is out of question.

In this paper we present an approach suitable for on-board diagnosis, which aims at automatically abstracting the domains of discrete variables in the model (i.e. behavioral modes of system components) in order to keep only those distinctions that are relevant given the available observations and their granularity. As we shall see, this can significantly reduce the number of returned diagnoses.

The paper is structured as follows. In section 2 we introduce some definitions, in particular the notion of indistinguishability among the behavioral modes of a component. In section 3 we present an algorithm which identifies indistinguishable behavioral modes by taking into account specific classes of available observations and derives an abstract model where such modes are merged and the domain model is revised accordingly. Section 4 discusses some ways the algorithm can be used effectively in diagnostic problem solving.

In section 5 we report experimental results obtained by implementing and running the algorithm on the model for a space robotic arm. Finally, in section 6 we briefly review other approaches in the literature and underline similarities and differences with respect to our own.

2 Basic Definitions

First, we define a system structure description (SSD) by slightly modifying the definition in [5]:

Definition 2.1 *A Structured System Description (SSD) is a tuple* $\langle V, \mathcal{G}, DT \rangle$ *where:*

- V *is a set of variables whose domains* $DOM(v), v \in V$ *are discrete and finite. Moreover, variables in* V *are partitioned in the following sorts: CXT (inputs), COMPS (components), STATES (endogenous variables), OBS (observables)* [1]
- DT *(Domain Theory) is a set of Horn clauses defined over* V *representing the behavior of the system (both normal and faulty). Note that the clauses are constructed in such a way that the roles associated with variables belonging to different sorts are respected:* CXT *and* $COMPS$ *variables will always appear in the body of clauses;* OBS *variables will always appear as heads of clauses;* $STATES$ *variables can appear in both*
- $\mathcal{G}$ *(System Structure) is a DAG whose nodes are in* V *representing the structure of the system. The graph can be directly computed from* DT*, being just a useful way for making explicit the structural properties "hidden" in* DT *clauses: whenever a formula* $N_1(bm_1) \wedge \ldots \wedge N_k(bm_k) \Rightarrow M(bm_l)$ *appears in* DT*, nodes* N_1 *through* N_k *are parents of* M *in the graph*

Since the system structure graph $\mathcal{G}$ is a DAG, a partial precedence relation holds between connected nodes in the graph:

Definition 2.2 *We denote with* $\succ$ *the usual precedence partial order relation over nodes in DAG* $\mathcal{G}$*, i.e.:* $N \succ M$ *if there exists a directed path from* N *to* M*.*

Given an SSD we can define specific diagnostic problems over it:

Definition 2.3 *A diagnostic problem is a tuple* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$ *where SSD is the System Structured Description,* $\mathbf{OBS}'$ *is an instantiation of* $OBS' \subseteq OBS$ *and* $\mathbf{CXT}$ *is a complete instantiation of* CXT

We are now ready to give our definition of diagnosis, which is a fully abductive characterization [2] (see [4]):

Definition 2.4 *Given a diagnostic problem* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$ *an assignment* $H = \{c_1(bm_1), \ldots, c_n(bm_n)\}$ *of a behavioral mode to each component* $c_i \in COMPS$ *is a diagnosis for DP if and only if:*

$$\forall m(x) \in \mathbf{OBS}' \quad DT \cup \mathbf{CXT} \cup H \vdash m(x)$$

and

$$\forall m(x) \in \mathbf{OBS}' \quad DT \cup \mathbf{CXT} \cup H \nvdash m(y) \text{ for } y \neq x$$

[1] We assume that observables never influence other variables. This is not restrictive: each observable parameter which influences other variables is modeled as an endogenous variable (i.e. it belongs to $STATES$) with an associated observable in OBS

[2] Note however that our approach does not depend on the definition of diagnosis being abductive vs consistency-based

Since in our definition, $\mathbf{OBS}'$ is (in general) a partial instantiation of OBS, we can introduce the notion of diagnoses that can't be discriminated given $\mathbf{OBS}'$ but that may be discriminated if more observables were available:

Definition 2.5 *Given a diagnostic problem* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$*, let us suppose that* $H1$ *and* $H2$ *are two diagnoses for DP.* $H1$ *and* $H2$ *are discriminable if and only if* $\exists\ m \mid m \in (OBS - OBS')$ *such that*
$DT \cup \mathbf{CXT} \cup H1 \vdash m(a)$
$DT \cup \mathbf{CXT} \cup H2 \vdash m(b)$
$m(a) \neq m(b)$

Diagnoses are complete instantiations of variables in sort $COMPS$. We now turn into considering two such assignments $A1$ and $A2$ and compute the projections [3] of their transitive closures [4] over OBS ($\mathbf{OBS1} = project_{OBS}(tclosure(A1))$ and $\mathbf{OBS2} = project_{OBS}(tclosure(A2))$ respectively), given a fixed context $\mathbf{CXT}$.
If $\mathbf{OBS1} = \mathbf{OBS2}$ then $A1$ and $A2$ are indiscriminable diagnoses for diagnostic problem $\langle SDD, \mathbf{OBS}, \mathbf{CXT} \rangle$ where $\mathbf{OBS} = \mathbf{OBS1}$ (and $= \mathbf{OBS2}$). An interesting relation between $A1$ and $A2$ holds when this situation happens under *any* fixed context $\mathbf{CXT}$:

Definition 2.6 *Let* $A1$ *and* $A2$ *be two complete instantiations of* $COMPS$*; if, given any context* $\mathbf{CXT}$*,* $project_{OBS}(tclosure(A1)) = project_{OBS}(tclosure(A2))$*, then we say that* $A1$ *and* $A2$ *are indiscriminable.*

In the above definition we have considered the case where all OBS are available. Let's now consider the case (as it is usual in on-board diagnosis) when we can identify subsets of OBS that may be the only available manifestations (e.g. only sensorized manifestations may be available on-board, with no possibility to perform further measurements).
Let $\{CL_k\}$ denote such identified interesting subsets (not necessarily all disjoint); we can now refine definition 2.6 as follows:

Definition 2.7 *Two assignments* $A1$ *and* $A2$ *are* CL_k*-indiscriminable iff they are indiscriminable by considering* OBS *restricted to* CL_k*, i.e.* $\forall \mathbf{CXT}\ project_{CL_k}(tclosure(A1)) = project_{CL_k}(tclosure(A2))$

Given the above definitions, we are now ready to characterize formally two behavioral modes (i.e. values from the domain of a component variable c_i) that may be safely collapsed together without loosing any discriminability power of the model:

Definition 2.8 *Let* bm_r *and* bm_s *be two behavioral modes of component variable* c_i*; if for any two assignments* $A1 = (\alpha 1 \wedge c_i(bm_r) \wedge \alpha 2)$ *and* $A2 = (\alpha 1 \wedge c_i(bm_s) \wedge \alpha 2)$ *such that they differ only in the mode associated to* c_i*,* $A1$ *and* $A2$ *are* (CL_k*-)indiscriminable, then we say that* bm_r *and* bm_s *are* (CL_k*-)indistinguishable.*

[3] A projection of a set of instantiated variables $\mathbf{I}$ over a set of variables W ($project_W(\mathbf{I})$) is just the subset of $\mathbf{I}$ that mentions variables in W

[4] The transitive closure of Ai (tclosure(Ai)) is the set of $m(x)$ s.t. $DT \cup \mathbf{CXT} \cup Ai \vdash m(x)$

3 Automatic Domain Abstraction

3.1 The Algorithm

In this section we present an algorithm which identifies indistinguishable modes in a given model (that we will refer to as the *detailed* model), and generates a simplified model (that we will call *abstract*) where mutually indistinguishable modes are merged in new modes. The algorithm assumes that the model is defined as in definition 2.1 and further assumes that in the system structure graph $\mathcal{G}$ at most one directed path exists between any two nodes.

The top level function `Abstract()` is sketched as pseudo-code in figure 1 while other relevant functions called by `Abstract()` are showed in figure 2.

Parameter $CL_k \subseteq$ `Obs` of `Abstract()` contains the list of available manifestations, while Π_{CL_k} associates to each $M \in CL_k$ its granularity in the form of a partition Π_M over $DOM(M)$.
Manifestations that aren't available at all do not belong to CL_k. If M is available at a certain level of granularity, Π_M will contain as many classes as the distinguishable values for M, and each class will contain all the $v \in DOM(M)$ that can't be distinguished at the available level of granularity. As a special case, if M is available at its maximum granularity, Π_M will contain a separate class for each $v \in DOM(M)$.

The first few instructions of `Abstract()` perform an initial abstraction of the model based on Π_{CL_k}: indistinguishable values for each manifestation M (i.e. those that belong to the same class in Π_M) are substituted in `DT` by a new "abstract" value representing the whole class.

The call to `TopologicalSort()` returns a list containing variables in `Comps` $\cup$ `States` such that if two variables N, M satisfy relation 2.2 (i.e. $N \succ M$) we guarantee that $position(N) > position(M)$. In particular, we start a visit of the system structure graph $\mathcal{G}$ at the available observation nodes and proceed backwards by visiting a node only if all its immediate successors have already been visited.
Note that, by starting the visit at the available manifestations only (i.e. CL_k), some of the `Comps` and `States` may not be reached at all; these nodes, that are connected only to unavailable manifestations, are stored in a `TrivialNodes` list (see below).

The main loop in `Abstract()`, for each variable N in the list, first computes the conditions under which the variable influences its immediate successors modes (this is recorded in an associative memory `InfluencesMatrix[]`); then, by using `InfluencesMatrix[]` it computes the partition of all the modes of the variable in equivalence classes determined by the indistinguishability relation (`FindIndistinguishableModes()`); finally it replaces the occurences of the modes in the `DT` clauses with newly introduced "class representative" modes (`MergeModes()`).
If the call to `FindIndistinguishableModes()` produced a trivial partition for N (i.e. only one class coinciding with $DOM(N)$) then N itself is added to the list of trivial nodes.

When `Abstract()` terminates, `TrivialNodes` contains the components and states whose behavioral modes are all equivalent in influencing relevant manifestations (i.e. $M \in CL_k$). These nodes, together with unavailable manifestations (i.e. $M \in$ `Obs`$\backslash CL_k$) are obviously redundant for the diagnostic task and the caller of `Abstract()` may decide to completely remove them from the model.

Let's now describe into some more detail the functions called by `Abstract()` (figure 2).
Function `FindInfluences()` considers how each mode bm_r of variable N under consideration can cause mode bm_s of immediate successor variable M. The condition under which $N(bm_r)$ causes $M(bm_s)$ is clearly the disjunction of conjunctions of the form $\alpha = \alpha_1 \wedge \alpha_2$ where α_1 and α_2 occur in a formula $\alpha_1 \wedge N(bm_r) \wedge \alpha_2 \Rightarrow M(bm_s)$.
Function `FindIndistinguishableModes()` is recursive; at each call it partitions a set of modes into indistinguishability classes based on a single immediate successor node and then calls itself recursively on each of the generated equivalence classes in order to further discriminate by considering the remaining immediate successors.
Note that in the test $(\langle \alpha, \pi \rangle \in \Pi_{cond})$ we are testing propositional formulas for identity; we assume that any two equivalent formulas have been made identical at that point by calls to *normalize()* in `FindInfluences()`. Normalization is not too computationally expensive since the formulas we handle are in DNF and only positive literals can occur.

Function `MergeModes()`, given a partition Π (either an element of Π_{CL_k} or computed by `FindIndistinguishableModes()`), considers the equivalence classes π one at a time. It generates a new name ν as a "representative" for the class and then scans the `DT` set of formulas for occurrences of $bm \in \pi$ and replaces them with ν. This process can produce duplicate formulas [5]; by using set notation in the pseudo-code we underline that only one copy of the duplicate formulas has to be added to the new version of `DT`.

3.2 Correctness

In this paragraph we state two properties which imply that the abstraction algorithm behaves as intended.

Property 3.1 *If two behavioral modes are put in the same class π by function* `FindIndistinguishableModes()` *they are CL_k-indistinguishable in the sense of definition 2.8.*

Proof. Given assignments $A1 = \alpha_1 \cup \{N(bm_{r1})\} \cup \alpha_2$ and $A2 = \alpha_1 \cup \{N(bm_{r2})\} \cup \alpha_2$ suppose $DT \cup \mathbf{CXT} \cup A1 \vdash m(x)$ while $DT \cup \mathbf{CXT} \cup A2 \not\vdash m(x)$ for some $m \in CL_k$. Clearly, it can't be $m(x) \in tclosure(\alpha 1 \cup \alpha 2)$ because otherwise $m(x)$ would be derivable from $A2$ as well.
Then, the entailment of $m(x)$ by $A1$ must exploit at some point $N(bm_{r1})$ by using a formula $\varphi = (N(bm_{r1}) \wedge \gamma \Rightarrow L)$.
If $L = m(x)$, i.e. the formula directly entails $m(x)$, then, an analogous formula $\varphi' = (N(bm_{r2}) \wedge \gamma' \Rightarrow m(x))$ must exist in DT, with $\gamma \Leftrightarrow \gamma'$ (indeed, two modes are put in the same partition only if they have the same direct effects under the same conditions). Then, $DT \cup \mathbf{CXT} \cup A2 \vdash m(x)$, which is a contradiction.
This result can be extended to the case when $L \neq m(x)$ (i.e. the number of steps between the application of formula φ and the conclusion $m(x)$ is greater than 1) with a proof by induction. □

[5] This is not incidental: the value of our abstraction partially lies in the collapse of formulas

```
Function Abstract(V = ⟨ Cxt, Comps, States, Obs ⟩, G, DT, CL_k, Π_{CL_k})
     ForEach M ∈ CL_k
          DT := MergeModes(M, Π_{CL_k}(M), DT)
     Loop
     Candidates := TopologicalSort(States ∪ Comps, CL_k, G)
     TrivialNodes := States ∪ Comps \ Candidates
     InfluenceMatrix := ∅ × ∅ × ∅
     ForEach (N ∈ Candidates)
          ImmediateSuccessors := {children of N in the system structure graph G} ∩ (Canditates ∪ CL_k)
          InfluenceMatrix := InfluenceMatrix ∪ FindInfluences(N, ImmediateSuccessors)
          Π := FindIndistinguishableModes(N, modes(N), ImmediateSuccessors, InfluenceMatrix)
          If (Π = {DOM(N)}) Then TrivialNodes := TrivialNodes ∪ {N}
          DT := MergeModes(N, Π, DT)
     Loop
     Return
EndFunction
```

Figure 1. Sketch of the `Abstract()` function

The following property is intended to demonstrate the correspondence of a diagnosis at the abstract level to a set of diagnoses at the detailed level.

Property 3.2 *Let* $DP_d = \langle SSD_d, \mathbf{OBS}', \mathbf{CXT} \rangle$ *be a diagnostic problem and* $DP_a = \langle SSD_a, \mathbf{OBS}', \mathbf{CXT} \rangle$ *the corresponding problem at the abstract level. Then,* $D_a = \{c_1(\nu_1), \ldots, c_n(\nu_n)\}$ *where* ν_i *is a new behavioral mode introduced in place of set* $\{bm_{i1}, \ldots, bm_{ik_i}\}$ *of indistinguishable behavioral modes is a diagnosis for* DP_a *iff all the elements in the set:*
$\{\{c_1(bm_{1j_1}), \ldots, c_n(bm_{nj_n})\}, j_i = 1 \ldots k_i\}$
are diagnoses for DP_d.

Proof. Our proof is subdivided in 3 steps: first, we prove that for any two diagnoses at the detailed level D_{d1} and D_{d2}, $project_{OBS'}(tclosure(D_{d1})) = project_{OBS'}(tclosure(D_{d2}))$, where $OBS' \subseteq OBS$ represents the available manifestations (parameter `Obs` of function `Abstract()`). Then, we prove that for any detailed diagnosis D_d, $project_{OBS'}(tclosure(D_d)) = project_{OBS'}(tclosure(D_a))$. Finally, we exploit this result to prove the theorem thesis.
In the following, $project_{OBS'}(tclosure(.))$ has been abbreviated in $tc_{OBS'}(.)$.

For step 1, we proceed by induction on the number of components which are assigned different behavioral modes in assignments D_{d1} and D_{d2}. The case $n = 1$ (i.e. $D_{d1} = \alpha \cup \{c_i(bm_r)\}$ and $D_{d2} = \alpha \cup \{c_i(bm_s)\}$) follows from the definition of indistinguishability of bm_r and bm_s.
For the inductive step, where $D_{d1} = \alpha 1 \cup \{c_i(bm_r)\}$, $D_{d2} = \alpha 2 \cup \{c_i(bm_s)\}$ and $\alpha 1, \alpha 2$ differ in assignments to n components, we note the following relations hold:
$tc_{OBS'}(\alpha 1 \cup c_i(bm_r)) = tc_{OBS'}(\alpha 1 \cup c_i(bm_s))$
from indistinguishability of bm_r and bm_s, and:
$tc_{OBS'}(\alpha 1 \cup c_i(bm_s)) = tc_{OBS'}(\alpha 2 \cup c_i(bm_s))$
from inductive hypothesis. It then follows that $tc_{OBS'}(\alpha 1 \cup c_i(bm_r)) = tc_{OBS'}(\alpha 2 \cup c_i(bm_s))$.

In order to carry step 2, we note that, since ν_i is substituted by `MergeModes()` wherever a mode bm_{ij_i} of its associated class π appears, the following holds:
$tc_{OBS'}(D_a) = \bigcup_{j_1,\ldots,j_n} tc_{OBS'}(\{c_1(bm_{1j_1}), \ldots, c_n(bm_{nj_n})\})$
But, in step 1, we have proved that all the terms of the union are equal. So, $tc_{OBS'}(D_a)$ is equal to the $tc_{OBS'}$ of any of the D_d.

We use this result for step 3: D_a is a diagnosis with the abstracted model iff $DT \cup \mathbf{CXT} \cup D_a \vdash \mathbf{OBS}'$; but, then, for any D_d the same entailment must hold, thus any D_d is a diagnosis at the detailed level. The converse is analogous. □

3.3 An Example

We end this section by illustrating how the abstraction algorithm works on a very simple SSD. Let the original Domain Theory DT contain the following clauses (figure 3 shows the associated System Structure Graph):

$s1(a) \wedge s2(a) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(a) \Rightarrow m2(x)$
$s1(a) \wedge s2(b) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(b) \Rightarrow m2(x)$
$s1(a) \wedge s2(c) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(c) \Rightarrow m2(z)$
$s1(b) \wedge s2(a) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(a) \Rightarrow m2(y)$
$s1(b) \wedge s2(b) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(b) \Rightarrow m2(y)$
$s1(b) \wedge s2(c) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(c) \Rightarrow m2(z)$

$i1(a) \wedge c1(a) \wedge c2(a) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(a) \Rightarrow s1(b)$
$i1(a) \wedge c1(a) \wedge c2(b) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(b) \Rightarrow s1(b)$
$i1(a) \wedge c1(a) \wedge c2(c) \Rightarrow s1(b)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(c) \Rightarrow s1(a)$
$i1(a) \wedge c1(b) \wedge c2(a) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(a) \Rightarrow s1(b)$
$i1(a) \wedge c1(b) \wedge c2(b) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(b) \Rightarrow s1(b)$
$i1(a) \wedge c1(b) \wedge c2(c) \Rightarrow s1(b)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(c) \Rightarrow s1(a)$

$i2(a) \wedge c3(a) \Rightarrow s2(c)$
$i2(a) \wedge c3(b) \Rightarrow s2(a)$
$i2(a) \wedge c3(c) \Rightarrow s2(b)$
$i2(b) \wedge c3(a) \Rightarrow s2(c)$
$i2(b) \wedge c3(b) \Rightarrow s2(a)$
$i2(b) \wedge c3(c) \Rightarrow s2(b)$

```
Function FindInfluences(N, ImmediateSuccessors)
     NodeInfluenceMatrix := ∅ × ∅ × ∅
     ForEach (bm_r ∈ modes(N), M ∈ ImmediateSuccessors, bm_s ∈ modes(M))
          Formulas := {clauses where N(bm_r) occurs in the body and M(bm_s) occurs in the head}
          α := false
          ForEach ((α_1 ∧ N(bm_r) ∧ α_2 ⇒ M(bm_s)) ∈ Formulas)
               α := α ∨ (α_1 ∧ α_2)
          Loop
          NodeInfluenceMatrix := NodeInfluenceMatrix ∪ {(N(bm_r), M(bm_s), normalize(α))}
     Loop
     Return NodeInfluenceMatrix
EndFunction

Function FindIndistinguishableModes(N, Modes, Nodes, InfluenceMatrix)
     M := first(Nodes)
     Π_cond := ∅
     ForEach (bm_r ∈ Modes)
          α := ⋃_{bm_s} ⟨ InfluenceMatrix(N(bm_r), M(bm_s)), M(bm_s) ⟩
          If (⟨α, π⟩ ∈ Π_cond) Then
               Π_cond := Π_cond − {⟨α, π⟩} ∪ {⟨α, π ∪ {bm_r}⟩}
          Else
               Π_cond := Π_cond ∪ {⟨α, {bm_r}⟩}
          EndIf
     Loop
     Π := ⋃_{⟨α,π⟩∈Π_cond} {π}
     If (tail(Nodes) ≠ ∅)
          ForEach (π ∈ Π)
               Π := Π − π ∪ FindIndistinguishableModes(N, π, tail(Nodes), InfluenceMatrix)
          Loop
     EndIf
     Return Π
EndFunction

Function MergeModes(N, Π, DT)
     DT' := ∅
     ForEach (π ∈ Π)
          ν := GenerateNewModeName(π)
          Formulas := {clauses for which ∃bm_r ∈ π s.t. N(bm_r) appears in the body or head}
          ForEach ((φ = α_1 ∧ N(bm_r) ∧ α_2 ⇒ M(bm_s)) ∈ Formulas)
               DT' := DT' ∪ {(α_1 ∧ N(ν) ∧ α_2 ⇒ M(bm_s))}
          Loop
          ForEach ((φ = α ⇒ N(bm_r)) ∈ Formulas)
               DT' := DT' ∪ {(α ⇒ N(ν))}
          Loop
     Loop
     Return DT'
EndFunction
```

Figure 2. Sketch of the main functions called by `Abstract()`

with $OBS = \{m1, m2\}$, $STATES = \{s1, s2\}$, $COMPS = \{c1, c2, c3\}$ and $CXT = \{i1, i2\}$.
Let the domains of the variables be as follows:
$DOM(m1) = \{x, y\}, DOM(m2) = \{x, y, z\}$
$DOM(s1) = \{a, b\}, DOM(s2) = \{a, b, c\}$
$DOM(c1) = \{a, b\}, DOM(c2) = DOM(c3) = \{a, b, c\}$
$DOM(i1) = \{a, b\}, DOM(i2) = \{a, b\}$
Furthermore, let's assume for simplicity that all the OBS are available at their maximum granularity.
The algorithm starts by trying to merge modes of $s1$. The `InfluenceMatrix()` entries relating $s1$ to $m1$ are:
$\langle s1(a), \{\langle m1(x), s2(a) \vee s2(b) \vee s2(c)\rangle, \langle m1(y), \bot\rangle\}\rangle$
$\langle s1(b), \{\langle m1(x), \bot\rangle, \langle m1(y), s2(a) \vee s2(b) \vee s2(c)\rangle\}\rangle$
it follows that modes a, b of $s1$ can't be merged. It is now $s2$ turn to be considered; the entries relating $s2$ to $m1$ are:
$\langle s2(a), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
$\langle s2(b), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
$\langle s2(c), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
it may seem that modes a, b, c of $s2$ can be merged; however, $s2$ also influences another manifestation, $m2$:
$\langle s2(a), \{\langle m2(x), s1(a)\rangle, \langle m2(y), s1(b)\rangle, \langle m2(z), \bot\rangle\}\rangle$
$\langle s2(b), \{\langle m2(x), s1(a)\rangle, \langle m2(y), s1(b)\rangle, \langle m2(z), \bot\rangle\}\rangle$
$\langle s2(c), \{\langle m2(x), \bot\rangle, \langle m2(y), \bot\rangle, \langle m2(z), s1(a) \vee s1(b)\rangle\}\rangle$
we can thus merge modes a, b in new mode ab, but not mode c.
Having considered all the states, we now turn to the components, starting from $c1$:

$\langle c1(a), \{\langle s1(a), (i1(a) \wedge c2(a)) \vee (i1(a) \wedge c2(b)) \vee (i1(b) \wedge c2(c))\rangle,$
$\langle s1(b), (i1(a) \wedge c2(c)) \vee (i1(b) \wedge c2(a)) \vee (i1(b) \wedge c2(b))\rangle\}\rangle$
$\langle c1(b), \{\langle s1(a), (i1(a) \wedge c2(a)) \vee (i1(a) \wedge c2(b)) \vee (i1(b) \wedge c2(c))\rangle,$
$\langle s1(b), (i1(a) \wedge c2(c)) \vee (i1(b) \wedge c2(a)) \vee (i1(b) \wedge c2(b))\rangle\}\rangle$

modes a, b of $c1$ can then be merged in new mode ab; note that $c1$ goes into the trivial-nodes list, since all its domain has collapsed into a singleton. Similar arguments lead to merging modes a, b of $c2$; however, mode c of $c2$ can't be merged with the other two modes.
Component $c3$ is the only one left to be considered:
$\langle c3(a), \{\langle s2(ab), \bot\rangle, \langle s2(c), i2(a) \vee i2(b)\rangle\}\rangle$
$\langle c3(b), \{\langle s2(ab), i2(a) \vee i2(b)\rangle, \langle s2(c), \bot\rangle\}\rangle$
$\langle c3(c), \{\langle s2(ab), i2(a) \vee i2(b)\rangle, \langle s2(c), \bot\rangle\}\rangle$
we can merge modes b, c into a new node bc. Note that we can merge these modes only because we already unified modes a and b of $s2$; the importance of processing variables in the $\succ$ relation order is now evident.
Note also that we could have considered for abstraction $s2$ before $s1$, or $c2$ before $c1$ or after $c3$ since $s1, s2$ and $c1, c2, c3$ are not tied by the precedence order relation. It is easy to see that in such case the same mergings would have taken place anyway.

The output of the process described above results in a revised domain theory:

$s1(a) \wedge s2(ab) \Rightarrow m1(x)$
$s1(a) \wedge s2(c) \Rightarrow m1(x)$
$s1(b) \wedge s2(ab) \Rightarrow m1(y)$
$s1(b) \wedge s2(c) \Rightarrow m1(y)$

$s1(a) \wedge s2(ab) \Rightarrow m2(x)$
$s1(a) \wedge s2(c) \Rightarrow m2(z)$
$s1(b) \wedge s2(ab) \Rightarrow m2(y)$
$s1(b) \wedge s2(c) \Rightarrow m2(z)$

$i1(a) \wedge c1(ab) \wedge c2(ab) \Rightarrow s1(a)$
$i1(a) \wedge c1(ab) \wedge c2(c) \Rightarrow s1(b)$
$i1(b) \wedge c1(ab) \wedge c2(ab) \Rightarrow s1(b)$

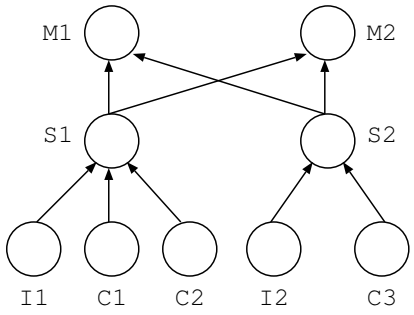


Figure 3. System Structure Graph for the Example Domain Theory

$i1(b) \wedge c1(ab) \wedge c2(c) \Rightarrow s1(a)$

$i2(a) \wedge c3(a) \Rightarrow s2(c)$
$i2(a) \wedge c3(bc) \Rightarrow s2(ab)$
$i2(b) \wedge c3(a) \Rightarrow s2(c)$
$i2(b) \wedge c3(bc) \Rightarrow s2(ab)$

and abstracted domains:
$DOM(m1) = \{x, y\}, DOM(m2) = \{x, y, z\}$
$DOM(s1) = \{a, b\}, DOM(s2) = \{ab, c\}$
$DOM(c1) = \{ab\}, DOM(c2) = \{ab, c\}, DOM(c3) = \{a, bc\}$
$DOM(i1) = \{a, b\}, DOM(i2) = \{a, b\}$

4 Using Abstract Models in On-Board Diagnosis

Having described how the abstraction algorithm works, we now consider how it can be used in real scenarios to practically improve the performance of the diagnostic problem solver.

A first scenario is when the manifestations of the system can be naturally subdivided in classes CL_k (see section 2); one such classes will contain all the manifestations (CL_{all}), another may contain only sensorized manifestations (CL_{sens}), further ones may exclude from CL_{sens} other groups of manifestations that can potentially all become unavailable together in some contexts. Similarly, manifestation granularities (expressed as abstraction functions τ_i) may be identified and associated to classes they apply to.
Equipped with this set of pairs $\langle CL_k, \tau_i\rangle$, we can generate off-line a corresponding set of models M_{ki}; when a specific diagnostic problem is presented to the on-line diagnostic agent, the minimal $\langle CL_k, \tau_i\rangle$ that covers the available observations is selected, and the corresponding model M_{ki} is used to compute diagnoses.

Sometimes, however, classes of manifestations (and their granularity) cannot be conveniently identified a-priori. In such cases we may want to compute an abstract model *on-demand*, given the particular CL_k and τ_i that have been identified as currently available [6].
The system may perform this on-line model synthesis as a lower priority task, asynchronously with the diagnostic tasks;

[6] How this info can be gathered, either manually or automatically, is out of the scope of the present paper

once the ad hoc M_{ki} has been computed it may be reused for many diagnostic problems until some conditions on the available observations or their granularity changes.
Obviously, time overhead is added by the computation of models but in some situations this may well be paid off by the benefits (see below). Moreover, experimental data presented below in section 5 show that such overhead may be in the order of the time needed for solving a few easy diagnostic cases (involving a single fault) or just a difficult one (involving multiple faults); keeping in mind that the abstraction algorithm is only run once whilst many diagnostic problems can exploit such a abstract model, this overhead may be acceptable.

In both the above scenarios, the number of returned diagnoses is reduced by returning diagnoses for the abstract model that correspond to sets of diagnoses for the detailed model that carry essentially the same information, as proved in section 3.2.
Moreover, whenever a diagnosis for the abstract model mentions a "compound mode" (i.e. a new mode introduced in place of a non-singleton set of indistinguishable modes), we explicitly know that the set of modes it represents couldn't be discriminated even in different contexts. Thus, in case further tests involving different contexts are planned, they shouldn't aim at that kind of discrimination.
Both reduced-size and increased informativity of the result should be helpful for the human or automatic supervisor which must interpret it and take action accordingly.

5 Experimental Results

We have implemented the algorithm described above as a module of the diagnostic agent for the space robotic arm SPIDER developed by ASI (Agenzia Spaziale Italiana); for a description of the diagnostic agent please see [12] and [11].
The model of the robotic arm (which obeys definition 2.1) is enough complex to represent an interesting test-bed: it consists of 35 assumables ($COMPS$) with an average 3.43 behavioral modes each, 45 manifestations (OBS) and 1143 formulas [7].

Observations in such a model are explicitly partitioned into two classes: sensorized (CL_{sens}) and non-sensorized (CL_{nosens}). While observations in CL_{sens} can reasonably be assumed to always be available, observations in CL_{nosens} are available through manual measurements that can be carried only during off-line diagnosis.

We have applied the abstraction algorithm to the model by passing CL_{sens} as the available observations (assuming $\tau_{OBS} = identity$, i.e. manifestations available at their maximum granularity) and obtained a simplified model as output. Table 1 shows some relevant static measures on the detailed and abstract models: the number of clauses has been reduced by 18.6%, and the average number of behavioral modes per system component has been reduced by 16.9%. Compilation of the detailed model in the abstract one took 1494msec of CPU time (all results in this section are referred to a Java implementation of both the diagnostic agent and the abstraction algorithm, compiled and run using jdk1.3 on a Sun Sparc Ultra 5 equipped with SunOS 5.8).

[7] The number of formulas is greatly reduced by the use of a noisy-max modeling technique, see [12]

model	clauses	modes avg
detailed	1143	3.43
abstract	930	2.85

Table 1. Comparison between abstract and detailed models

We have then compared the performance of the diagnostic agent when it uses the detailed (original) versus the generated abstract model. Using the simulator for the diagnostic agent, three test sets of 100 diagnostic problems each have been automatically generated; problems in test sets 1, 2 and 3 had 1, 2 and 3 faults injected respectively. Table 2 reports on the reduction of the average number of diagnoses returned. Particularly significant appear the reductions obtained in test set 2 (-43%) and test set 3 (-61.6%).
It should be noted that the diagnostic agent returns only preferred diagnoses (in particular, those that have a minimal number of faults), thus the reported reductions are obtained by compacting "good-quality" diagnoses, not by discarding implausible ones.

model	testset 1	testset 2	testset 3
detailed	5.0 ± 0.6	17.9 ± 3.6	123.3 ± 23.1
abstract	3.7 ± 0.4	10.2 ± 1.9	47.3 ± 8.4

Table 2. Average number of elementary diagnoses obtained with abstract and detailed models (confidence 95%)

Even if our diagnostic agent uses a compact encoding for candidate diagnoses during the search process, thus obtaining an optimized search space size that is not proportional to the number of diagnoses ([11]), the average time employed for solving problems using the abstract model appears to be at least no worse than that obtained by using the detailed model (see table 3).

model	testset 1	testset 2	testset 3
detailed	241 ± 19	337 ± 45	1212 ± 182
abstract	235 ± 25	259 ± 32	988 ± 153

Table 3. Average CPU times obtained with abstract and detailed models (in msec; confidence 95%)

Consistent results (both in terms of static reduction of the model size and reduction of diagnoses) have been obtained by applying the abstraction algorithm to other subclasses of manifestations in the model. Space precludes reporting them in this paper.

Please note that the faulty behavioral modes modeled for the components were the ones listed in the FMECA document for the real device, thus proving that the results obtained with the abstraction algorithm and reported above are of interest for a real-world system.

6 Related Work and Conclusions

Literature on MBD contains several proposals to use abstraction as a means of simplifying system model and, consequently, characterization and computation of diagnoses.

Some of them formulate *abstraction rules* and prove that abstractions obtained by their application preserve important properties, i.e. by reasoning at the abstract level we don't overlook any diagnoses ([9], [13]).

Among the rules proposed by Mozetič, rule 1 (Refinment/collapse of values) aims exactly at abstracting sets of values at the detailed level into a single value at a more abstract level. Compared to our approach, however, both Mozetič and Provan assume that the abstraction is done manually, by a human knowledgeable about the system behavior and structure. Moreover, they use the abstract models only in order to reduce the computational complexity, but still return detailed level diagnoses.

In a recent paper ([2]), the authors improve Mozetič approach so that the hierarchy (still manually provided) is automatically rearranged on a case by case basis in order not to hide any available observations from the abstract levels.

Sachenbacher and Struss ([14]) have defined a relational-based approach (i.e. the behavior model is given as a relation R among tuples of variables) for automated abstraction of variables domains and showed its usefulness in building system models by composing sub-system models and then abstracting away values details that are not of interest in the resulting model. In contrast to our approach, their work assumes that a desired abstraction τ_{targ} be given as part of the abstraction problem, together with the restrictions on available observations information τ_{obs} that appear also in our approach.

Travé-Massuyès, Escobet and Milne ([15]) define a notion of *indiscriminability* among faulted components which is somewhat similar to our notion of *indistinguishability* among behavioral modes. Their work, which is based on a relational model of the system, aims at suggesting the addition of sensors in order to make all the possible faults discriminable.

In this paper we have shown how abstraction of variable domains in propositional, qualitative system models, can significantly reduce the average number of admissible diagnoses when only a subset of observables is available.

This is particularly useful in on-line diagnosis, where limitations on the number and/or granularity of observables can likely apply.

The algorithm presented in the paper has a larger applicability than discussed so far. For example, it can be used as a support tool for diagnosability during system modeling (i.e. the algorithm can point out discrepancies between the granularity of the model being defined and that of the system observables).

There are many directions we are considering for extending our work. The current version of the algorithm assumes that in the device structure nodes are connected by at most one directed path, while representation of some systems of practical interest does not obey to this restriction.

We also could explore how our automatic abstraction techniques can be extended in order to merge together components whose contributions to the available observations are indiscriminable (i.e. introducing the notion of indistinguishable components).

REFERENCES

[1] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupré, 'Generating on-board diagnostics of dynamic automotive systems based on qualitative models', *AI Communications*, **12**, 33–43, (1999).

[2] L. Chittaro and R. Ranon, 'Hierarchical diagnosis guided by observations', in *Proc.* 17^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI01*, pp. 573–578, Seattle, USA, (2001).

[3] L. Console and O. Dressler, 'Model-based diagnosis in the real world: lessons learned and challenges remaining', in *Proc.* 16^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI99*, pp. 1393–1400, Stockholm, Sweden, (1999).

[4] L. Console and P. Torasso, 'A spectrum of logical definitions of model-based diagnosis', *Computational Intelligence*, **7**(3), 133–141, (1991).

[5] A. Darwiche, 'Model-based diagnosis using structured system descriptions', *Journal of Artificial Intelligence Research*, **8**, 165–222, (1998).

[6] J. de Kleer, A. Mackworth, and R. Reiter, 'Characterizing diagnoses and systems', *Artificial Intelligence*, **56**(2–3), 197–222, (1992).

[7] J. de Kleer, O. Raiman, and M. Shirley, 'One step lookahead is pretty good', in *Proc.* 2^{nd} *Int. Workshop on Principles of Diagnosis - DX91*, pp. 136–142, Milan, Italy, (1991).

[8] W. Hamscher, 'Modeling digital circuits for troubleshooting', *Artificial Intelligence*, **51**, 223–271, (1991).

[9] I. Mozetič, 'Hierarchical model-based diagnosis', *Int. J. of Man-Machine Studies*, **35**(3), 329–362, (1991).

[10] N. Muscettola, P. Nayak, B. Pell, and B. Williams, 'Remote agent: to boldly go where no ai system has gone before', *Artificial Intelligence*, **103**, 5–47, (1998).

[11] L. Portinale and P. Torasso, 'Diagnosis as a variable assignment problem: a case study in a space robot fault diagnosis', in *Proc.* 16^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI 99*, pp. 1087–1093, Stockholm, Sweden, (1999).

[12] L. Portinale, P. Torasso, and G. Correndo, 'Knowledge representation and reasoning for fault identification in a space robot arm', in *Proc.* 5^{th} *Int. Symposium on AI, Robotics and Automation is Space - iSAIRAS99*, pp. 539–546, Noordwijk, Holland, (1999).

[13] G. Provan, 'Hierarchical model-based diagnosis', in *Proc.* 12^{th} *Int. Workshop on Principles of Diagnosis - DX01*, pp. 329–362, San Sicario, Italy, (2001).

[14] M. Sachenbacher and P. Struss, 'Aqua: A framework for automated qualitative abstraction', in *Proc.* 15^{th} *Int. Workshop on Qualitative Reasoning - QR01*, San Antonio, USA, (2000).

[15] L. Travé-Massuyès, T. Escobet, and R. Milne, 'Model-based diagnosability and sensor placement. application to a frame 6 gas turbine sub-system', in *Proc.* 17^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI01*, pp. 551–556, Seattle, USA, (2001).

Structural Analysis utilizing MSS Sets with Application to a Paper Plant

Mattias Krysander and **Mattias Nyberg**
Department of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
Phone: +46 13 282198, Email:matkr@isy.liu.se, matny@isy.liu.se

Abstract. When designing model-based fault-diagnosis systems, the use of *consistency relations* (also called e.g. *parity relations*) is a common choice. Different subsets are sensitive to different subsets of faults, and thereby isolation can be achieved. This paper presents an algorithm for finding a small set of submodels that can be used to derive consistency relations with highest possible diagnosis capability. The algorithm handles differential-algebraic models and is based on graph theoretical reasoning about the structure of the model. An important step, towards finding these submodels and therefore also towards finding consistency relations, is to find all *minimal structurally singular* (MSS) sets of equations. These sets characterize the fault diagnosability. The algorithm is applied to a large nonlinear industrial example, a part of a paper plant. In spite of the complexity of this process, a small set of consistency relations with high diagnosis capability is successfully derived.

1 Introduction

When designing model-based fault-diagnosis systems, using the principle of consistency based diagnosis [5, 11, 6], a crucial step is the conflict recognition. As shown in [3], conflict recognition can be achieved by using pre-computed consistency relations (also called e.g. *analytical redundancy relations* or *parity relations*). With properly chosen consistency relations, different subsets of consistency relations are sensitive to different subsets of faults. In this way isolation between different faults can be achieved.

The systems considered in this paper are assumed to be modeled by a set of nonlinear and linear differential-algebraic equations. To find consistency relations by directly manipulating these equations is a computationally complex task, especially for large and nonlinear systems. To reduce the computational complexity of deriving consistency relations, this paper proposes a two-step approach. In the first step, the system is analyzed structurally to find overdetermined *submodels*. Each of these submodels are then in the second step transformed to consistency relations. The benefit with this two-step approach is that the submodels obtained are typically much smaller than the whole model, and therefore the computational complexity of deriving consistency relations from each submodel is substantially lower compared to directly manipulating the whole model.

The main contribution and the focus of the paper is a structural algorithm for finding these submodels. Instead of directly manipulating the equations themselves, the proposed algorithm only deals with the structural information contained in the model, i.e. which variables that appear in each equation. This structural information is collected in a *structural model*. In addition to finding all submodels that can be used to derive consistency relations, the algorithm also selects a small set of submodels that corresponds to consistency relations with the highest possible diagnosis capability.

In industry, design of diagnosis systems can be very time consuming if done manually. Therefore it is important that methods for diagnosis-system design are as systematic and automatic as possible. The algorithm presented here is fully automatic and only needs as input a structural model of the system. This structural model can in turn easily be derived from for example simulation models.

Structural approaches have also been studied in other works dealing with fault diagnosis. In [10] a structural approach is investigated as an alternative to dependency-recording engines in consistency based diagnosis. Furthermore a structural approach is used in the study of supervision ability in [2] and an extension to this work considering sensor placement is found in [12].

In Sections 2 and 3, structural models and their usefulness in fault diagnosis are discussed. Then in Section 4, a complete description of the algorithm is given. The algorithm is then in Section 5 applied to a large nonlinear industrial process, a part of a paper plant. In spite of the complexity of this process, a small set of consistency relations with high diagnosis capability is successfully derived.

2 Structural models

The behavior of a system is described with a model. Usually the model is a set of equations. A structural model [2] contains only the information of which variables that are contained in each equation. Let M_{orig} denote the structural model obtained from the equations, describing the system to be diagnosed. This structural model will contain three different kinds of variables: known variables Y, e.g. sensor signals and actuators; unknown variables X_u, for example internal states of the system; and finally the faults F. If faults are decoupled then they will also be included in X_u. The differentiated and non-differentiated version of the same variable are considered to be different variables. The time shifted variables in the time discrete case are also considered to be separate variables.

A structural model can be represented by an *incidence matrix* [4, 1]. The rows correspond to equations and the columns to variables. A cross in position (i, j) tells that variable j is included in equation i.

Example 1 *A simple example is a pump, pumping water into the top of a tank. The water flows out of the tank through a pipe connected to the bottom of the tank. The known variables are the pump input u, the measured water level in the tank y_h, and the measured flow from*

the tank y_f. One fault denoted f_i is assumed to be associated with each known variable. The actual flows to and from the tank are denoted F_i, and the actual water level in the tank is denoted h. Without knowing the exact physical equations describing the analytic model the structural model can be set up as follows:

$equation$	$unknown$				$fault$				$known$	
	F_1	F_2	h	$\dot{h}$	f_u	f_{yh}	f_{yf}	$\dot{f}_f$	u	y_h y_f
e_1	X				X				X	
e_2	X	X		X						
e_3		X				X				X
e_4	X	X								
e_5	X						X			X
e_6								X		

(1)

Equation e_1 describes the pump, e_2 the conservation of volume in the tank, e_3 the water level measurement, e_4 the flow from the tank caused by the gravity, e_5 the flow measurement, and e_6 a fault model for the flow measurement fault f_{yf}.

3 Fault Diagnosis Using Structural Models

The task is to find submodels that can be used to form consistency relations. To be able to draw a correct conclusion about the diagnosability from the structural analysis, it is crucial that for each of these submodels there is a consistency relation that validates all equations included in the submodel. The common definition of consistency relation does not ensure this. Therefore the new definition of *consistency relation for an equation set* is introduced that explicitly points out the submodel considered. Before consistency relation for E is defined some notation is needed.

Let $\mathbf{x}$ and $\mathbf{y}$ denote the vectors of variables contained in X_u and Y respectively. Then $E(\mathbf{x}, \mathbf{y})$ denote an equation set that depends on variables contained in X_u and Y.

Definition 1 (Consistency Relation for E) *A scalar equation $c(\mathbf{y}) = 0$ is a* consistency relation for *the equations $E(\mathbf{x}, \mathbf{y})$ iff*

$$\exists \mathbf{x} E(\mathbf{x}, \mathbf{y}) \Leftrightarrow c(\mathbf{y}) = 0 \tag{2}$$

and there is no proper subset of E that has property (2).

Definition 1 differ from the common definition of consistency relation in two ways, the left implication in (2) and that there is no proper subset of E that has property (2). Refer the latter as the minimality condition in Definition 1. The following example shows the importance of the left implication in (2).

Example 2 *Consider the model $E = \{y_1 = x, y_2 = x, y_3 = x\}$. The equation $y_1 - y_2 = 0$ is not a consistency relation for E, because it is true even if e.g. $y_3 \neq y_1 = y_2$ and then it is impossible to find a consistent x in E. However $y_1 - y_2 = 0$ is a consistency relation for $\{y_1 = x, y_2 = x\}$.*

The expression $y_1 + y_2 - 2y_3 = 0$ includes y_3. The right implication in (2) holds, but the opposite direction does not hold. The conclusion is that also this expression is not a consistency relation for E or any equation subset of E.

However $(y_1 - y_2)^2 + (y_2 - y_3)^2 = 0$ is a consistency relation for E.

The minimality condition in Definition 1 is important, because it guarantees that any invalid equation can infer an inconsistency.

3.1 Basic Assumptions

Basic assumptions are needed to guarantee that the subsets found only by analyzing structural properties are exactly those subsets that can be used to form consistency relations. Before the basic assumptions are presented, some notation is needed. Let E be any set of equations and X any set of variables. Then define $var_X(E) = \{x \in X | \exists e \in E : e \text{ contains } x\}$ and $equ_E(X) = \{e \in E | \exists x \in X : e \text{ contains } x\}$. Also, let $var_X(e)$ and $equ_E(x)$ be shorthand notations for $var_X(\{e\})$ and $equ_E(\{x\})$ respectively. If g is any equation, function or variable, let $g^{(i)}$ denote the i:th time derivative of g. Then define $\overline{var}_X(E) = \{\text{undifferentiated } x | \exists i (x^{(i)} \in var_X(E))\}$, e.g. $\overline{var}_{X_u \cup Y}(\{y = \dot{x}\}) = \{y, x\}$. Finally, the number of elements in any set E is denoted $|E|$.

The first assumption is introduced to ensure that the model becomes finitely differentiated in Section 4.1.

Assumption 1 *The model M_{orig} has the property*

$$\forall E \subseteq M_{orig} : |E| \leq |\overline{var}_{X_u \cup Y}(E)|. \tag{3}$$

The meaning of condition (3) is that each subset of equations include more or equally many different variables, considering derivatives as the same variable. If condition (1) is not fulfilled and there are no redundant equations, the model would normally be inconsistent.

As mentioned earlier, the structural model contains less information than the analytical model. The next assumption makes it possible to draw conclusions about analytical properties from the structural properties.

Assumption 2 *There exists a consistency relation $c(\mathbf{y}) = 0$ for the equation set H iff*

$$\forall X' \subseteq var_{X_u}(H), X' \neq \varnothing : |X'| < |equ_H(X')| \tag{4}$$

According to Assumption 2 the unknown variables in H can be eliminated if and only if it holds that for each subset of variables in H the number of variables is less then the number of equations in H which contain some of the variables in the chosen subset.

The Assumptions 1 and 2 are often fulfilled. For example all subsets of equations found in the industrial example in the end of the paper satisfy Assumption 2. Even though the "only if" direction of Assumption 2 is difficult to validate in an application, the results of the paper can still be used to produce a lower bound of the actual detection and isolation capability.

If all subsets of the model fulfill Assumption 2, the structural analysis will find all subsets that can be used to find consistency relations.

3.2 Finding Consistency Relations via MSS Sets

Now, the task of finding those submodels that can be used to derive consistency relations will be transformed to the task of finding the subsets of equations that have the structural property (4). To do this, two important structural properties are defined [9].

Definition 2 (Structurally Singular) *A finite set of equations E is* structurally singular *with respect to the set of variables X if $|E| > |var_X(E)|$.*

Definition 3 (Minimal Structurally Singular) *A structurally singular set is a* minimal structurally singular *(MSS) set if none of its proper subsets are structurally singular.*

For simplicity, MSS will always mean MSS with respect to X_u in the rest of the text. The next theorem tells that it is sufficient and necessary to find all MSS sets to get all different sets that can be utilized to form consistency relations. The task of finding all submodels that can be used to derive consistency relations has thereby been transformed to the task of finding all MSS sets.

Theorem 1 *Let $H \subseteq M_{orig}$, where M_{orig} fulfills Assumption 1. Further, let H and all E_i fulfill Assumption 2. Then there exists a consistency relation $c(\mathbf{y}) = 0$ for $H(\mathbf{x}, \mathbf{y})$ where $|H| < \infty$ iff $H = \bigcup_i E_i$ where for each i, E_i is an MSS set.*

For a proof, see [7].

4 Algorithm for finding and selecting MSS sets

The objective is to find all MSS sets in a differentiated version of the model M_{orig} and then choose a small subset of these MSS sets with the same diagnosability as the full set of MSS sets. The algorithm can be summarized in the following steps.

Algorithm 1

1. *Differentiating the model: Find equations that are meaningful to differentiate for finding MSS sets.*
2. *Simplifying the model: Given the original model and the additional equations found in step (1), remove all equations that cannot be included in any MSS set. To simplify the next step, merge sets of equations that have to be used together in each MSS set.*
3. *Finding MSS sets: Search for MSS sets in the simplified model.*
4. *Analyzing Diagnosability: Examine the diagnosability of the MSS sets found in step (3).*
5. *Decoupling faults: If the diagnosability has to be improved, some faults have to be decoupled. For decoupling faults, return to step (1) and consider these faults as unknown variables in X_u.*
6. *Selecting a subset of MSS sets: Select the simplest set of MSS sets that contains the desired diagnosability.*

Note that to avoid searching for all MSS sets decoupling all possible faults, Algorithm 1 has been organized so that first, the fault free model is analyzed. Then if it is necessary for achieving higher isolability, faults are decoupled. The following sections discuss each of the steps in Algorithm 1.

4.1 Differentiating the Model

To handle dynamic models, Algorithm 1 needs a way to deal with derivatives. In this section an algorithm for handling derivatives is defined. This algorithm is referred to as Algorithm 2. A small example will show what Algorithm 2 must be capable of handling.

Example 3 *Consider the model $E = \{e_1, e_2, e_3\} = \{y_1 = x, y_2 = \dot{x}, y_3 = x^2\}$. It is obviously impossible to eliminate $\dot{x}$ in e_2 if differentiation of any equation is forbidden. In general, all derivatives of E have to be considered. If $E^{(i)}$ denote the set of the i:th time derivative of each element, the equation set generally considered is $\cup_{i=0}^{\infty} E^{(i)}$.*

Even though $var_{X_u}(e_1) = var_{X_u}(e_3) = \{x\}$ the derivatives of e_1 and e_3 contain different sets of variables, because $var_{X_u}(\dot{e}_1) = \{\dot{x}\} \neq var_{X_u}(\dot{e}_3) = \{x, \dot{x}\}$. Since x is linearly contained in e_1, the variable x in $\dot{e}_1$ disappears. Knowledge about which of the variables that are contained linearly in an equation determines the set of variables in the differentiated equation completely.

For all natural numbers j, $y_1^{(j+1)} - y_2^{(j)} = 0$ is a consistency relation. Most of these consistency relations contain high orders of derivatives of y_1 and y_2. The derivatives of known variables are in general not known, but they can usually be estimated. The higher order of derivative, the more difficult it is to estimate the derivative. Thus it is reasonable to make a limitation $m(y)$ for variable y of the order of derivative that can be considered as possible to estimate. Derivatives up to $m(y)$ are then considered to be known and higher derivatives belong to X_u.

To summarize the example, Algorithm 2 must be capable of differentiating equations. To produce a correct structural representation of differentiated equations, the algorithm must take linearly contained variables into account. Further, it has to handle the limitation $m(y)$ for each $y \in Y$.

Algorithm 2 consists of two parts. The first part is a modification of Pantelides' algorithm [9]. Let $M = \bigcup_{i=1}^{n} \bigcup_{j=0}^{\alpha_i} \{e_i^{(j)}\}$, then α_i is the highest number of differentiations in M of equation i. Then M is a differentiated model of $M_{orig} = \bigcup_{i=1}^{n} \{e_i\}$. Let $\{e_i^{(\alpha_i)} | 1 \leq i \leq n\}$ be the set of most differentiated equations in M. The highest derivative of a non-differentiated variable x in the model M is defined as $\max(\{i | x^{(i)} \in var_{X_u}(M)\})$.

Pantelides' algorithm differentiates equation subsets, so that the original equations together with the differentiated equations have a *complete matching* [4] of the most differentiated equations into the unknown variables with the highest derivatives.

The modification of Pantelides' algorithm is that derivatives of known variables, higher or equal to $m(y)$, are also allowed to be included in the matching.

Algorithm 2

Input: The original model M_{orig}, a description of which variables that are linearly contained, and for each $y \in \overline{var}_Y(M_{orig})$, $m(y) < \infty$.

(1) Apply the modified Pantelides' algorithm to M_{orig} and the limits $m(y)$. The output is the number of times each equation must be differentiated to find all MSS sets.

(2) Differentiate the equations in M_{orig} the number of times suggested in step (1) and use the description of which variables that are linearly contained, to get the correct structural description of the differentiated structural model denoted M_{diff}.

Output: M_{diff}.

It is critical that step (1) in Algorithm 2 terminates, i.e. no equation should be differentiated an infinite number of times. In Pantelides (1988) the condition when the algorithm terminates is stated. This condition can be written as the structural property (3). Since the model M_{orig} has this property according to Assumption 1, the algorithm will terminate.

Let now $MSS(M)$ denote the set of MSS sets found in equations M and $MSS_{all}(M) = MSS(\cup_{i=0}^{\infty} M^{(i)})$. Then it is possible to state the following theorem proven in [7].

Theorem 2 *If Assumption 1 is satisfied and for each $y \in \overline{var}_Y(M_{orig})$, $m(y) < \infty$, then*

$$MSS_{all}(M_{orig}) = MSS(M_{diff})$$

The consequence of this theorem is that all MSS sets that are possible to find if the original model M_{orig} is differentiated an infinite number of times, can always be found in M_{diff}.

Example 4 *The following example is a continuation of Example 1 with the structural model shown in (1). Let* $m(u) = m(y_f) = 1$ *and* $m(y_h) = 0$*. According to Algorithm 1 the first iteration uses the fault free model, i.e. all faults are zero. The equation* e_6 *contains only a fault. Since all faults are at the moment assumed to be zero, then* e_6 *is not considered. Further, assume that no variable is linearly contained in any equation. Then no variable will disappear in the differentiation. The structural model* M_{diff} *obtained from Algorithm 2 is*

$equation$	$unknown$					$fault$				$known$			
	F_1	F_2	$\dot{F}_2$	h	$\dot{h}$	f_u	f_{yh}	f_{yf}	$\dot{f}_f$	u	y_h	y_f	$\dot{y}_f$
e_1	X					X				X			
e_2	X	X			X								
e_3			X				X				X		
e_4		X		X									
$\dot{e}_4$		X	X	X	X								
e_5		X						X				X	
$\dot{e}_5$		X	X					X	X			X	X

(5)

4.2 Simplifying the Model

It is a complex task to find all MSS sets in a structural model. Therefore it can be of great help if it is possible to simplify the model. Here two kinds of simplifications are used.

In a first step, all equations in M_{diff} that include any variable that is impossible to eliminate, are removed. This can be done with Canonical Decomposition [2].

In a second step, variables that can be eliminated without losing any structural information are found. The rest of this section will be devoted to a discussion about this second step.

If there is a set $X \subseteq X_u$ with the property $1 + |X| = |equ_{M_{diff}}(X)|$, then all equations in $equ_{M_{diff}}(X)$ have to be used to eliminate all variables in X. Since all unknown variables must be eliminated in an MSS set this means particularly that all MSS sets including any equation of $equ_{M_{diff}}(X)$ has to include all equations in $equ_{M_{diff}}(X)$. The idea is to find these sets. Then it is possible to eliminate internal variables, here denoted X, in these sets. Every set is replaced with one new equation.

This second simplification step finds subsets of variables that are included in exactly one more equation than the number of variables. To reduce the computational complexity, a complete search for such sets is in fact not performed here. Instead only a search for single variables included in two equations is done. When a variable is included in just two equations, these equations are used to eliminate the variable. If all variables are examined and some simplification was possible, then all remaining variables have to be examined once more. When no more simplifications can be made, the simplification step is finished and the resulting structural model is denoted M_{simp}. Note that with this strategy larger sets than two equations will also be found, since the algorithm can merge sets found in previous steps.

The next theorem ensures that no MSS set is lost in the simplification step.

Theorem 3 $MSS(M_{diff}) = MSS(M_{simp})$

For a proof, see [7]. Consider again Example 4 and the output (5) from the differentiation step. No equations can be removed in the first simplification step.

The second step searches for variables which belong only to two equations. In the first search, the algorithm finds F_1 in $\{e_1, e_2\}$, $\dot{F}_2$ in $\{\dot{e}_4, \dot{e}_5\}$, and $\dot{h}$ in the equations produced by $\{e_1, e_2\}$ and $\{\dot{e}_4, \dot{e}_5\}$. This makes one group of $\{e_1, e_2, \dot{e}_4, \dot{e}_5\}$. This search made simplifications and therefore the search is performed once more. The second time no simplifications have been done and the simplification step is therefore complete. The remaining system is

$equation$	$unknown$		$fault$			$known$			
	F_2	h	f_u	f_{yh}	f_{yf}	$\dot{f}_f$	u	y_h	y_f	$\dot{y}_f$
$\{e_1, e_2, \dot{e}_4, \dot{e}_5\}$	X	X	X		X	X	X		X	X
e_3		X		X				X		
e_4	X	X								
e_5	X				X					

(6)

4.3 Finding MSS Sets

After the simplification step is completed, step (3) in Algorithm 1 finds all MSS sets in the simplified model M_{simp}. This section explains how the MSS sets are found.

The task is to find all MSS sets in the model M_{simp} with equations $\{e_1, \cdots, e_n\}$. Let $M_k = \{e_k, \cdots, e_n\}$ be the last $n - k + 1$ equations. Let E be the current set of equations that is examined. The set of MSS sets found is denoted M_{alg3}. Then the following algorithm finds all MSS sets in M_{simp}.

Algorithm 3
Input: The model M_{simp}*.*

1. *Set* $k = 1$ *and* $M_{alg3} = \varnothing$*.*
2. *Choose equation* e_k*. Let* $E = \{e_k\}$ *and* $X = \varnothing$*.*
3. *Find all MSS sets that are subsets of* M_k *and include equation* e_k*.*
   (a) *Let* $\tilde{X} = var_{X_u}(E) \backslash X$ *be the unmatched variables.*
   (b) *If* $\tilde{X} = \varnothing$*, then* E *is an MSS set. Insert* E *into* M_{alg3}*.*
   (c) *Else take a remaining variable* $\tilde{x} \in \tilde{X}$ *and let* $X = X \cup \{\tilde{x}\}$*. Let* $\tilde{E} = equ_{M_k \backslash E}(\tilde{x})$ *be the remaining equations. For all equations* e *in* $\tilde{E}$ *let* $E = E \cup \{e\}$ *and goto (a).*
4. *If* $k < n$ *set* $k = k + 1$ *and goto number (2).*

Output: The set of MSS sets found, i.e. M_{alg3}*.*

Algorithm 3 finds all MSS sets in M_{orig} according to the next theorem proven in [7].

Theorem 4 $M_{alg3} = MSS(M_{simp})$

The following small example with five equations shows how the algorithm works.

	x_1	x_2	x_3
1	X	X	
2		X	X
3	X	X	X
4	X		
5		X	

This model gives the following time evolution of current equations, i.e. E in Algorithm 3 is

				2			**3**		**2**	
		2	5	**5**		2	**2**	3	**3**	**5**
	3	3	3	**3**	4	4	**4**	4	**4**	**4**
1	1	1	1	**1**	1	1	**1**	1	**1**	**1**
					4					
		4		3	**3**			5		
	3	3	5	5	**5**		4	4		
2	2	2	2	2	**2**	3	3	3	4	5

The bold columns represent the MSS sets found. This example also shows that if there are several matchings including the same equations, the algorithm finds the same subset of equations several times.

4.4 Analyzing Diagnosability

When the MSS sets are found, the next step is to analyze their diagnosability. The continuation of the example in (6) will be used to illustrate how this analysis is done. The 4 MSS sets that can be found in (6) are shown in the left column in Figure 1 (a). The matrix in this figure is the incidence matrix of the MSS sets in (6). If any equation in the MSS set i include fault j, the element (i, j) of the incidence matrix is equal to X. Note that an X in position (i, j) is no guarantee for fault j to appear in the MSS set i. For an example of the interpretation of an incidence matrix, consider the third MSS set in Figure 1 (a). This MSS set could contain f_u and f_{yf}, but it is impossible that it could contain f_{yh}, since f_{yh} is only included in equation e_3. For simplicity, the derivatives of the faults are omitted in Figure 1.

If the number of different faults is large it is not easy to see which faults that can be isolated from each other. The incidence matrix of the MSS sets show which faults that could be responsible for an inconsistency of each MSS set, but it is more interesting to see which faults that can be explained by other faults. A *fault matrix* shows the maximum isolation and detection capability of the diagnosis system. The maximum isolation capability with a diagnosis system designed with this structural method is obtained if it is assumed that each fault makes all MSS sets including this fault inconsistent. If fault j is sensitive to at least all MSS sets that fault i is sensitive to, then element (i, j) of the fault matrix is equal to X. The interpretation of an X in position (i, j) is that fault f_i can not be isolated from fault f_j.

The fault matrix corresponding to the incidence matrix in Figure 1 (a) is shown in Figure 1 (b). Consider the first row of the fault matrix. Suppose that fault f_u is present. Then, the first three MSS sets are not satisfied in an ideal case. This means that f_u certainly can explain fault f_u, but also f_{yf} can explain fault f_u. Fault f_{yh} cannot explain fault f_u, since if f_{yh} is present, the third MSS set is satisfied. Note that the fault matrix is not symmetric. For example fault f_{yf} can explain fault f_u but the opposite is not true. The fault matrix can more easily be analyzed after Dulmage-Mendelsohn permutations [8]. This algorithm returns a *maximal matching* [4] which is in block upper-triangular form. The diagonal blocks corresponds to strong Hall components of the adjacency graph of the fault matrix. The interpretation is that faults in a diagonal block can never be distinguished with that diagnosis system. In the small example in Figure 1 (b), the same matrix is returned after Dulmage-Mendelsohn permutations, which usually is not the case. The diagonal blocks are the 1×1 diagonal elements.

MSS	f_u	f_{yh}	f_{yf}
$\{e_1, e_2, e_3, e_4, \dot{e}_4, \dot{e}_5\}$	X	X	X
$\{e_1, e_2, e_3, \dot{e}_4, e_5, \dot{e}_5\}$	X	X	X
$\{e_1, e_2, e_4, \dot{e}_4, e_5, \dot{e}_5\}$	X		X
$\{e_3, e_4, e_5\}$		X	X

(a)

present fault	interpreted fault f_u	f_{yh}	f_{yf}
f_u	X		X
f_{yh}		X	X
f_{yf}			X

(b)

Figure 1. The incidence matrix of MSS sets is shown in (a). The fault matrix of (a) is shown in (b).

4.5 Decoupling faults

Suppose that the element (i, j) of the fault matrix is equal to X for some $i \neq j$. It could still be possible to isolate fault i from fault j by trying to decouple fault j. Include fault j among the unknown variables X_u and search for new MSS sets by applying Algorithm 1 step (1) to the new model obtained. An MSS set that is able to isolate fault i from fault j has to include at least one equation that includes fault i. If any such MSS set is found, it has to include an elimination of fault j. If not, this MSS would have been discovered earlier.

In the example in Figure 1, the fault matrix shows that f_u and f_{yh} can not be isolated from f_{yf}. The problem is that there is no MSS set that decouple fault f_{yf}. But there could be one if f_{yf} is eliminated. The fault f_{yf} is moved from the faults F to the unknown variables X_u. The procedure starts all over from the step (1) in Algorithm 1. The result is a new MSS set in which f_{yf} is decoupled. This gives a possibility to detect and isolate all faults.

4.6 Selecting a Subset of MSS Sets

It is not unusual that the number of MSS sets found is very large. Many of the MSS sets probably use almost as many equations as unknown variables in the entire system. These MSS sets usually rely on too many uncertainties to be usable for fault isolation. Small MSS sets are more robust and are usually sensitive to fewer faults. Therefore the goal must be to find the set of most robust MSS sets but with the same diagnosis capability as the set of all MSS sets.

Start to sort the MSS sets in an ascending order of complexity. The complexity measure is here the number of equations, even though more informative measures are also a possibility. The MSS sets are examined in the rearranged order. If an MSS set increase the diagnosability, then select the MSS set. The diagnosability is increased if some fault becomes detectable or some fault i can be isolated from some other fault j. This means that for each detection of a fault and for each isolation between two faults, the smallest MSS sets with this diagnosis ability will be one of the chosen MSS sets. In this way the final output from Algorithm 1 will be the most robust set of MSS sets with highest possible diagnosis capability.

5 Industrial example: A part of a paper plant

This example is a stock preparation and broke treatment system of a paper plant located in Australia. The system is used for mixing and purifying recycled paper for production of new paper. An overview of the system is shown in Figure 2.

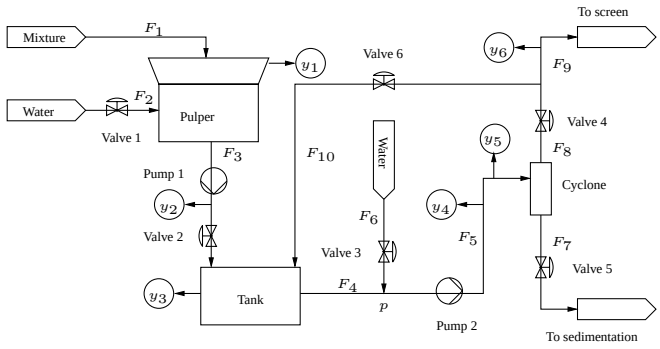


Figure 2. A stock preparation and broke treatment system of a paper plant.

5.1 System Description

Most parts of the system are nonlinear and it is only the tank and the pulper that are considered to be dynamic. The model has shown to compare well to real measured data. Because of space considerations, the details of the model are omitted, but can be found in [7]. The system has 4 states: the volume and concentration in the pulper and in the tank. There are 6 sensors in the system. Sensor y_1 and y_3 measure the water levels of the pulper and the tank respectively, y_2 and y_4 measure concentration, y_5 and y_6 measure pressure. The flows and concentrations into this system are known and the flows out from the system are also known. There are 6 valves and two pumps that are actuators with known inputs.

There are 21 faults that are considered. All sensors can have a constant offset fault $(f_1, \ldots, f_6)$. All valves can have a constant offset in the actuator signal $(f_7, \ldots, f_{12})$. Clogging can occur in the pipes near the valves $(f_{13}, \ldots, f_{18})$ and also directly after the tank f_{19}. Finally, the pumps can have a constant offset in the actuator signal (f_{20}, f_{21}).

The system is described by 29 equations. Equations $(e_1, \ldots, e_4)$ describe the dynamics, $(e_5, \ldots, e_{14})$ are pressure loops, e_{15} relates the concentration in the junction after the tank with the flows F_4 and F_6, (e_{16}, e_{17}) describe the two pumps, $(e_{18}, \ldots, e_{23})$ are valve equations, $(e_{24}, \ldots, e_{26})$ are flow equations, and finally $(e_{27}, \ldots, e_{29})$ are sensor equations for sensor 1, 2, and 3. The structural model for these equations can be viewed in the first 29 rows in the matrices in Figure 3.

5.2 Differentiating the Model

The highest order of derivatives that is known for all known variables are assumed to be one. If a variable is contained linearly in an equation the variable disappears in the differentiated expression. This knowledge is used since the equations are known. Algorithm 2 is applied to the first 29 equations in Figure 3. The result is that all equations except equation 1, 2, 3, and 4 are differentiated. This results in additionally 25 differentiated equations shown in the lower part of Figure 3.

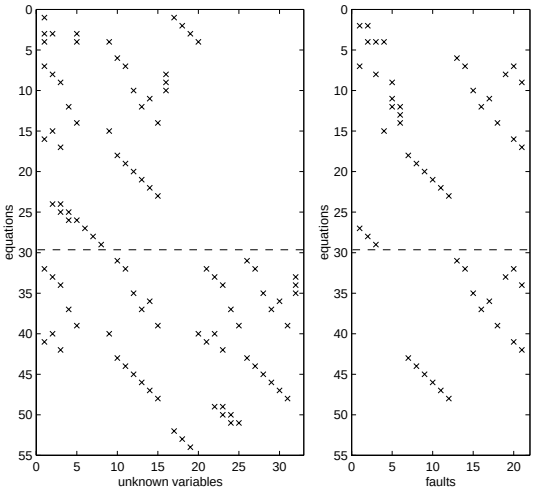
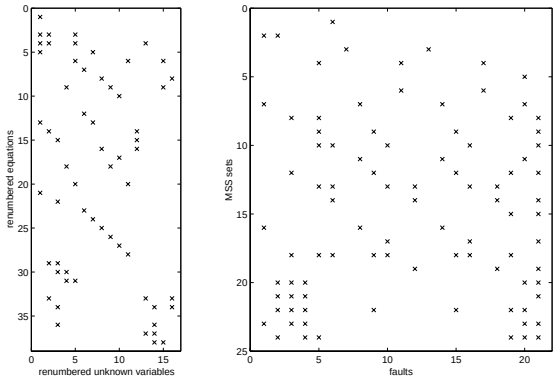


Figure 3. Structural model of the stock preparation and broke treatment system.

5.3 Simplifying the Model

In the first step of simplification applied to the left matrix in Figure 3, the equations $\{27, 28, 29\}$ include variables belonging only to one equation, i.e. they cannot be included in any MSS sets.

The second part of the simplification finds that the variables $\{9, 17, 18, 19, 20, 21, 25, 26, 27, 28, 29, 30, 31\}$ can be eliminated. The equations that form groups are $\{1, 52\}$, $\{2, 53\}$, $\{3, 54\}$, $\{4, 15, 40\}$, $\{32, 41, 44\}$, $\{39, 48, 51\}$, $\{31, 43\}$, $\{35, 45\}$, $\{37, 46\}$ and $\{36, 47\}$. The simplified structural model is shown in Figure 4 (a). Note the simplification of the model by comparing Figure 3 and Figure 4 (a).



(a) (b)

Figure 4. The simplified structural model is shown in (a). The incidence matrix of the MSS sets is shown in (b)

5.4 Finding MSS sets

Algorithm 3 is then applied to the simplified model. The algorithm returns 35770 MSS sets that are contained in the simplified model. The largest MSS set consists of 24 equations.

5.5 Analyzing Diagnosability

The two different fault matrices are seen in Figure 5. The Dulmage-Mendelsohn permutations gives that the faults $\{7, 13\}$, $\{8, 14\}$, $\{9, 15\}$, $\{10, 16\}$,$\{11, 17\}$ and $\{12, 18\}$ are never distinguishable. These pairs of faults all belong pairwise to the same valve. This isolation performance for faults concerning valves is in this case acceptable. To give an example of how elimination of faults is done, the attention is focused on isolating faults 4, 8, and 14.

5.6 Decoupling faults

Considering Figure 5, it is still important to discover if any MSS set can decouple fault 2 or 3 and be sensitive to fault 4. It is also necessary to decouple fault 20. Apply Algorithm 1 to the original model, but where fault 2 now is considered to be an unknown variable. Then apply the Algorithm 1 to the model where faults 3 is decoupled and finally also when fault 20 is decoupled. The algorithm finds thereby additional MSS sets that isolate fault 4, 8, and 14.

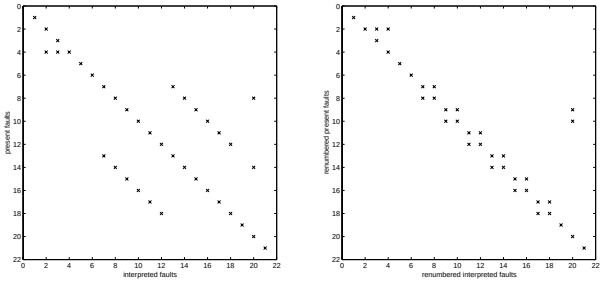
5.7 Selecting a subset of MSS sets

The 24 chosen MSS sets are

MSS	
1	13
2	2 53
3	6 18
4	11 22
5	1 16 52
6	22 36 47
7	7 16 19
8	8 9 17 24
9	9 10 17 20
10	12 17 21 25
11	16 19 32 41 44
12	8 10 17 20 24
13	12 14 21 23 26
14	14 17 23 25 26
15	17 24 33 34 42 49
26	7 16 19 32 41 44
17	17 21 25 37 42 46 50
18	8 10 12 20 21 24 25
19	17 23 25 26 39 42 48 50 51
20	3 4 15 16 17 24 40 42 49 54
21	1 3 4 15 17 24 40 42 49 52 54
22	3 4 8 10 15 16 20 33 35 40 45 54
23	2 3 4 15 16 17 24 40 42 49 53 54
24	3 4 8 9 15 16 17 24 40 42 49 54

(7)

From these sets and the structural model in Figure 3 the incidence matrix in Figure 4 (b) is obtained.



(a) (b)

Figure 5. These matrices are the fault matrices before (a) and after (b) the Dulmage-Mendelsohn permutation.

5.8 Generating Consistency Relations

Consistency relations corresponding to the 24 MSS sets are calculated by using the function Eliminate in Mathematica. Most of the equations in the model are polynomial equations. For polynomial equation-systems, the function Eliminate uses Gröbner Basis techniques for elimination. Each MSS set with 7 or less equations was easily eliminated to a consistency relation. The consistency relations from the MSS set 17 and 18 were obtained from the Eliminate function, but were to complex to be numerically reliable. Elimination of the unknown variables in MSS sets with 8 or more equations was computational intractable with the Eliminate function. Therefore, by using only consistency relations obtained from the 15 first MSS sets, the isolation capability was reduced slightly. Some further results of the investigation can be found in [7].

6 Conclusion

This paper has presented a systematic and automatic method for finding a small set of submodels that can be used to derive consistency relations with highest possible diagnosis capability. The method is based on graph theoretical reasoning about the structure of the model. It is assumed that a condition on algebraic independency is fulfilled.

An important idea, towards finding these submodels, is to use the mathematical concept *minimal structurally singular* sets. These sets have in Theorem 1 been shown to characterize these submodels, i.e. the consistency relations, which give the fault detection and the fault isolation capability.

The method is capable of handling general differential-algebraic non-causal equations. Further, the method is not limited to any special type of fault model. Algorithm 1 finds all submodels that can be used to derive consistency relations and this is proven in Theorem 2, 3, and 4. The key step in Algorithm 1 is step (3) that finds all MSS sets in the model it is applied to.

Finally the method has been applied to a large nonlinear industrial example, a part of a paper plant. The algorithm successfully manage to derive a small set of submodels. In spite of the complexity of this process, a sufficient number of submodels could be transformed to consistency relations so that high diagnosis capability was obtained.

REFERENCES

[1] E. Carpanzano and C. Maffezzoni, 'Symbolic manipulation techniques for model simplification in object-oriented modeling of large scale continuous systems', *Mathematics and Computers in Simulations*, (48), 133–150, (1998).

[2] J. P. Cassar and M. Staroswiecki, 'A structural approach for the design of failure detection and identification systems.', in *IFAC Control of Industrial Systems*, (1997).

[3] M-O. Cordier, P. Dague, M. Dumas, F. Levý, J. Montmain, M. Staroswiecki, and L. Travé-Massuyés, 'AI and automatic control approaches of model-based diagnosis: Links and underlying hypotheses', in *4th IFAC Symosium on Fault Detection Supervision and Safety for Technical Processes*, ed., A.M.Edelmayer, volume 1, pp. 274–279, (2000).

[4] F. Harary, *Graph theory*, Addison-Wesley publishing company, ISBN 0-201-41033-8, 1969.

[5] J. De Kleer, A. K. Mackworth, and R. Reiter, 'Characterizing diagnoses and systems', *Artificial Intelligence*, (1992).

[6] J. De Kleer and B.C. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, (1987).

[7] M. Krysander and M. Nyberg, 'Structural analysis for fault diagnosis of DAE systems utilizing graph theory and MSS sets', Technical Report LiTH-ISY-R-2410, Dept. of Electrical Engineering Linköping University, (2002). URL:http://www.vehicular.isy.liu.se/Publications/.

[8] G. Meurant, *Computer Solution of Large Linear Systems*, Elsevier Science B. V., ISBN 0-444-50169-X, 1999.

[9] C C. Pantelides, 'The consistent initialization of differential-algebraic systems', *SIAM J. SCI. STAT. COMPUT.*, **9**(2), 213–231, (1988).

[10] B. Pulido and C. Alonso, 'An alternative approach to dependency-recording engines in consistency-based diagnosis', in *Lecture Notes in Artificial Intelligence*, volume 1904, pp. 111–121. Artificial Intelligence: Methodology, Systems, and Applications. 9th International Conference, AIMSA 2000., Springer-Verlag, Berlin, Germany, (2000).

[11] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, (1987).

[12] L. Travé-Massuyès, T. Escobet, and R. Milne, 'Model-based diagnosability and sensor placement. application to a frame 6 gas turbine subsystem', in *DX01 twelfth international workshop on principals of diagnosis*, ed., D. T. Dupre' S. Mcilraith, pp. 205–212, (2001).

Diagnostic Reasoning with Multilevel Set-Covering Models

Joachim Baumeister [1] and **Dietmar Seipel** [1]

Abstract. We consider multilevel set-covering models for diagnostic reasoning: though a lot of work has been done in this field, *knowledge acquisition* efforts have been investigated only insufficiently. We will show how set-covering models can be build incrementally and how they can be refined by knowledge enhancements or representational extensions. All these extensions have a primary characteristic: they can be applied without changing the basic semantics of the model.

Keywords: set-covering diagnosis; model-based diagnosis; qualitative modeling; knowledge acquisition; abductive reasoning

1 Introduction

In this paper we will present a new interpretation of set-covering models [1] which is a suitable representation for the manual development of knowledge-based systems. Because of its simple semantics set-covering models are rapidly understood by the experts, but still maintain a well-known model-based interpretation. In [2] we showed how knowledge-based diagnostic systems can be developed incrementally with set-covering models, thus supporting rapid prototyping of such systems. In this paper we will extend this approach to multilevel set-covering models, and we will describe how simple set-covering models can be enhanced by representational extensions. Practical experience has shown that these additions facilitated the development of a real world example from a medical ICU domain.

A *set-covering model* consists of a set of diagnoses, a set of findings (observations) and covering relations between the elements of these two sets. There exists a covering relation between a diagnosis and a finding, iff the diagnosis implies the observation of the finding. We can define covering relations between diagnoses as well, iff a diagnosis implies the observation of another diagnosis. The basic idea of set-covering diagnosis is the detection of a reasonable set of diagnoses which can explain the given observations. To do this, we propose an abductive reasoning step: Firstly, hypotheses are generated in order to explain the given observations. Secondly competing hypotheses are ranked using a *quality measure*.

Reasoning with set-covering models has got a long tradition in diagnostic reasoning: Early work was done by Patil [3] with his system ABEL, which implemented a comprehensive set-covering representation including causal, associational and grouping relations. Reggia et al. [1] contributed a formal approach to set-covering models and addressed the problem of hypothesis generation with a precise description of *generator sets*. Later [4] they introduced the integration of Bayesian probabilities in set-covering models. With the system MOLE [5] Eshelman focussed on the problem of acquiring set-covering knowledge. He proposed an interactive process that allows for refining previously acquired knowledge after a reasoning step to differentiate between conflicting hypotheses. Console et al. [6] showed with the system CHECK how to combine heuristic and causal knowledge. There heuristic knowledge was used to find reasonable hypotheses for a given observation. In a second step the causal knowledge was used to generate abductive explanations for the hypotheses. Long [7] extended covering models with probabilities and a rich syntax of temporal and non-temporal causation events. Since knowledge acquisition is a cost sensitive task, reuse of existing knowledge is another emerging aspect. Puppe [8] showed how set-covering knowledge can be combined with other classes of knowledge like heuristic rules, case-based knowledge or decision trees.

Most of these approaches only investigated syntax and semantics of the reasoning process, but did not consider the knowledge engineering process. Eshelman's MOLE system [5] differs from our knowledge acquisition approach, since there knowledge refinement is performed by adding new covering relations to the model. In our paper we will present (multilevel) set-covering models and show how to enrich these simple models with knowledge enhancements like *similarities* and *weights* or *representational extensions* for more complex covering relations. A primary characteristic of the presented extensions is the incrementality: each extension can be applied independently from other enhancements and will not change the basic semantics of the model, but refine special aspects of it.

The rest of the paper is organized as follows: In Section 2 we will introduce the basic concepts of set-covering models and show how to enrich set-covering models with additional knowledge like similarities and weights. Beyond that we will introduce representational extensions of set-covering models in Section 3 that will enable us to formulate exclusions, necessary relations and complex covering relations (conjunctions, disjunctions, cardinalities). In Section 4 we will shortly summarize the problem of hypothesis generation and we will introduce constraints that shrink the exponentiell size of possible hypotheses. We will conclude this paper in Section 5 with an overview of the work we have done so far and promising directions we are planning to work on in the future.

2 Set-Covering Models

A set-covering model consists of a set of diagnoses, a set of findings (observations) and covering relations between the elements of these two sets. There exists a covering relation between a diagnosis and

[1] University of Würzburg, Department of Computer Science, Am Hubland, 97074 Würzburg, Germany, email: {baumeister, seipel}@informatik.uni-wuerzburg.de

a finding, iff the diagnosis predicts the observation of the finding. Furthermore we can define covering relations between two diagnoses to state that a diagnosis implies another diagnosis. In this way we can build a *covering-tree* for a diagnosis, where we postulate that the leafs of the covering-tree have to be observable findings. So each covering path will start with a diagnosis and lead to an observable finding.

2.1 The Basic Model

The basic idea of set-covering diagnosis is the detection of a reasonable set of diagnoses which can explain the given observation of findings. In an *abductive reasoning* step hypotheses are firstly generated in order to explain the given observations (*hypothesis generation*). In a second step, we define a quality measure for ranking competing hypotheses (*hypothesis testing*). Set-covering models describe relations like:

> A diagnosis D predicts that the parameters $A_1, \ldots, A_n$ are observed with corresponding values $v_1, \ldots, v_n$.
> A diagnosis D predicts the diagnoses $D_1, \ldots, D_m$.

We call each of these relations *covering relations* and we denote them by

$$r_i = D \rightarrow A_i : v_i, \quad 1 \leq i \leq n,$$
$$r'_i = D \rightarrow D_i, \quad 1 \leq i \leq m.$$

Covering models can be visually described like in Figure 1. In this

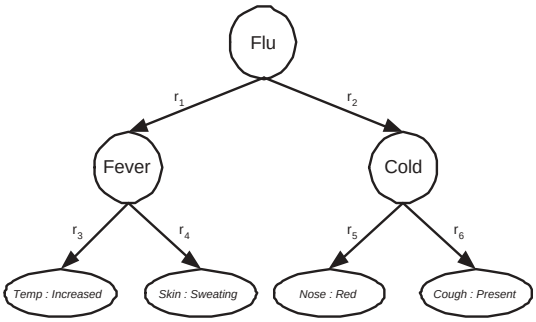


Figure 1. Basic set-covering model for diagnoses *Flu*, *Fever* and *Cold*.

example the model states that diagnosis *Flu* implies the observation of the diagnoses *Fever* and *Cold*. Diagnosis *Fever* itself forces the observation of the attributes *Temperature* and *Skin* with their corresponding values *Increased* and *Sweating*.

The basic algorithm for set-covering diagnosis is very simple: Given a set of observed findings, it uses a simple hypothesize-and-test strategy, which generates hypotheses (coined from diagnoses) in the first step and tests them against the given observations in a second step. The test is defined by calculating a quality measure, which expresses the covering degree of the hypothesis regarding the observed findings. The generation and evaluation of the hypotheses is an iterative process, which stops when a satisfying hypothesis has been found or all hypotheses have been considered. Usually the algorithm will look at single diagnoses, compute the corresponding quality measure, and then it will generate hypotheses with multiple diagnoses, if needed. In the worst case this procedure will generate 2^n candidates for n diagnoses. So heuristics are needed to keep the method computationally tractable (c.f. Section 4).

The basic sets for this task are the following: We define $\Omega_{\mathcal{D}}$ to be the set of all diagnoses and $\Omega_{\mathcal{A}}$ the set of all observable parameters (attributes). To each parameter $A \in \Omega_{\mathcal{A}}$ a range $dom(A)$ of values is assigned, and $\Omega_{\mathcal{V}} = \bigcup_{A \in \Omega_{\mathcal{A}}} dom(A)$ is the set of all possible values for the parameters. If a parameter A is assigned to a value v, then we call $A : v$ a *finding*.

$$\Omega_{\mathcal{F}} = \left\{ A : v \mid A \in \Omega_{\mathcal{A}},\ v \in dom(A) \right\}$$

is the set of all findings. Furthermore we call an element $S \in \Omega_{\mathcal{S}} = \Omega_{\mathcal{D}} \cup \Omega_{\mathcal{F}}$ a *state*.

A *covering relation* r between a diagnosis D and a state S ($S \neq D$) is denoted by $r = D \rightarrow S$. We say that "D predicts S" or that "D covers S". Then $c_r = D$ is called the *cause* and $e_r = S$ is called the *effect*. We define $\Omega_{\mathcal{R}}$ to be the set of all covering relations contained in the model. Then $D^+ \in \Omega_{\mathcal{R}}$ is the set of all covering relations with diagnosis D as the cause, i.e. $D^+ = \{ r \in \Omega_{\mathcal{R}} \mid c_r = D \}$. E.g., for the model in Figure 1 we obtain $c_{r_1} = Flu$ and $e_{r_1} = Fever$, $Cold^+ = \{r_5, r_6\}$.
Since S can be a diagnosis itself, we are able to build *multilevel* set-covering models. A state S *transitively covers* another state S', if either S covers S' or S covers another state S'' that transitively covers S'.

We call $\mathcal{F}_{\mathcal{O}} \subset \Omega_{\mathcal{F}}$ the set of *observed findings* and a set $\mathcal{H} \subseteq \Omega_{\mathcal{D}}$ of diagnoses a *hypothesis*. A finding that is not transitively covered by the hypothesis $\mathcal{H}$ is called *isolated*, and the set of all observed findings that are isolated will be denoted by $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}} \subseteq \mathcal{F}_{\mathcal{O}}$. E.g. for a hypothesis $\mathcal{H} = \{D_1\}$ and $\mathcal{F}_{\mathcal{O}} = \{A_1 : v_1, A_2 : v_2, A_4 : v_4\}$ we obtain $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}} = \{A_2 : v_2\}$.

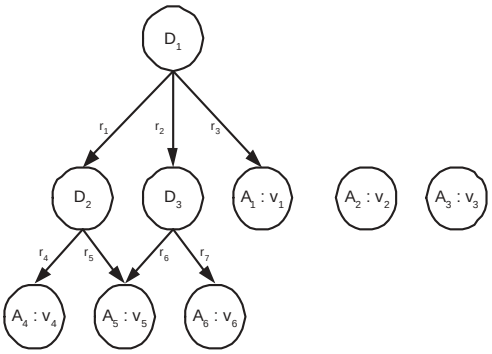


Figure 2. Basic set-covering model for diagnosis D

Now we will describe the computation of the precision of a state for a given observation. The precision $\pi(S)$ of a state S provides a real value between 0 and 1 to describe the degree of accuracy the covered states of S are observed.

Bottom-Up Computation of Precisions. Given the set $\mathcal{F}_{\mathcal{O}}$ of observed findings, the precision π of each state is computed bottom-up starting with the findings:

$$\pi(A:v) = \begin{cases} 1, & \text{if } A:v \in \mathcal{F}_{\mathcal{O}} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

The precision $\pi(D)$ of a diagnosis D can be computed as soon as the precisions of all its successors S are known. For this we define

$$\begin{aligned} D^+_{\geq c} &= \{\, r \in D^+ \,|\, \pi(e_r) \geq c(e_r) \,\}, \\ D^+_{>0} &= \{\, r \in D^+ \,|\, \pi(e_r) > 0 \,\}, \end{aligned}$$

as the sets of all *relevant* covering relations, i.e. relations that predict states with a precision greater than a user defined threshold function.

$$\pi(D) = \begin{cases} \dfrac{\sum\limits_{r \in D^+_{\geq c}} \pi(e_r)}{|D^+_{>0}|}, & \text{if } D^+_{>0} \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The denominator counts all successor states of D with a positive precision, which gives us the maximally achievable score. The nominator sums up the precision of all successor states with a precision, that is greater than or equal to the completeness value, which gives us the actually achieved score.

The *completeness value* $c(D)$ of a diagnosis is specified by the modeler and is motivated by the fact, that a covering model for a diagnosis will contain more states than the diagnosis will cause in an average case. Nevertheless in most cases the observation of a percentage of the modeled states will legitimate the validation of this diagnosis. To emphasize this percentage the modeler has to specify a completeness value $c(D)$. Unless this factor is reached by the observation set in the current case, the diagnosis may neither be considered as a validly observed state, nor will it be considered as a valid hypothesis candidate.

Since we also want to consider multiple faults, i.e. hypotheses containing more than one diagnosis, we define

$$\mathcal{H}^+ = \bigcup_{D \in \mathcal{H}} D^+ \qquad \mathcal{H}^+_{>0} = \bigcup_{D \in \mathcal{H}} D^+_{>0} \qquad \mathcal{H}^+_{\geq c} = \bigcup_{D \in \mathcal{H}} D^+_{\geq c}$$

The covering relations $r \in \mathcal{H}^+_{\geq c}$ are called *relevant* for $\mathcal{H}$. Observe, that relevancy depends on $\mathcal{F}_\mathcal{O}$, since the precisions have been computed based on $\mathcal{F}_\mathcal{O}$.

Quality Measures. The quality measures are used to rank the possible hypotheses with respect to the given observation. As we already introduced the precision of a single diagnosis we now will define the *quality* of a hypothesis, which can contain multiple diagnoses. The quality of a hypothesis provides a real value between 0 and 1 to describe the degree of accuracy with which the hypothesis $\mathcal{H}$ can explain the given observation $\mathcal{F}_\mathcal{O}$.

Definition 2.1 (Quality Measure) The *quality* $\varrho(\mathcal{H})$ of a hypothesis $\mathcal{H}$ is given by

$$\varrho(\mathcal{H}) = \frac{\sum\limits_{r \in \mathcal{H}^+_{\geq c}} \pi(e_r)}{|\mathcal{H}^+_{>0}| + |\,\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}\,|}. \tag{3}$$

Notice that, in contrast to the precision, the quality measure does not evaluate a single diagnosis with respect to the transitively observed predictions, but assesses a hypothesis (containing possibly multiple diagnoses) on the basis of the transitively predicted and observed findings and the unexplained (isolated) findings.

We see that $\varrho(\mathcal{H}) \in [0,1]$ for any hypothesis $\mathcal{H} \in \Omega_\mathcal{H}$: The lower bound 0 is obtained, if $\mathcal{H}^+_{\geq c} = \emptyset$. The upper bound 1 is obtained, if all predictions are fully observed, i.e. $\mathcal{H}^+_{\geq c} = \mathcal{H}^+_{>0}$, and the set $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}} = \emptyset$.

Example. For the covering relation given in Figure 2, the set

$$\mathcal{F}_\mathcal{O} = \{\, A_2\!:\!v_2, A_3\!:\!v_3, A_4\!:\!v_4, A_5\!:\!v_5, A_6\!:\!v_6 \,\}$$

of findings, and the hypothesis $\mathcal{H} = \{D_1\}$, we obtain $\pi(D_2) = 1$, $\pi(D_3) = 1$ (with $c(D_2) = c(D_3) = 0.7$). Since we obtain $\mathcal{H}^+ = \{r_1, r_2, r_3\}$ for hypothesis $\mathcal{H}$ we can calculate

$$\begin{aligned} \mathcal{H}^+_{\geq c} &= \{\, r_1, r_2 \,\}, \\ \mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}} &= \{\, A_2\!:\!v_2, A_3\!:\!v_3 \,\}. \end{aligned}$$

Up to now we presented the basic representation for set-covering models containing diagnoses and findings connected with covering relations. Of course this simple representation might not always meet the requirements of real world applications. Therefore we will shortly present knowledge extensions of set-covering models. In [2] we showed how to apply these extensions in an incremental way.

2.2 Extension by Similarities and Weights

Similarities between findings and weights for states provide significant knowledge extensions for set-covering models. In the following we will show how to include these enhancements into the quality measures given above.

Similarities. Consider a parameter A with the domain

$$dom(A) = \{\, no,\ si,\ mi,\ hi \,\},$$

with the meanings normal (*no*), slightly increased (*si*), medium increased (*mi*), and heavily increased (*hi*), where $A:hi$ is predicted. We clearly see that the observation $A\!:\!mi$ deserves a better precision than the observation $A\!:\!no$. Nevertheless the simple quality measure considers both observations as unexplained findings and makes no difference between the similarities of the parameter values. For this reason we want to define *similarities* as an extension to set-covering models.

We define the similarity function

$$sim : \Omega_\mathcal{V} \times \Omega_\mathcal{V} \rightarrow [0,1]$$

to capture the similarity between two values assigned to the same parameter. The value 0 means no similarity and the value 1 indicates two equal values. In cluster analysis problems this function is also called *distance function* (cf. [9]).
With similarities we need to adapt Equation (1) for computing the precision of findings.

$$\pi\big(A\!:\!v\big) = sim\big(\, Val_\mathcal{H}(A),\, Val_{\mathcal{F}_\mathcal{O}}(A)\big),$$

where Val returns the value of a specified attribute contained in a specified set of states.

$$Val : 2^{\Omega_\mathcal{S}} \times \Omega_\mathcal{A} \rightarrow \Omega_\mathcal{V}.$$

If no special similarity is included in the model, then we get the simple quality measure by defining the *default similarity* $sim(v, v') = \delta_{v,v'}$, where $\delta_{v,v'} = 1$, if $v = v'$, and $\delta_{v,v'} = 0$, otherwise.

Weights. The introduction of weights for covered states is another common generalization of the basic covering model. Here we apply

a weight function $w : \Omega_S \rightarrow I\!N_+$, to emphasize that some states (findings and diagnoses) have a more significant pathological importance than other states.
When applying weights to the model we need to adapt Equation (2) which calculates the precision for a given diagnosis:

$$\pi(D) = \begin{cases} \dfrac{\sum\limits_{r \in D^+_{\geq c}} w(e_r) \cdot \pi(e_r)}{\sum\limits_{r \in D^+_{>0}} w(e_r)}, & \text{if } D^+_{>0} \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Like for the precision of a diagnosis, we need to adapt Equation (3) to calculate the quality of a given hypothesis:

$$\varrho(\mathcal{H}) = \frac{\sum\limits_{r \in \mathcal{H}^+_{\geq c}} w(e_r) \cdot \pi(e_r)}{\sum\limits_{r \in \mathcal{H}^+_{>0}} w(e_r) + \sum\limits_{F \in \mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}} w(F)}$$

If all states have the same weight, i.e., $w(S) = 1$ for all $S \in \Omega_S$, then the model reduces to the simple covering model.

In addition to similarities and weights we already have introduced uncertain covering relations and causal effect functions as possible extensions (cf. [2]).

3 Complex Covering Relations

In the previous section we introduced the basic set-covering model and extensions that allow for the refinement of set-covering knowledge build with basic covering relations. In this section we propose some further extensions of the representation, AND-, OR- and [MIN, MAX]-relations.

To keep the interpretation of covering models simple, we only allow these extensions for covering relations between diagnoses and (directly observable) findings.

3.1 Conjunction of Covering Relations

It is desirable to be able to represent conjunctions between covering relations. An AND-*covering relation*

$$D \rightarrow_{\text{AND}} \{ F_1, \ldots, F_n \}$$

denotes the characteristic, that all covering relations $D \rightarrow F_i$ have to be fulfilled simultaneously.

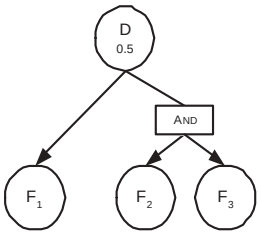


Figure 3. Covering relation $D \rightarrow_{\text{AND}} \{F_2, F_3\}$

Then the weights of the AND-connected findings F_i will only contribute to the precision of D if *all* of these findings are observed. If not all findings are observed, then D cannot explain the findings and we have to check if another diagnosis from the hypothesis can explain these observations. All remaining findings – so far unexplained – will be added to the set of isolated findings $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$. This will decrease the quality measure for the current hypothesis, since $\mathcal{H}^+_{\geq c}$ will not contain relations covering the unexplained observations. Given an AND-covering relation of the form

$$r = D \rightarrow_{\text{AND}} \{ F_1, \ldots, F_n \}$$

we define for each $F_i \in \{ F_1, \ldots, F_n \}$:

$$\pi_r(F_i) = \begin{cases} \pi(F_i), & \text{if for all } F_j \in e_r : \pi(F_j) > 0 \\ 0, & \text{otherwise} \end{cases}$$

We try to explain all findings F_i with $\pi_r(F_i) = 0$ but $\pi(F_i) > 0$ by other diagnoses $D' \in \mathcal{H} \setminus \{D\}$. All remaining findings F_i, which cannot be explained by other diagnoses are added to $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.

Example. Assume that we have the covering model of Figure 3, where $c(D) = 0.5$, and we observe the set $\mathcal{F}_\mathcal{O} = \{F_1, F_2\}$. Then $\pi(F_3) = 0$, since F_3 is not in $\mathcal{F}_\mathcal{O}$. Therefore not all precisions of the AND-covered findings are greater than 0, and we define $\pi_r(F_2) = \pi_r(F_3) = 0$. We obtain $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}} = \{F_2\}$ for hypothesis $\mathcal{H} = \{D\}$. Notice, that F_3 is not in $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$, since it is not observed.

3.2 Disjunction of Covering Relations

We also can express alternative covering relations with disjunction. Here we can distinguish between inclusive (OR) and exclusive (XOR) disjunctions.

In Figure 4 we can see two different disjunctive covering relations for diagnosis D: in the left one the findings F_2, F_3 are connected with the OR-covering relation $D \rightarrow_{\text{OR}} \{F_2, F_3\}$, whereas at the right side the findings are connected with an XOR-covering relation $D \rightarrow_{\text{XOR}} \{F_2, F_3\}$. These OR/XOR-relations state, that only one

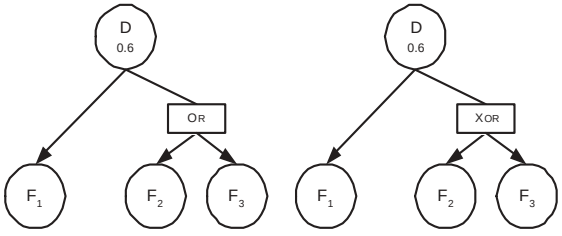


Figure 4. OR-/XOR-covering relations.

of the connected finding has to be observed to fulfill the relation. Of course we need to consider the different semantics in covering models. When computing the quality measures we have to take the following three cases into account:

1. If *none* of the predicted findings is observed, then nothing has to be done. The covering relations connected with the OR/XOR-condition cannot contribute to the quality measure of the parent state.

2. If *one* of the predicted findings is observed, then we simply cut all other states connected by OR/XOR-relations from the model. When computing the quality measure we only take the observed finding into account.
3. If *more than one* of the predicted findings are observed (e.g. $\{F_2, F_3\} \subseteq \mathcal{F}_\mathcal{O}$), then we have to differentiate between OR and XOR relations. For both we take the finding with the maximal contribution; e.g. regarding the weighted precision

$$\pi_w(F) = \pi(F) \cdot w(F).$$

For OR-relations we simply ignore the remaining observations for assessing the quality. They will neither contribute to the quality of the hypothesis nor will they need to be explained by other diagnoses.
For XOR-relations the observations left over still have to be explained. Like for the AND-relations we try to explain them with the other diagnoses contained in the current hypothesis. All remaining findings, that cannot be explained by other diagnoses, are added to the set of isolated findings $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.

We see that we carefully have to use OR/XOR-relations, because of their different interpretation of the observation. For example, multiple observations of one XOR-covering relation are taken negatively into account (i.e., they are assumed to be unexplained findings of the current hypothesis), whereas in ordinary OR-relations they will not contribute in any way.

As shown for AND-covering relations we also have to locally define the precision for OR/XOR-covered findings in context of the given diagnosis: Consider an OR-relation (analogous for XOR):

$$r = D \rightarrow_{\text{OR}} \{ F_1, \ldots, F_n \}.$$

We select a finding $F_{max} \in \{ F_1, \ldots, F_n \}$, such that $\pi_w(F_{max}) = max\big(\pi_w(F_i),\, 1 \leq i \leq n\big)$. Then we say that

$$\pi_r(F_i) = \begin{cases} \pi(F_i), & \text{if } F_i = F_{max} \\ 0, & \text{otherwise.} \end{cases}$$

If there is more than one F_i with maximum weighted precision $\pi_w(F_i)$, then all but one (randomly selected) finding will set to the precision $\pi_r(F_i) = 0$.
When we compute the precision $\pi(D)$ of a diagnosis D, then the precisions of the findings F_i that are covered by an OR/XOR-covering relation contribute with the measure $\pi_r(F_i)$ and not with the usual precision measure $\pi(F_i)$.
For XOR-relations we have to explain the remaining findings by other diagnoses contained in the hypothesis or add them to $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.

3.3 Cardinalities in Covering Relations

Another enrichment of the set-covering representation is the connection of covering relations by cardinality constraints. We express such cardinalities by [MIN, MAX]-*covering relations*. Consider the example in Figure 5. The covering relation between diagnosis D and the findings F_1, F_2, F_3, F_4 and F_5 means, that between 2 and 4 of the predicted findings have to be observed. We denote such relations by

$$r = D \rightarrow_{[2,4]} \{ F_1, F_2, F_3, F_4, F_5 \}.$$

When we interpret [MIN, MAX]-relations $r = D \rightarrow_{[\text{MIN},\text{MAX}]} \mathcal{F}$, then we have to consider three possible cases for the number $k = |\mathcal{F} \cap \mathcal{F}_\mathcal{O}|$ of relevant findings:

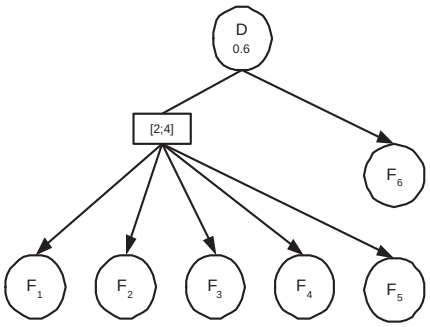


Figure 5. A [MIN, MAX]-covering relation.

1. If $k \in [\text{MIN}, \text{MAX}]$, then all findings in $\mathcal{F} \cap \mathcal{F}_\mathcal{O}$ will contribute.
2. If $k >$ MAX, then let $\mathcal{F}_{max} \subsetneq \mathcal{F} \cap \mathcal{F}_\mathcal{O}$ be the MAX findings with the maximum weighted precisions among the findings in $\mathcal{F}$ (i.e. $|\mathcal{F}_{max}| =$ MAX). We explain the findings in $\mathcal{F}_{max}$ by D. Then we try to explain the findings in $(\mathcal{F} \cap \mathcal{F}_\mathcal{O}) \setminus \mathcal{F}_{max}$ by other diagnoses also contained in the hypothesis. These findings $(\mathcal{F} \cap \mathcal{F}_\mathcal{O}) \setminus \mathcal{F}_{max}$, which we cannot explain by other diagnoses $D' \in \mathcal{H} \setminus \{D\}$, are added to $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.
3. If $k <$ MIN, then we try to explain all findings in $\mathcal{F} \cap \mathcal{F}_\mathcal{O}$ by other diagnoses $D' \in \mathcal{H} \setminus \{D\}$. Findings, which cannot be explained by other diagnoses, are added to $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.

We integrate [MIN, MAX]-relations into set-covering models by locally defining the precision for findings connected by a [MIN, MAX]-relation $r = D \rightarrow_{[\text{MIN},\text{MAX}]} \mathcal{F}$. Then we say that for each $F \in \mathcal{F}$:

$$\pi_r(F) = \begin{cases} 0, & \text{if } k < \text{MIN} \\ \text{or} & \text{if } k > \text{MAX} \wedge F \notin \mathcal{F}_{max} \\ \pi(F'), & \text{if } k \in [\text{MIN}, \text{MAX}] \\ \text{or} & \text{if } k > \text{MAX} \wedge F \in \mathcal{F}_{max} \end{cases}$$

where $\mathcal{F}_{max}$ is again the set of the MAX findings with the best weighted precisions among the findings in $\mathcal{F}$.
When calculating the quality measure for a diagnosis or hypothesis we apply the precision $\pi_r(F)$ for all findings F connected by the relation r. Findings F with $\pi_r(F) = 0$ but $\pi(F) > 0$ need to be explained by other diagnoses contained in the hypothesis or will be added to $\mathcal{F}^{isolated}_{\mathcal{H},\mathcal{O}}$.

It is worth mentioning that ordinary covering relations for a diagnosis are following a similar concept, since we only will consider predicted findings that are also observed but not all predicted findings of the diagnosis. But as opposed to [MIN, MAX]-relations all observed predictions will contribute to the quality. In [MIN, MAX]-relations only MAX observed findings will contribute; more than MAX findings have to be explained by other diagnoses. In general, an ordinary covering model for a diagnosis D with n covered findings is comparable to a $[c(D) \cdot n, n]$-relation connecting the n findings.

3.4 Bounded Covering Relations

The introduction of similarities for finding values is a useful knowledge extension. Nevertheless in some situations the expert wants to express that a relation is only fulfilled if a covered parameter is observed with exactly the predicted value, rather than a similar value. Therefore we supplement necessary covering relations, disjunctive,

conjunctive and constrained covering relations with the optional label *bounded*. We obtain the required behaviour by locally defining the *default similarity* measure for bounded relations:

$$sim\big(Val_{\mathcal{H}}(A),\, Val_{\mathcal{F}_\mathcal{O}}(A)\big) = \delta_{Val_{\mathcal{H}}(A),\, Val_{\mathcal{F}_\mathcal{O}}(A)}.$$

I.e., only if a parameter A is observed with the predicted value, then 1 is assigned to its precision.

4 Constraints for Hypothesis Generation

As mentioned in the introduction of Section 2, the problem of hypothesis generation is exponential, since for n diagnoses we need to consider about 2^n hypotheses in the worst case for an observation. In the following we want to sketch some heuristics to restrict the hypothesis space.

In a first step, we will filter all diagnoses $D \in \Omega_\mathcal{D}$, that are *relevant*, i.e. having the minimum precision. For this, we define the set of relevant diagnoses

$$\Omega_D^{rel} = \big\{ D \in \Omega_\mathcal{D} \,\big|\, \pi(D) \geq c(D) \big\}.$$

Then, only diagnoses $D \in \Omega_D^{rel}$ will be taken into account, when generating hypotheses. Before describing concepts to shrink the set of hypotheses, we will define *generators* as a compact representation for sets of hypotheses, which had been introduced by Reggia et al. [1].

Definition 4.1 (Generator) A generator $\mathcal{G}_I = \{G_1, \ldots, G_n\}$ consists of non-empty pairwise-disjoint subsets $G_i \subseteq \Omega_D^{rel}$ The hypotheses $\mathcal{H}_{\mathcal{G}_I}$ generated by $\mathcal{G}_I$ is defined as

$$\mathcal{H}_{\mathcal{G}_I} = \big\{ \mathcal{H} \subseteq \Omega_\mathcal{D} \,\big|\, |\mathcal{H} \cap G_i| \leq 1, \text{ for all } 1 \leq i \leq n \big\}.$$

For $\mathcal{G}_I = \emptyset$, it holds that $\mathcal{H}_{\mathcal{G}_I} = \{\emptyset\}$. We can see, that $\mathcal{H}_{\mathcal{G}_I}$ is analogous to a cartesian set product.

For example, for the set-covering model defined in Figure 1 and $\mathcal{F}_\mathcal{O} = \{temp : inc, skin : sweat, nose : red\}$, we obtain $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$ with $\mathcal{G}_1 = \big\{\{\text{cold}\}, \{\text{fever}\}\big\}$ and $\mathcal{G}_2 = \big\{\{\text{flu}\}\big\}$. So we can compute $\mathcal{H}_\mathcal{G} = \big\{\emptyset, \{\text{cold}\}, \{\text{fever}\}, \{\text{cold, fever}\}, \{\text{flu}\}\big\}$ to be the set of interesting hypotheses.

A method for computing and updating generator sets is extensively described in [4]. Generators are used to efficiently generate hypotheses in an incremental manner: In a first step, sets of generators describing higher level diagnoses (concepts) are created. For hypotheses containing higher level diagnoses and having a high quality measure, we build sets of generators containing underlying specialized diagnoses and test them with their corresponding quality measure. In the following, we introduce two basic knowledge extension, that additionally shrink the space of generated hypotheses.

4.1 Exclusion Constraints

We can define *exclusion constraints* to filter diagnoses from the process of hypotheses generation. In general, two kinds of constraints are possible:

$\neg(D \wedge F_1 \wedge \cdots \wedge F_n)$
If findings $F_1, \ldots, F_n$ are observed, then remove generated hypotheses, containing diagnosis D.

$\neg(D_1 \wedge \cdots \wedge D_m)$
Remove generated hypotheses, containing all the diagnoses $D_1, \ldots, D_m$ at the same time.

Thus, we create hypotheses using generator sets and check each generated hypothesis against the available exclusion constraints. If one exclusion constraint evaluates true, the hypothesis is discarded.

It it worth noticing, that the modification of generator sets with respect to exclusion constraints yields a combinatorial size of generators and therefore is not reasonable. An evaluation of the generated hypotheses according to existing exclusion constraints has been proven to be more efficient.

4.2 Necessary Covering Relations

A stronger type of covering relations are *necessary covering relations*. A necessary covering relation between a diagnosis D and a finding F_1 means, that D *necessarily covers* F_1 and that F_1 always has to be observed if D is hypothesized. We depict a necessary covering relation with $D \xrightarrow{nec} F_1$ as shown in Figure 6.

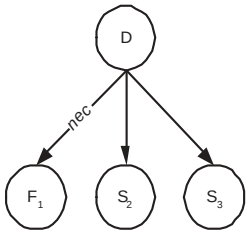


Figure 6. Necessary Covering relation for a diagnosis D.

For applying necessary covering relations we introduce an adapted definition of the precision π_{nec} for each diagnosis $D \in \Omega_\mathcal{D}$:

$$\pi_{nec}(D) = \begin{cases} 0, & \text{if } \exists\, r \in \Omega_\mathcal{R} : r = D \xrightarrow{nec} F \text{ with} \\ & \quad F \in \Omega_\mathcal{F} \wedge \pi(F) < \tau \\ \pi(D), & \text{otherwise} \end{cases}$$

where $\tau \in [0, 1]$ is a specified threshold, which defines when a finding is sufficiently observed (e.g. $\tau = 0.8$).

Therefore a diagnosis D does not propagate any contribution to its parent states until all necessarily covered findings are (sufficiently) observed. Consequently, D will not appear in any generator and thus will not be included in any hypothesis.

5 Conclusions and Future Work

After describing the basic structures of set-covering relations we have shown how to enrich the model with additional knowledge like similarities or weights. We also considered the computation of quality measures of these parts. Furthermore, we have shown representational extensions to the set-covering model to facilitate necessary, disjunctive, conjunctive or constrained covering relations. An important characteristic of all these extensions is the incrementality: some enhancements can be added to refine special aspects of the model but will not change its basic semantics; others are used to guide the process of candidate generation.

In the future we are planning to work on the following fields: Incremental development requires restructuring the model from time to time. We are currently working on restructuring methods for set-covering models that do not alter the basic semantics but improve the design of the diagnosis knowledge. In software engineering *refactoring* [10, 11] has been emerged as the corresponding method. In general we have to look at *validation techniques* for set-covering models besides simple case testing. Because of the special structure of the model we also¡ have to consider static verification techniques for the set-covering representation. For a survey in this field we refer to [12, 13, 14, 15].

In this paper we presented a hand-driven development of set-covering models. But it seems to be possible to *learn* coarse models *automatically* from a small number of available cases. Later on these models should be refined by the developer with additional knowledge. With such a semi-automatic development step, the initial costs of knowledge acquisition can be reduced conveniently. Some work in this field has been done by Thompson et al. [16] and Wang et al. [17]. This step is not considered if we have a sufficiently large set of data, since then traditional machine learning methods (e.g. learning neural networks, learning Bayes networks) seem to be more appropriate.

ACKNOWLEDGEMENTS

The authors would like to thank Frank Puppe for his helpful suggestions and comments.

REFERENCES

[1] James A. Reggia, Dana S. Nau, and Pearl Y. Wang. Diagnostic Expert Systems Based on a Set Covering Model. *Journal of Man-Machine Studies*, 19(5):437–460, 1983.

[2] Joachim Baumeister, Dietmar Seipel, and Frank Puppe. Incremental Development of Diagnostic Set–Covering Models with Therapy Effects. In *Proceedings of the KI-2001 Workshop on Uncertainty in Artificial Intelligence*, Vienna, Austria, 2001.

[3] Ramesh S. Patil, Peter Szolovits, and William B. Schwartz. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. *In: Szolovits, P. (Ed.). Artificial Intelligence in Medicine, Westview Press, Boulder, Colorado*, 1982.

[4] Yun Peng and James A. Reggia. *Abductive Inference Models for Diagnostic Problem-Solving*. Springer, Berlin, 1990.

[5] Larry Eshelman. *Mole: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems*, pages 37–79. In: Sandra Marcus (ed.): Automating Knowledge Acquisition for Expert Systems. Kluwer Academic Publishers, 1988.

[6] Luca Console, Luigi Portinale, Daniele Theseider Dupr, and Pietro Torasso. Combining Heuristic and Causal Reasoning in Diagnostic Problem Solving. In Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors, *Second Generation Expert Systems*, pages 46–68. Springer, 1993.

[7] William J. Long. Temporal Reasoning for Diagnosis in a Causal Probabilistic Knowledge Base. *Artificial Intelligence in Medicine*, 8(3):193–215, 1996.

[8] Frank Puppe. Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3. *Int. J. Human-Computer Studies*, 49:627–649, 1998.

[9] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, California, 2000.

[10] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois, Urbana-Champaign, IL, USA, 1992.

[11] Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.

[12] Anca Vermesan and Frans Coenen. *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice*. Kluwer Academic Publishers, 1999.

[13] Alun Preece. Building the Right System Right. In *Proceedings of KAW'98 Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, 1998.

[14] Frans Coenen and Trevor Bench-Capon. *Maintenance of Knowledge-Based Systems*. Academic Press, 1993.

[15] Marc Ayel and Jean-Pierre Laurent. *Validation, Verification and Test of Knoweldge-Based Systems*. Wiley, 1991.

[16] Cynthia A. Thompson and Raymond J. Mooney. Inductive Learning for Abductive Diagnosis. In *Proceedings of the AAAI-94, Vol. 1*, pages 664–669, 1994.

[17] Xue Z. Wang, M.L. Lu, and C. McGreavy. Learning Dynamic Fault Models based on a Fuzzy Set Covering Method. *Computers in Chemical Engineering*, 21:621–630, 1997.

Computing Minimal Conflicts for Rich Constraint Languages

Jakob Mauss[1] and Mugur Tatar[1]

Abstract. We address here the following question: Given an inconsistent theory, find a minimal subset of it responsible for the inconsistency. Such conflicts are essential for problem solvers that make use of conflict-driven search (cf. [2, 4, 9]), for interactive applications where explanations are required (cf. [16, 22]), or as supporting tools for consistency maintenance in knowledge-bases (cf. [11]). Conflict computation in AI applications was usually associated with dependency recording as performed by TMSs (cf. [2, 3, 18]). This techniques, however, have a rather limited applicability for languages that go beyond the expressiveness power of propositional logic. For more powerful languages and solvers constraint suspension appeared, until now, to be the only available alternative for the computation of minimal conflicts.

We present here an algorithm for computing minimal conflicts that can be used with powerful constraint languages, e.g. possibly including finite and non-finite variable domains, algebraic and FD constraints, etc. The conflicts are extracted post mortem from the proof (a tree with inferences of the form $A \wedge B \Rightarrow C$) that lead to the derivation of the inconsistency by an informed search that computes and generalizes conflicting relations. The algorithm is based on a simple but powerful principle that allows to recursively decompose the minimization problem into smaller sub-problems. This principle can also lay the foundation for efficient constraint suspension algorithms that can be used in case no intermediary results are cached during the constraint solving, i.e. in case no proof structures are available.

1 INTRODUCTION

For problems expressed using propositional logic or using finite-domain (FD) constraints there exist some efficient solutions for the computation of conflicts and explanations (cf. [13, 16, 18]). Unfortunately, this is not the case for more expressive constraint languages. Due to the scope of our application interests, namely supporting engineering tasks such as safety and diagnosability analysis and also design and configuration (cf. [12, 15, 20, 22]), we are especially interested in modeling languages adequate for engineering problems. Such languages have to mix logical and FD constraints with (more or less) classical systems of linear and non-linear algebraic or even differential equations. The general purpose techniques that can be applied in this case for the (minimal) conflict computation are constraint suspension (cf. [7]) and TMS-like dependency recording (cf. [3]). Constraint suspension can guarantee conflict minimality, but it is in many cases too inconvenient due to the large amount of time required to recompute many subsets of the initial problem. When applied to systems of equations where local value propagation is not enough for solving, TMS-based architectures usually become a heavy machinery that consumes considerable amounts of time and memory (see also [17]) and, in the end, still do not have any guarantees for conflict minimality – the minimality is (at most) guaranteed with respect to the propositional clauses that represent the dependencies and not with respect to the semantic of the original constraint language. The following example is an attempt to illustrate this. Consider a system of five algebraic constraints

$A_1 \equiv x > 4$ $\quad A_3 \equiv y \geq 2$ $\quad A_5 \equiv x > 2\,y + 1$
$A_2 \equiv x < 5$ $\quad A_4 \equiv y \leq 2$

A solver may process these constraints in 4 steps as shown in Figure 1. In step ④, they are discovered inconsistent. A minimal conflict among the given constraints is $\{ A_2, A_3, A_5 \}$. If the solver were using dependency recording it would not find the above minimal conflict – just the trivial $\{A_1, A_2, A_3, A_4, A_5\}$ in this case!

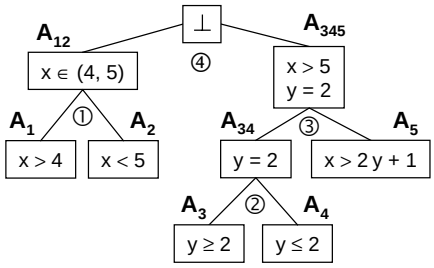


Figure 1. Tree for proving the inconsistency of 5 constraints.

Of course, this was a just simple example where no symbolic variable elimination was required and, for the above example, one can easily define a strategy to handle correctly the conflict computation – for instance by maintaining separate dependencies for lower and upper bounds of intervals as in [6]. However, this unnecessarily overloads the solving process in case of consistency and, still, would not solve the problem in general.

In contrast, the key idea of this paper is to do a (guided) post mortem analysis of the context in order to compute the minimal conflicts. The algorithm uses the information that A_{345} is conflicting with A_{12} (we say that A_{345} is a conflicting relation for A_{12}) and propagates and updates these conflicting relations through the proof tree in order to select only those parts of it that are really contributing to the conflict. The paper is organized as follows: in section 2 we present the basic procedure for extracting a minimal conflict from a binary proof tree. In section 3 we describe how a constraint solver can control the inferences in order to easily provide such trees. In section 4 we report some first empirical results regarding the performance of the algorithms. Section 5 concludes the paper with a comparison to related work.

[1] DaimlerChrysler AG, Research and Technology, RIC/EK
Alt-Moabit 96a, D-10559 Berlin, Germany.
Email: {jakob.mauss, mugur.tatar}@DaimlerChrysler.com

2 COMPUTING MINIMAL CONFLICTS

We assume in the following a relational framework, i.e. constraints are noted as *relations* over variables with finite or continuous domains. These relations may be represented extensionally (as in Figure 4), or intensionally using formulas (as in Figure 1). In relational terms, '$\wedge$' represents the join (intersection) of relations, falsity '$\bot$' is represented by the empty relation, and the implication $A \Rightarrow B$ is interpreted as subset relation $A \subseteq B$. A set of constraints forms a *conflict* if it is not satisfiable, i.e. in relational terms, if the join of the relations representing the constraints is the empty relation. Given an initial set of inconsistent constraints, we are interested in extracting a *minimal conflict*, i.e. a minimal subset that is still inconsistent. Of course, there can be more than one minimal conflict in an inconsistent context, but we focus for the moment on finding just one such minimal conflict. In the following, we show how to extract a minimal conflict from a binary proof tree such as the one shown in Figure 2. The initial constraints appearing as leaves in the proof tree are also called *assumptions* in the following.

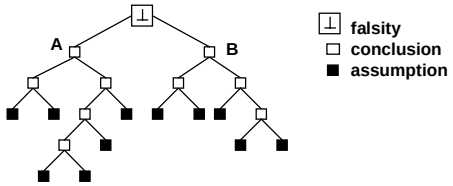


Figure 2. Tree proving the inconsistency of 11 assumptions.

Assume that we have two conflicting relations A, B, none of them being empty, i.e. $A \neq \bot$, $B \neq \bot$, and $A \wedge B \Rightarrow \bot$. Then we have to consider two cases.

1. A and B are both assumptions. In this case, {A, B} is the minimal conflict.
2. At least one of A, B is not an assumption. Assume without loss of generality that A has been derived from A_1 and A_2, i.e. $A_1 \wedge A_2 \Rightarrow A$. Let now: $C_1 := A_1 \wedge B$ and $C_2 := A_2 \wedge B$. We can then distinguish 4 cases, as shown in Figure 3.

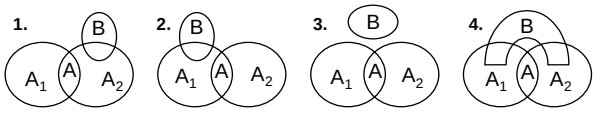


Figure 3. Four cases distinguished by computing intersections. Each relation is depicted as a set of variable assignments.

2.1: $\mathbf{C_1 = \bot \wedge C_2 \neq \bot}$ In this case, the assumptions leading to the derivation of A_2 do not contribute to the conflict with B. Consequently, we can prune the whole sub-tree A_2 and continue the conflict search in A_1.

2.2: $\mathbf{C_1 \neq \bot \wedge C_2 = \bot}$ Analog to case 2.1. A_1 can be ignored.

2.3: $\mathbf{C_1 = \bot \wedge C_2 = \bot}$ There are at least two independent conflicts with B, at least one in the sub-tree A_1, and at least one in A_2. If we want to find just one conflict then we can non-deterministically decide to skip one of the sub-trees.

2.4: $\mathbf{C_1 \neq \bot \wedge C_2 \neq \bot}$ All minimal conflicts are spread across both sub-trees. A minimal conflict has to be composed from a partial solution retrieved from the sub-tree A_1 and an appropriate completion retrieved from the sub-tree A_2. If B was a conflicting relation for A, then C_1 is a conflicting relation for A_2 and C_2 is a conflicting relation for A_1. With these new conflicting relations we can descend recursively in the A_1, A_2 sub-trees and collect the sub-conflicts.

This case analysis leads to the following procedure for extracting a minimal conflict from a proof tree.

Specification: Let

R be a non-empty set of relations (assumptions)
A the root of a binary proof tree with the leaves given by R
B a conflicting relation for A, i.e.: $B \neq \bot$ and $A \wedge B = \bot$.

The proof tree satisfies the requirement that, for any non-leaf node A: $\text{left}(A) \wedge \text{right}(A) \Rightarrow A$.

The procedure XC1(A, B) returns one minimal and non-empty set $M \subseteq R$ such that $(\wedge M) \wedge B = \bot$. As a consequence, if A is the root of a refutation tree then XC1(A, ⊤) returns a minimal conflict from the tree - where ⊤ represents the universal relation i.e. the complement of ⊥.

```
XC1(A, B)
⓪ if (isLeaf(A)) return { A }
   A1 ← left(A)
   A2 ← right(A)
   C1 ← A1 ∧ B
   C2 ← A2 ∧ B
① if (C1 = ⊥ and C2 ≠ ⊥)   return XC1(A1,B)
② if (C1 ≠ ⊥ and C2 = ⊥)   return XC1(A2,B)
③ if (C1 = ⊥ and C2 = ⊥)   return XC1(A1,B)
                      or   return XC1(A2,B)
④ if (C1 ≠ ⊥ and C2 ≠ ⊥)
             M1 ← XC1(A1, C2)
             M2 ← XC1(A2, (∧M1) ∧ B)
             return M1 ∪ M2
```

In case ④ the procedure first descends in the sub-tree A_1 with C_2 as conflicting relation. Before it descends in the sub-tree A_2 we, however, have to generalize A_1 to $\wedge M_1$ and C_1 to $(\wedge M_1) \wedge B$. This is necessary in case we have *several* minimal conflicts that span over the sub-trees A_1 and A_2 in order to select from A_2 a sub-conflict that is part of the same conflict as the sub-conflict that was non-deterministically chosen (case ③) from the sub-tree A_1. Such a case is also illustrated by the following example.

Example Consider the set $R = \{A_1, \ldots A_5\}$ shown in Figure 4. The constraints are extensionally defined relations in this example. E.g. $A_1 \equiv$ '(x = a ∧ y = 1) ∨ (x = b ∧ y = 0)'. R is inconsistent, actually it contains two minimal conflicts. Figure 5 shows how XC1 computes one of them. Circled numbers correspond to the five cases marked in the pseudo code above.

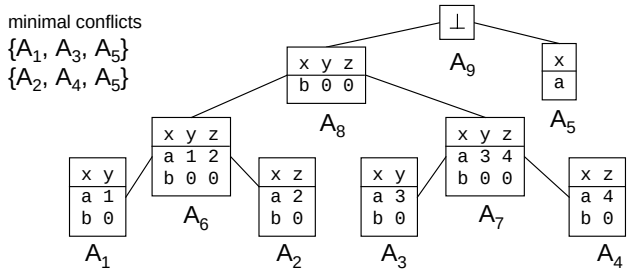


Figure 4. A proof tree for $R = \{A_1, A_2, A_3, A_4, A_5\}$

The crucial part of the procedure is handled in case ④, where a minimal conflict is composed as a disjoined union of two sets M_1 and M_2 computed using the left and right sub-tree. Note that the second set M_2 depends on the first set M_1. During the recursive call at A_6 the procedure non-deterministically decides to select the conflict containing A_1. This decision is reflected in the arguments of the succeeding call at A_7 in order to select the right sub-conflict - i.e A_3 and not A_4 which could be erroneously selected if we did not update the conflicting relation for A_7!

Some properties of XC1 that are worth discussing are:
(1) During top-down traversal of the proof tree, only direct fathers of the nodes contained in the returned minimal conflict are visited. Sub-trees not involved in the minimal conflict are pruned without investigating their nodes. The worst-case appears when the pruning is not effective and we have to inspect the whole tree (always in case ④). For a tree with *n* leaves there are no more than *4(n-1)* joins for the worst case (see also the incremental computation of $\wedge M_1$ later on). However, the complexity of the conflict minimization crucially depends on the complexity of the basic join operations.

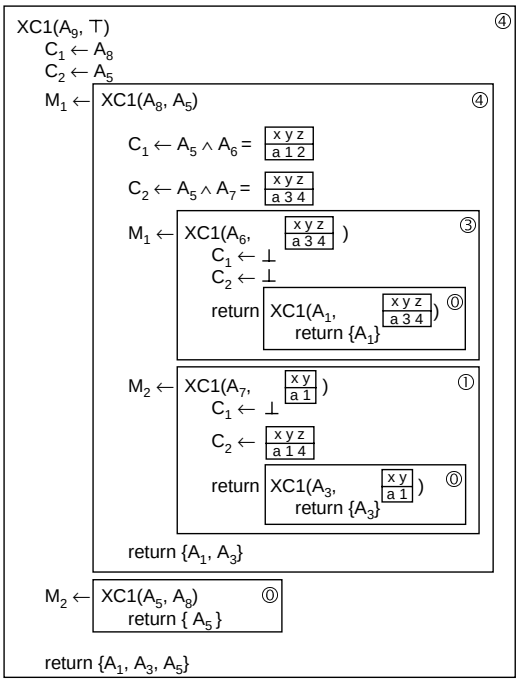


Figure 5. Trace of computation of a minimal conflict

(2) An inference engine will be unable in general to provide complete implementations of the join and empty-check operations - for instance in case we are dealing with systems of non-linear equations. When used in conjunction with a correct but *incomplete* inference engine, XC1 may return a non-minimal conflict. The conflict 'minimallity' is only relative to the completeness degree of the inference services supplied by the solver.

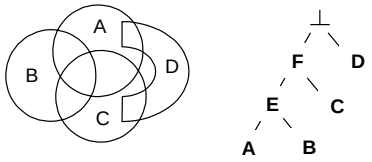


Figure 6. Two minimal conflicts {B, D} and {A, C, D}

(3) The procedure can easily be extended to return several minimal conflicts instead of only one. Basically, in case ③, one can continue search in both sub-trees, instead of non-deterministically choosing one of them. However, this simple extension of XC1 will not always return all of the minimal conflicts. See Figure 6 for an example. The second conflict {A, C, D} is missed, when using the given proof tree. Anyway, the computation of all minimal conflicts from a context can require significantly more effort and is seldom justified in practice.

(4) There are several obvious improvements of the efficiency of XC1 as given above. If the proof structure is a tree then $M_1 \cup M_2$ can be computed as a disjoined union in case ④. If case ③ is always mapped to (say) case ① then the computation of C_2 is required only if $C_1 \neq \bot$. The repeated $\wedge$ computations $\wedge M_1$ can be avoided if XC1, in addition to returning the set M, also returns the join $\wedge M$, which allows for an incremental computation of the conjunction in case ④. Moreover, the generalization of the conflicting relation C_1, i.e. $(\wedge M_1) \wedge B$ in case ④, is required only if it is a strict generalization, i.e. if M_1 is a strict subset of the leaves of A_1.

3 DERIVING PROOF TREES

In the previous section we have seen how to extract one or several minimal conflicts from a proof structure. In this section we sketch how a constraint solver operating on a set R of input relations can control the inference in order to

1. check whether R is consistent, i.e. whether $\wedge R \neq \bot$
2. solve R for any variable
3. provide the proof structure required for conflict computation.

Conflict computation using XC1 works however with any well-formed proof structure, irrespective if the proof was generated by a solver like the one described in this section or not[2].

We note with V(A) the set of *variables* constrained by a relation A. $\pi(A, X)$ denotes the *projection* of A onto a variable set X. The projection $\pi(A, X)$ results from eliminating all variables $V(A) \setminus X$ from the relation A. For example, if

$A \equiv x^2 + y^2 < 1$ $\quad$ $B \equiv (x\text{-}1)^2 + y^2 < 1$

then $\pi(A \wedge B, \{x\}) = \text{'}0 < x \wedge x < 1\text{'}$

and $\pi(A \wedge B, \{y\}) = \text{'}\text{-}1 < y \wedge y < 1\text{'}$.

The projection operation is an abstraction (generalization) operation, i.e. $A \Rightarrow \pi(A, X)$. Hence, the computation $C := \pi(A \wedge B, X)$ can be seen as an inference of the form $A \wedge B \Rightarrow C$. We call such an inference, i.e. projecting the join of two relations A and B onto a variable set X, an *aggregation*.

The computed proofs will contain aggregations as the only kind of inference. The proof structures will be used to derive minimal conflicts, or minimal explanations of variable solutions.

The consistency check, may seem trivial to specify. We could simply ask the solver to compute $\wedge R$ to check whether $\wedge R \neq \bot$. However, in the practical applications with which we are commonly confronted, R may contain hundreds of algebraic and logical constraints with thousands of variables. In this case, the intermediate relations created during the computation of $\wedge R$ would be huge. Instead, following [1], after computing a single conjunction $A \wedge B$, we eliminate all those variables from the result that do not occur in the remaining relations. Consequently, the intermediate relations remain 'small' – the size depending, of course, on the degree of connectivity of the constraint network. This works fine, as long as a variable is shared by a relatively small number of constraints. If the connectivity degree increases (cf. induced width w* in [5]), then many of the aggregations degrade to simple joins and the approach is likely to become inappropriate.

[2] Such proof structures can be recovered, for instance, also from the well-founded-support recorded by a TMS (cf. [18]) - in which case XC1 could be used for further conflict minimization (recall that a TMS guarantees minimality only with respect to *propositional* dependencies and not with respect to the more expressive constraint language).

The creation of a proof tree for the consistency check is given by the following procedure.

Specification: Let R be a non-empty set of assumptions, $\bot \notin R$. Then the procedure isConsistent(R) returns true, iff R is satisfiable.

```
isConsistent(R)
  if ( |R|= 1 )
     return true
  else
     choose {A, B} ⊆ R
     S ← R \ {A, B}
     X ← (vars(A) ∪ vars(B)) ∩ (∪ vars(S))
     C ← π(A ∧ B, X)
     if (C = ⊥) return false
     return isConsistent(S ∪ { C })

vars(A)
  if (A is a leaf)
     return V(A)
  else
     return vars(left(A)) ∪ vars(right(A))
```

Obviously, the procedure isConsistent computes a proof tree containing aggregations as the only kind of inference. Therefore, we call this an *aggregation tree*. If A is the root of an aggregation tree for an inconsistent set R of assumptions then, as shown in section 2, XC1(A, T) returns a minimal conflict. To keep the conflicting relation B small, we may add a projection step $B \leftarrow \pi(B, vars(A))$ as first instruction in XC1. The strategy used to choose a pair of relations for aggregation may for example minimize the variable set X, or try to achieve a balanced tree.

For checking the consistency of *n* assumptions, isConsistent computes *n - 1* aggregations. A significant feature of proof trees as derived above is their ability to support *incremental context analysis*. Assume we have performed a consistency check for a set R of *n* assumptions, and we want to analyze a second context R', constructed by replacing an assumption A in the proof tree for R by a new assumption B with the same variable set. In order to check the new context $R \cup \{B\} \setminus \{A\}$, we only have to re-compute the inferences on the path from A to the root of the proof tree, i.e. if the proof tree is balanced, we only have to compute *log(n)* aggregations. As we see next, the computation of variable solutions can be performed using aggregations as well.

Specification: Let R be a non-empty consistent set of assumptions, and A the root of an aggregation tree computed by the procedure isConsistent. Then the procedure solve(A, T) computes for every variable x in R the solution $S[x] := \pi(\wedge R, \{x\})$.

```
solve(A, B)
   if (A is a leaf) return
   B ← π(B, vars(A))
   A1 ←  left(A);   A2 ← right(A)
   A12 ← A1 ∧ A2
   X ← vars(A1) ∪ vars(A2) \ vars(A)
   for each x ∈ X
      S[x] ← π( A12 ∧ B, {x})
   solve(A1, A12 ∧ B)
   solve(A2, A12 ∧ B)
```

If we take a closer look at the procedure solve, we note that each S[x] is the root of a proof structure defined by a sequence of aggregations. In this case the proof is not purely a tree, it is actually a DAG because some nodes are used more than once. Still, the proof is well-formed, i.e. there are no cyclic justifications. XC1 can be modified to cope with the DAG structure. The resulting procedure XE1(S[x], ¬S[x]) returns a *minimal supporting set of assumptions* for the solution of x, i.e. a minimal subset $E \subseteq R$ such that $S[x] = \pi(\wedge E, \{x\})$.

4 APPLICATION AND EMPIRICAL RESULTS

We have recently finished a prototype implementation of a Relational Constraint Solver (RCS) that follows the principles described in this paper, including the computation of explanations and conflicts. RCS is already integrated in our environment for engineering knowledge management, and its integration in MDS [12] is planned to follow.

In this section we compare XC1 with the conflict computation based on naive constraint suspension. Let R be an inconsistent set of relations. Then the procedure MC(R, {}) returns a minimal conflict, computed by constraint suspension.

```
MC(R, M)
  if R = {} return M
  else choose A ∈ R
       if ∧(R ∪ M \ {A}) = ⊥
            return MC(R \ {A}, M)
       else return MC(R \ {A}, M ∪ {A})
```

The procedure MC resembles Junker's ROBUSTXPLAIN [8], which may use a trailing-mechanism not described in [8] to perform incremental (i.e. fast) consistency checking. If $|R| = n$ and a consistency check for R requires *n* aggregations, then MC(R, {}) needs $O(n^2)$ aggregations for computing a minimal conflict. In contrast, XC1 requires only *O(n)* aggregations for the same task, given an arbitrary, not necessarily balanced tree. Our implementation of MC uses an incremental consistency check as explained in section 3 – thus, a check requires only *O(log(n))* instead of *O(n)* aggregations in the best case.

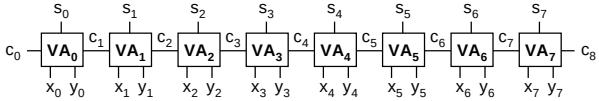


Figure 7. An 8-bit full adder

For the empirical comparison, we used a set R of 137 relations, representing eight 1-bit full adders connected in series as shown in Figure 7, and the assignments $c_0 = 1$, and for $0 \le k \le 7$: $x_k = 0$, $y_k = 1$. If we add one more relation of the form $c_k = 0$, then R becomes inconsistent and contains a minimal conflict M of size $|M| = 2 + 3k$. This gives us 8 different sets R_k of size 138, containing a minimal conflict M of size 2 + 3 k.

conflict detection		MC			XC1		t(MC)
|M|	***n***	**∧, π**	**∧**	**π**	**∧**	**π**	**t(XC1)**
2	8.5	59.1	95.7	91.6	8.8	3.5	446
5	32.6	120.8	76.2	59.4	31.4	13.9	12
8	52.4	129.1	108.3	80.4	52.5	22.9	6.9
11	70.5	131.5	140.4	103.6	72.8	31.5	6.4
14	93.0	133.6	178.5	130.1	93.9	40.4	5.4
17	107.7	134.9	208.5	151.1	113.5	48.3	4.1
20	122.6	135.8	247.5	179.8	131.3	55.7	2.5
23	134.8	136.7	278.3	201.0	151.8	63.7	1.3

Figure 8. Empirical results

For each k isConsistent(R_k) is run for consistency check and it returns a refutation tree that is used as input by both MC and XC1. The leaves of this tree represent an initial (not necessarily minimal) conflict of size *n* - see Figure 8. The table gives the average results obtained for running both algorithms 100 times for all eight R_k. For each run, we permutated the order of the input relations which resulted in different structures of the derived aggregation trees. The columns in the table denote the average number of join and project

operations needed for conflict detection in isConsistent and by the subsequent minimization call to MC or to XC1. The last column gives the ratio of the measured runtimes for MC and XC1. For example, for the case of a minimal conflict of size 2, the average initial conflict provided by isConsistent has size 8.5 and it takes 59.1 aggregations (join followed by project) to detect the conflict. MC needs then 95.7 more joins and 91.6 projections to minimize the initial conflict by suspension, while XC1 is 446 times faster than MC and needs only 8.8 joins and 3.5 projections for the same task. The performance gain of XC1 relative to MC depends strongly on the structure of the proof trees supplied by the solver – i.e. whether the conflicting assumptions are uniformly spread among the leaves of the tree or whether they are clustered in a few sub-trees.

5 RELATED WORK AND DISCUSSION

Dependency recording, like the one performed by TMSs (cf. [2, 3, 18]) works relatively fine as long as we stay in a propositional framework (or, anyway, in a finite world). In more expressive frameworks these techniques gradually become both

- very resource consuming (in time and space)
- incomplete with respect to the more expressive framework.

Constraint suspension is another technique used for conflict computation. It is in general expensive because it relies on performing the consistency check many times for different subsets of the initial problem. A recent enhancement to constraint suspension is the one reported in [8]. The performance of the conflict computation is improved there in two ways:

(a) by adding the constraints to the solver's store one after the other and performing each time a complete consistency check, one knows that the last constraint added that caused the store to become inconsistent is part of all conflicts from the already considered subset; and

(b) by employing an intelligent search, where sets of constraints are simultaneously suspended and then are binary split if necessary.

The proof structure corresponding to the control strategy assumed by Junker is a linear tree. We do not need to enforce the sequential consistency check as assumed by (a). We can assume any clustering technique, such as the ones resulting after structure analysis, e.g. cycle-cutset, hypertree decomposition, etc. (cf. [10]), and thus take advantage of the performance improvements for constraint solving enabled by these methods. Our solution suits better the solvers employing such decompositions or the solvers that are recording (at least partially) their proof structures in order to support incremental operation. Although developed independently and using quite differing notations, the *principle* underlying the decomposition of the conflict minimization problems *is the same* for our XC1 and for Junker's QUICKXPLAIN algorithm. After several years of trying to improve constraint solving and dependency recording (cf. [15]), the existence of such simple and general algorithms for minimal conflict computation came for us as a surprising positive result.

One of our main application areas is model-based diagnosis. We do not argue here that one should perform diagnosis by always first computing conflicts and then generating minimal / preferred / etc. diagnoses. Several authors point out that the direct computation of diagnoses can be more efficient (cf. [16, 19, 21, 23]). The ideas from an algorithm such as TREE* (cf. [21]) can be probably easily adapted to a general relational framework such as the one of RCS. One weak point, however, of the available computation techniques that are not based on conflicts is that they basically address static problems. It would be interesting to see if the ideas of the *temporal decomposition* that can be applied for computing minimal conflicts (cf. [14]) can be also applied for the direct computation of diagnoses or interpretations.

Although we discussed here about the computation of *minimal* conflicts, in practice minimality and completeness have to be traded against efficiency. Nevertheless, sometimes the definition of the application (minimisation, compilation, explanation, etc) require a higher degree of completeness that is more important than the computation times.

REFERENCES

[1] Y. El Fattah: An Elimination Algorithm for Model-based Diagnosis. *Dx98*, Cape Cod, USA, pp. 47-54, 1998.

[2] K. Forbus, J. de Kleer: Building Problem Solvers. MIT Press, 1993.

[3] J. de Kleer: An Assumption-based truth maintenance system. *Artificial Intelligence*, 28, pp. 127-162, 1986.

[4] J. de Kleer, B. Williams: Diagnosing Multiple Faults. *Artificial Intelligence*, 32, pp. 97-130, 1987.

[5] R. Dechter: Bucket Elimination: a Unifying Framework for Reasoning. *Artificial Intelligence*, 113, pp. 41 - 85, 1999.

[6] D. J. Goldstone: Controlling inequality reasoning in a TMS-based analog diagnosis system. *9th Nat. Conf. On AI*, pp. 512-517, 1991.

[7] R. Bakker, F. Dikker, F. Tempelman, P. Wognum: Diagnosing and solving over-determined CSP. *Proc. IJCAI-93*, 1993

[8] U. Junker: QUICKXPLAIN: Conflict Detection for Arbitrary Constraint Propagation Algorithms. *IJCAI'01 Workshop on Modelling and Solving Problems with constraints*, pp. 75-82, 2001.

[9] N. Muscettola, P. Nayak, B. Pell, B. Williams: Remote Agent: To boldly go where no AI system has gone before. *Artificial Int.*, 103, pp. 5-47, 1998.

[10] G. Gottlob, N. Leone, F. Scarcello: A comparison of structural CSP decomposition methods. *Artificial Int.*, 124(2), pp. 243-282, 2000.

[11] A. Fleming, G. Friedrich, D. Jannach, M. Stumptner: Consistency-based Diagnosis of Configuration Knowledge Bases. *ECAI-2000*, Berlin, 2000.

[12] J. Mauss, V. May, M. Tatar: Towards Model-based Engineering: Failure Analysis with MDS. *ECAI-2000 Workshop W31*, 2000. http://www.dbai.tuwien.ac.at/event/ecai2000-kbsmbe/papers.html

[13] F. Bouquet, P. Jegou: Solving over-constrained CSP using weighted OBDDs. Proc. Over-Constrained Systems, Lecture Notes in Computer Science, Vol. 1106, Springer, Berlin, 1996.

[14] M. Tatar : Diagnosis with cascading defects. *ECAI-1996*, 1996.

[15] M. Tatar : Model-based failure analysis in engineering – an experience report. *Invited talk at Dx 2001*. Available at request.

[16] J. Amilhastre, H. Fargier, P. Marquis: Consistency restoration and explanations in dynamic CSPs. *Artificial Intelligence* 135, 2002.

[17] G. Katsillis, M. Chantler: Can Dependency-based Diagnosis Cope with Simultaneous Equations? *Dx97*, France, 1997.

[18] D. McAllester: Truth Maintenance. *AAAI-90*, pp. 1109-1116, 1990.

[19] W. Nejdl, B. Giefer: DRUM: Reasoning without conflicts and justifications. *Dx94* , pp. 226-233, New Paltz, NY, 1994.

[20] M. Tatar, P. Dannenmann: Integrating Simulation and model-based Diagnosis into the Life Cycle of Aerospace Systems. *Dx99*, Loch Awe, Scotland, 1999.

[21] M. Stumptner, F. Wotawa: Diagnosing tree-structured systems. *Artificial Intelligence*, 127, pp. 1-29, 2001.

[22] F. Feldkamp, M. Heinrich, K.-D. Meyer-Gramann: SyDeR - System Design for Reusability. *AI-EDAM Special Issue on Configuration Design*. Sept. 1998.

[23] A. Darwiche: Decomposable Negation Normal Form. *Journal of ACM*, July 2001.

A Model Counting Characterization of Diagnoses

T. K. Satish Kumar
Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

Abstract

Given the description of a physical system in one of several forms (a set of constraints, Bayesian network etc.) and a set of observations made, the task of model-based diagnosis is to find a suitable assignment to the modes of behavior of individual components (this notion can also be extended to handle transitions and dynamic systems [Kurien and Nayak, 2000]. Many formalisms have been proposed in the past to characterize diagnoses and systems. These include consistency-based diagnosis, fault models, abduction, combinatorial optimization, Bayesian model selection etc. Different approaches are apparently well suited for different applications and representational forms in which the system description is available. In this paper, we provide a unifying theme behind all these approaches based on the notion of model counting. By doing this, we are able to provide a universal characterization of diagnoses that is independent of the representational form of the system description. We also show how the shortcomings of previous approaches (mostly associated with their inability to reason about different elements of knowledge like probabilities and constraints) are removed in our framework. Finally, we report on the computational tractability of diagnosis-algorithms based on model counting.

1 Introduction

Diagnosis is an important component of autonomy for any intelligent agent. Often, an intelligent agent plans a set of actions to achieve certain goals and because some conditions may be unforeseen, it is important for it to be able to reconfigure its plan depending upon the state in which it is. This state identification problem is essentially a problem of diagnosis. In its simplest form, the problem of diagnosis is to find a suitable assignment to the modes of behavior of individual components in a static system (given some observations made on it). It is possible to handle the case of dynamic systems by treating the transition variables as components (in one sense) [Kurien and Nayak, 2000]. The theory developed in this paper is therefore equally applicable to dynamic systems too (although we omit the discussion due to restrictions on the length of the paper).

Many approaches have been used in the past to characterize diagnoses and systems. Among the most comprehensive pieces of work are [de Kleer and Williams, 1989], [Reiter, 1987], [Struss and Dressler, 1989], [Console *et al.*, 1989], [de Kleer *et al.*, 1992], [Poole, 1994], [Kohlas *et al.*, 1998] and [Lucas, 2001]. The popular characterizations of diagnoses include consistency-based diagnosis, fault models, abduction, combinatorial optimization, and Bayesian model selection. These approaches are however tailored for different applications and representational forms in which the system description is available. They also have one or more shortcomings arising out of their inability to provide for a framework that can incorporate knowledge in different forms like probabilities, constraints etc.

In this paper, we provide a unifying theme behind all these approaches based on the notion of model counting. By doing this, we are able to provide a universal characterization of diagnoses independent of the representational form of the system description. Because model counting bridges the gap between different kinds of knowledge elements, the shortcomings of previous approaches are removed.

2 Background

Before we present our characterization of diagnoses based on model counting, we choose to provide a quick overview of the previous approaches so that we can compare and contrast our approach with them.

Definition *(Diagnosis System)* A *diagnosis system* is a triple $(SD, COMPS, OBS)$ such that:

1. SD is a system description expressed in one of several forms — constraint languages like propositional logic, probabilistic models like Bayesian network etc. SD specifies both component behavior information and component structure information (i.e. the topology of the system).
2. $COMPS$ is a finite set of components of the system. A component $comp_i$ ($1 \leq i \leq |COMPS|$) can behave in one of several, but finite set of modes (M_i). If these modes are not specified explicitly, then we assume two modes — failed ($AB(comp_i)$) and normal ($\neg AB(comp_i)$).
3. OBS is a set of observations expressed as variable values.

Definition The task in a *complete diagnosis call* is to find a "suitable" assignment of modes to all the components in the

system given SD and OBS. The task in a *partial diagnosis* call is to find a suitable assignment of modes to a specified subset S ($S \subseteq COMPS$) of the components in the system given SD and OBS.

Unless stated otherwise, we will use the term "diagnosis" to refer to a complete diagnosis. Later in the paper we will show that the characterization of partial diagnoses is a simple extension of the characterization of complete diagnoses.

Definition *(Candidate)* Given a set of integers $i_1 \cdots i_{|COMPS|}$ (such that for $1 \leq j \leq |COMPS|$, $1 \leq i_j \leq |M_j|$), a candidate $Cand(i_1 \cdots i_{|COMPS|})$ is defined as $Cand(i_1 \cdots i_{|COMPS|}) = (\bigcup_{k=1}^{|COMPS|}(comp_k = M_k(i_k)))$.

Here, $M_u(v)$ denotes the v^{th} element in the set M_u (assumed to be indexed in some way).

Notation When the indices are implicit or arbitrary, we will use the symbol H to denote a candidate or a hypothesis i.e. an assignment of modes to all the components in the system.

Consistency-Based Diagnosis

The task of consistency-based diagnosis can be summarized as follows. Note that the definition of a diagnosis in this framework does not discriminate between single and multiple faults.

Definition *(Consistency-Based Diagnosis)* A candidate H is a diagnosis if and only if $SD \cup OBS \cup H$ is satisfiable.

There are other characterizations of diagnoses under this framework called *partial diagnoses*, *prime diagnoses*, *kernel diagnoses* etc. We will examine these later in the paper.

Fault Models

Consider diagnosing a system consisting of three bulbs B_1, B_2 and B_3 connected in parallel to the same voltage source V under the observations $off(B_1)$, $off(B_2)$ and $on(B_3)$. $AB(V) \wedge AB(B_3)$ is a diagnosis under the consistency-based formalization of diagnosis if we had constraints only of the form $\neg AB(B_3) \wedge \neg AB(V) \rightarrow on(B_3)$. Intuitively however, it does not seem reasonable because B_3 cannot be *on* without V working properly. One way to get around this is to include fault models in the system. These are constraints that explicitly describe the behavior of a component when it is not in its nominal mode (most expected mode of behavior of a component). Such a constraint in this example would be $AB(B_3) \rightarrow off(B_3)$. Diagnosis can become indiscriminate without fault models. It is also easy to see that the consistency-based approach can exploit fault models (when they are specified) to produce more intuitive diagnoses (like only B_1 and B_2 being abnormal).

Diagnosis as Combinatorial Optimization

The technique of using fault models is associated with the problem of being too restrictive. We may not be able to model the case of some strange source of power making B_3 *on* etc. The way out of this is to allow for many modes of behavior for each component of the system. Every component has a set of modes (in which it can behave) with associated models. One of these is the nominal (or normal) mode and the others are fault modes. Each component has the *unknown* fault mode with the empty model. The unknown mode tries to capture the *modeling incompleteness assumption* (obscure modes that we cannot model in the system). Also, each mode has an associated probability that is the prior probability of the component being in that mode. Diagnosis can now be cast as a combinatorial optimization problem of assigning modes of behavior to each component such that it is not only consistent with $SD \cup OBS$, but also maximizes the product of the prior probabilities associated with those modes (assuming independence in the behavior of components).

Definition *(Combinatorial Optimization)* A candidate $H =$ Cand($i_1 \cdots i_{|COMPS|}$) is a diagnosis if and only if $SD \cup H \cup OBS$ is satisfiable and $P(H) = (\Pi_{k=1}^{|COMPS|} P(comp_k = M_k(i_k)))$ is maximized.

Diagnosis as Bayesian Model Selection

Sometimes we have sufficient experience and statistical information associated with the behavior of a system. In such cases, the system description is usually available in the form of a probabilistic model like a Bayesian network. Given some observations made on the system, the problem of diagnosis then becomes a Bayesian model selection problem.

Definition *(Bayesian Model Selection)* A candidate H is a diagnosis (for a probabilistic model of the system, SD) if and only if it maximizes the posterior probability $P(H/SD, OBS)$.

Diagnosis as Abduction

Yet another intuition behind characterizing diagnoses is the idea of explanation. Explanatory diagnoses essentially try to capture the notion of cause and effect in the physics of the system. The observations are asymmetrically divided into inputs (I) and outputs (O) [de Kleer *et al.*, 1992]. The inputs (I) are those observation variables that can be controlled externally.

Definition *(Abductive Diagnosis)*: An abductive diagnosis for (SD, $COMPS$, $OBS = I \cup O$) is a candidate H such that $SD \cup I \cup H$ is satisfiable and $SD \cup I \cup H \rightarrow O$.

3 Probabilities and Model Counting

Before we present our own characterization of diagnoses based on the notion of model counting, we show an interesting relationship between probabilities and model counting (see Figure 1). The model counting problem is the problem of counting the number of solutions to a SAT (satisfiability problem) or a CSP (constraint satisfaction problem).

Definition *(Binary representation of a CPT)*: The *binary representation of a CPT (Conditional Probability Table)* is a table in which all the floating-point entries of the CPT are re-written in a binary form (base 2) up to a precision of P binary digits and the decimal point along with any redundant zeroes to the left of it are removed.

We provide a set of definitions and results relating the probability of a partial assignment A to the number of solutions (under the same partial assignment A) to CSPs composed out of the binary representations of the CPTs (see Figure 1). Basic definitions related to CSPs can be found in [Dechter, 1992].

Definition *(Zero-one-layer of a CPT)* The k^{th} *zero-one-layer of a CPT* is a table of zeroes and ones derived from the k^{th}

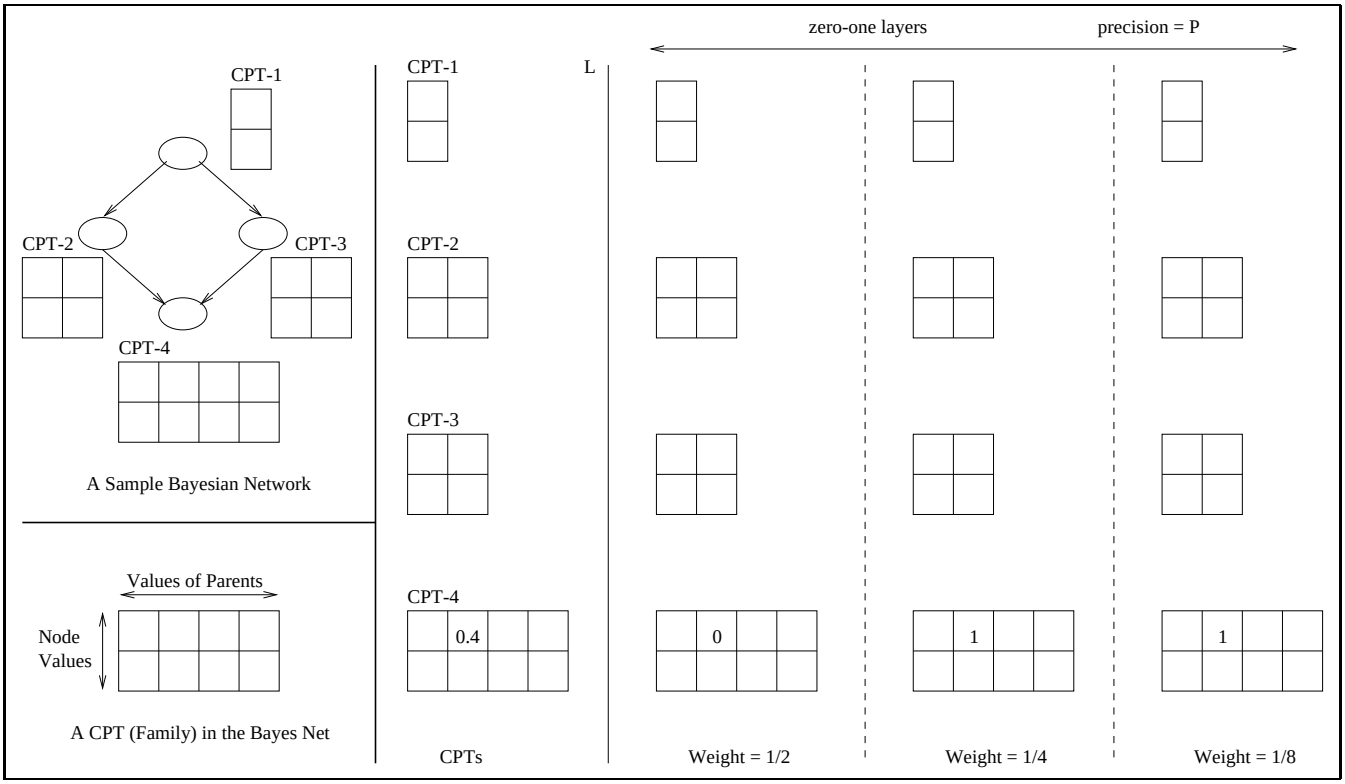


Figure 1: Shows the conditional probability tables (CPTs) of a Bayes net on the left of the vertical line L. On the right of L are the binary representations of these CPTs (example shown for 0.4 in decimal = 0.011 in binary). CPTs correspond to families in the Bayes net and let the number of families be C.

bit position of all the numbers in the binary representation of that CPT.

Definition *(Weight of a zero-one-layer)* The k^{th} zero-one-layer of a CPT is defined to have weight 2^{-k}.

Definition *(CSP Compilation of a CPT)* The k^{th} *CSP compilation of a CPT* is a constraint over the variables of the CPT that is derived from the k^{th} zero-one-layer of the CPT such that zeroes correspond to disallowed tuples and ones correspond to allowed tuples.

Definition *(CSP Compilation of Network)* The $(k_1, k_2 \cdots k_C)$ *CSP compilation of the Network* is the set of constraints $S = \{s_i : s_i \text{ is the } k_i^{th} \text{ CSP compilation of the } i^{th} \text{ CPT}\}$.

Definition *(Weight of a CSP Compilation)* The *weight of a* $(k_1, k_2 \cdots k_C)$ *CSP compilation* of a network is defined to be equal to $2^{-(k_1+k_2 \cdots k_C)}$.

Property There are an exponential number of CSP compilations for a given network. Since each CPT expands into P zero-one-layers and a CSP for the entire network can be compiled by taking any of these P layers for each CPT (there are C CPTs), the total number of CSP compilations possible is P^C.

Notation We will use the notation h_{ij} to mean the j^{th} CSP compilation of the i^{th} CPT. Let A indicate a complete or partial assignment to the variables. If A is an assignment that instantiates all the variables of CPT_i, then we will use the notation $h_{ij}(A)$ to indicate whether or not A satisfies h_{ij}. If A is a complete assignment for all the variables in the network, then all variables for all CPTs are instantiated and we will use the notation $CSP_{(k_1,k_2 \cdots k_C)}(A)$ to indicate whether A satisfies all the constraints h_{ik_i} $(1 \leq i \leq C)$. If A is not a complete assignment for all the variables, then we will use the notation $\#CSP_{(k_1,k_2 \cdots k_C)}(A)$ to indicate the number of solutions to the $(k_1, k_2 \cdots k_C)$ CSP compilation of the network that share the same partial assignment as A.

Theorem 1 The probability of a complete assignment $A = (X_1 = x_1 \cdots X_n = x_n)$ is just the sum of the weights of the different CSP compilations of the network that are satisfied by this complete assignment. That is, $P(A) = \sum_{(k_1,k_2 \cdots k_C)} CSP_{(k_1,k_2 \cdots k_C)}(A) 2^{-(k_1+k_2 \cdots k_C)}$ (for all $1 \leq i \leq C, 1 \leq k_i \leq P$).

Proof Consider the complete assignment $A = (X_1 = x_1 \cdots X_n = x_n)$ for all the variables. The probability of this assignment is equal to the product of the probabilities defined *locally* by each CPT. Now using the fact that the t^{th} bit in the binary representation of this local value has been written out as an allowed or disallowed tuple in the t^{th} CSP compilation of that CPT, we can rewrite the local value for A in a CPT as $\sum_{j=1}^{P} h_{ij}(A) 2^{-j}$. The total probability is then just the product over all local values = $\Pi_{k=1}^{C} \sum_{j=1}^{P} h_{kj}(A) 2^{-j}$. Expanding the product, we see that each term is essentially of the form $\sum_{(k_1,k_2 \cdots k_C)} 2^{-(k_1+k_2 \cdots k_C)} \Pi_{j=1}^{C} h_{jk_j}(A) = \sum_{(k_1,k_2 \cdots k_C)} 2^{-(k_1+k_2 \cdots k_C)} CSP_{(k_1,k_2 \cdots k_C)}(A)$.

Theorem 2 *(Model Counting)* The marginalized probability of a partial assignment A to a set of variables $S \subseteq V$ is equal to the product of the weight and the number of solutions (under the same partial assignment A) summed over all CSP compilations of the network. That is, $P(A) = \sum_{(k_1,k_2\cdots k_C)} \#CSP_{(k_1,k_2\cdots k_C)}(A)2^{-(k_1+k_2\cdots k_C)}$ (for all $1 \leq i \leq C$, $1 \leq k_i \leq P$).

Proof From the previous theorem, we know that the probability of a complete assignment B is $\sum_{(k_1,k_2\cdots k_C)} CSP_{(k_1,k_2\cdots k_C)}(B)2^{-(k_1+k_2\cdots k_C)}$ (for all $1 \leq i \leq C$, $1 \leq k_i \leq P$). Now, the marginalized probability of a partial assignment A is just the sum of the probabilities of all complete assignments B that agree with A on the assignment to variables in S. That is, $P(A) = \sum_B P(B)(B(S) = A)$. Using the result of the previous theorem to expand $P(B)$, we have $P(A) = \sum_B \sum_{(k_1,k_2\cdots k_C)} CSP_{(k_1,k_2\cdots k_C)}(B)2^{-(k_1+k_2\cdots k_C)}(B(S) = A)$. Switching the two summations and noting that $\sum_B CSP_{(k_1,k_2\cdots k_C)}(B)(B(S) = A)$ is the same as $\sum_{(k_1,k_2\cdots k_C)} \#CSP_{(k_1,k_2\cdots k_C)}(A)$, we get that $P(A) = \sum_{(k_1,k_2\cdots k_C)} \#CSP_{(k_1,k_2\cdots k_C)}(A)2^{-(k_1+k_2\cdots k_C)}$.

3.1 Probability-Equivalents and Incorporation of Probabilities

Often, we are given information in many forms. Probabilities are natural information elements when there is an element of statistical experience that we want to exploit. In other cases, constraints may be the most appropriate to use. The general idea in our framework is to use probabilities when we explicitly have them and to use model counting otherwise. We will use $\#(S_1, S_2 \cdots)$ to mean the number of consistent models to $(S_1 \cup S_2 \cdots)$ (with respect to the uninstantiated free variables in SD). Theorems 1 and 2 establish that model counting is a weaker form of probabilities and that probabilities provide only *precision information* over model counting. Therefore, it is natural for us to use probabilities (to describe events) when we have them explicitly, and to use model counting otherwise. For any event E, we use the expressions $\frac{\#(SD,E)}{\#(SD)}$ and $P(E)$ almost equivalently — except that we use the former when we do not know $P(E)$ explicitly. This framework allows us to reason about both probabilities and constraints.

Definition *(Probability Equivalents)* The *probability equivalent* of $\#(SD, E)$ for any event E is defined to be $P(E)\#(SD)$ when $P(E)$ is given explicitly.

4 Diagnosis as Model Counting

In this section, we characterize diagnoses based on model counting. We will then show how all the previous approaches are captured under this formalization. For the first part of the discussion we will consider only complete diagnoses (an assignment of modes for all the components).

Definition *(Model Counting Characterization)* A *diagnosis* is a candidate H that maximizes the number of consistent models to $SD \cup OBS \cup H$ using probability equivalents wherever necessary.

Notation We will use $M(H)$ to denote $\#(SD, OBS, H)$ (the number of consistent models to $SD \cup OBS \cup H$) when SD and OBS follow from context.

Theorem 3 *(Capturing Consistency-Based Diagnosis)* Consistency-Based diagnosis is looking for a hypothesis H for which $M(H)$ is non-zero.

Proof By definition, consistency-based diagnosis chooses H such that $SD \cup OBS \cup H$ is consistent. In other words, there exists at least one satisfying assignment for $SD \cup OBS \cup H$. Clearly, this is equivalent to saying that $M(H)$ is non-zero.

Theorem 4 *(Capturing Abduction)* Abduction chooses a hypothesis H that maximizes $M(H)$ assuming uniformity in prior probabilities $P(H)$.

Proof The maximum value of $\#(SD, OBS = I \cup O, H)$ is $\#(SD, H, I)$ and this happens when $H \cup SD \cup I \rightarrow O$. Given that the input variables are controlled externally, we know that $\#(SD, H) = N(I)\#(SD, H, I)$. Here, $N(I)$ is a constant that measures the number of different values for the input variables. Since $\#(SD, H)$ is equivalent to $P(H)\#(SD)$ which we assumed to be a constant for all H, maximizing $\#(SD, OBS, H)$ is equivalent to finding a hypothesis H for which $I \rightarrow O$ (under SD). The fact that abduction requires H to be consistent is also captured, because if H is inconsistent, then $M(H) = 0$ and clearly $M(H)$ will not be maximized.

Theorem 5 *(Capturing Bayesian Model Selection)* Bayesian model selection chooses a hypothesis H such that it maximizes the probability equivalent of $M(H)$.

Proof The probability equivalent of $M(H) = \#(SD, OBS, H)$ is $P(OBS, H)$. Clearly, if we are maximizing $P(OBS, H)$ then we are maximizing $P(H/OBS)P(OBS)$. Since $P(OBS)$ is independent of H, it is equivalent to maximizing $P(H/OBS)$ which is exactly what Bayesian model selection does.

Theorem 6 *(Capturing Combinatorial Optimization)* Combinatorial optimization is looking for a hypothesis H which maximizes $P(H)$ under the condition that $M(H)$ is non-zero.

Proof As noted earlier, H is consistent with $SD \cup OBS$ if and only if $M(H)$ is non-zero. We also know that combinatorial optimization is looking for a consistent H which maximizes $P(H)$. The theorem follows as a simple consequence of the above two statements. Basically, combinatorial optimization maximizes only the prior probabilities of hypotheses (instead of maximizing the equivalent of the posterior probabilities) unless they are obviously ruled out by being inconsistent.

4.1 Consequences (Removing Previous Shortcomings)

We now show the consequences of formalizing diagnosis as model counting. In particular, we identify problems with previous approaches and show how model counting removes all of them.

Problems with Consistency-Based Diagnosis

One of the problems with consistency-based diagnosis is that it allows for non-intuitive hypotheses as diagnoses. It provides only for a necessary but not a sufficient condition on the hypotheses that can be qualified as diagnoses. By itself, it is of little value unless we use an elaborate set of fault models

to remove non-intuitive hypotheses that could otherwise be consistent. Model counting removes these problems because of its ability to merge and incorporate the notions of both consistency and probabilities. In one sense, one can think of model counting as giving us a measure of the degree to which a hypothesis is consistent with SD and OBS. Some of these problems are alternatively addressed in [Kohlas *et al.*, 1998] and [Lucas, 2001].

Problems with Fault Models

The problem with fault-models is that of over-restriction (as explained at the beginning of the paper). We need to be able to reason not only about constraints relating SD and OBS, but also about any other kind of information we may have in the form of probabilities etc. The over-restriction problem can be removed by introducing probabilities. These probabilities can then be used in the unified framework of model counting.

Problems with Abduction

Like the consistency-based approaches, explanatory diagnoses are also unable to incorporate and reason about probabilities. Yet another problem with abduction is that it assumes we have completely modeled all cause-effect relationships in our system. This contradicts our modeling incompleteness assumption and is an unnecessary restriction on SD. Model counting removes this problem in a way very similar to how probabilities were used to deal with the modeling incompleteness assumption. Alternate treatments for these problems can be found in [Poole, 1994] (which links abduction with probabilistic reasoning) and [Console *et al.*, 1989] (which addresses the modeling incompleteness assumption).

Problems with Diagnosis as Bayesian Model Selection

Bayesian model selection agrees with our characterization of diagnoses — but the only problem it poses is that it requires SD to be in the form of a Bayesian network with known probabilities. Modeling a physical system as a Bayesian network is in many cases a non-intuitive thing to do. This is especially so when certain probability terms are hard to get. Parts of the system may be better expressed in the form of constraints or automata. In such cases, Bayesian model selection does not extend in a natural way and model counting is the right substitute (because it is defined under all frameworks).

Problems with Diagnosis as Combinatorial Optimization

One problem associated with casting diagnosis as a combinatorial optimization problem is that of being unable to give explanatory diagnoses a preference over the rest. Clearly, we would like to prefer hypotheses that not only maximize the prior probability $P(H)$ but that are also explanatory rather than just being consistent with $SD \cup OBS$. One way to incorporate this preference is to find all consistent hypotheses that maximize $P(H)$ and to pick an explanatory one among them. The question that arises then is how we would compare two hypotheses one of which is explanatory and the other just consistent (but not explanatory), with the latter having a slightly better prior probability. This question is left unanswered under the combinatorial optimization formulation of diagnoses. In the model counting framework however, it is easy to see that we really have to maximize $P(H)\frac{\#(SD,OBS,H)}{\#(SD,H)}$. The second factor is maximized for explanatory diagnosis — but this is as much as the preference we attach for them.

Another problem with the combinatorial optimization formulation is that probabilities are restricted to only behavior modes of components and only these prior probabilities are maximized. There is no framework to reason about probabilistic information connected with observation variables.

5 Partial Diagnoses

Sometimes, we are interested in finding a suitable assignment of modes to a specified subset S of the components $COMPS$ rather than for all components. We argue that our characterization of diagnoses under the model counting framework remains unchanged.

Definition *(Candidate)* Given a set of integer tuples $(k_1, i_{k_1}) \cdots (k_n, i_{k_n})$ such that for $1 \leq j \leq n \leq |COMPS|$, $1 \leq i_{k_j} \leq |M_j|$, a candidate $Cand((k_1, i_{k_1}) \cdots (k_n, i_{k_n}))$ is defined as $Cand((k_1, i_{k_1}) \cdots (k_n, i_{k_n})) = (\bigcup_{g=1}^{n}(comp_g = M_g(i_{k_g})))$.

Notation When the indices are implicit or arbitrary, we will use the notation J_S to denote a candidate or a hypothesis i.e. a set of mode assignments to all the components in $S \subseteq COMPS$.

Definition *(Model Counting Characterization)* A *partial diagnosis* for $S \subseteq COMPS$ is an assignment of modes J_S to the components in S that maximizes $\#(SD, OBS, J_S)$ using probability equivalents wherever necessary.

It is now not hard to verify that all previous approaches are captured in a way very similar to that for complete diagnoses. This is essentially a consequence of the theorem that relates the number of consistent models for (SD, OBS, J_S) to the marginalized probability of J_S (Theorem 2). Instead of presenting the proofs again (and making repetitive arguments), we choose to allude to another set of characterizations mostly associated with consistency-based diagnosis. These are the notions of *partial* (a different characterization in consistency-based diagnosis), *kernel* and *prime* diagnoses. These notions have the same kind of drawbacks associated with the general consistency-based framework [de Kleer *et al.*, 1992] and our investigation into these notions is just in the spirit of understanding their relationship to model counting.

Definition An $AB-$literal is $AB(c)$ or $\neg AB(c)$ for some component c in $COMPS$. An $AB-$clause is a disjunction of $AB-$literals containing no complementary pair of $AB-$literals.

Definition A *conflict* of $(SD, COMPS, OBS)$ is an $AB-$clause entailed by $SD \cup OBS$. A *minimal conflict* of $(SD, COMPS, OBS)$ is a conflict no proper sub-clause of which is a conflict of $(SD, COMPS, OBS)$.

Definition *(Consistency-Based Characterization)* The *partial diagnoses* of $(SD, COMPS, OBS)$ are the implicants of the minimal conflicts of $(SD, COMPS, OBS)$.

Theorem 7 A *partial diagnosis* in the consistency-based framework identifying an implicant T of the minimal conflicts of $SD \cup OBS$, is also a *partial diagnosis* in the model-counting framework maximizing $M(J_S) = \#(SD, OBS, J_S)$ for $S =$ variables of the implicant T, but

with free variables limited to *abnormality* (AB) variables.

Proof The implicant T fixes an assignment for the components in S but leaves $COMPS \setminus S$ unassigned. Let the set of minimal conflicts of $SD \cup OBS$ be π. Let $\#_{AB}(E)$ denote the number of consistent models of E restricted to free variables being from the uninstantiated $AB-$variables. Since T is an implicant of π, all models of T (restricted to $AB-$variables) also satisfy π and are hence consistent with $SD \cup OBS$. This makes $\#_{AB}(SD, OBS, T) = \#_{AB}(T)$. In general, since $\#_{AB}(SD, OBS, T)$ is upper bounded by $\#_{AB}(T)$, the truth of the theorem follows.

Definition *(Consistency-based Characterization)* A *kernel diagnosis* identifies the prime implicants of the minimal conflicts of $SD \cup OBS$.

Without a detailed discussion (due to lack of space), we claim that this notion is related to yet another task in diagnosis — that of "representing" complete diagnoses. This task is orthogonal to "characterizing" them [Kumar, 2002]. There are other notions of diagnosis called *prime diagnoses, irredundant diagnoses* etc. [de Kleer *et al.*, 1992] arising mostly out of the task of "representation" and all of which are captured in one or the other way by the model counting framework (which we omit in this paper).

6 Related Work on Characterizing Diagnoses and Model Counting

Related work in trying to unify model-based and probabilistic approaches can be found in [Poole, 1994], [Kohlas *et al.*, 1998], [Lucas, 1998] and [Lucas, 2001]. [Poole, 1994] links abductive reasoning and Bayesian networks and general diagnostic reasoning systems with assumption-based reasoning. [Kohlas *et al.*, 1998] shows how to take results obtained by consistency based reasoning systems into account when computing a posterior probability distribution conditioned on the observations (the independence assumptions are lifted in [Lucas, 2001]). [Lucas, 1998] gives a semantic analysis of different diagnosis systems using basic set theory. The issue of the *modeling incompleteness assumption* is referred to in [Console *et al.*, 1989].

Diagnosis algorithms based on model counting have not yet been developed. However, the problem of model counting itself has been extensively dealt with. Although this problem is $\#P$-complete, there are a variety of techniques that have been used to make it feasible in practice (including approximate counting algorithms running in polynomial time, structure-based techniques etc.). Model counting for a SAT instance in DNF (disjunctive normal form) is simpler than it is for CNF (conjunctive normal form). For DNF, there is a fully polynomial randomized approximation scheme (FPRAS) to estimate the number of solutions [Karp *et al.*, 1989]. CDP and DDP are two model-counting algorithms for SAT instances in CNF [Bayard and Pehoushek, 2000]. A version of RELSAT has also been used to do model counting on SAT instances in CNF. If a propositional theory is in a special form called the smooth, deterministic, decomposable, negation normal form (sd-DNNF), then model counting can be made tractable and incremental [Darwiche, 2001].

7 Summary and Future Work

In this paper, we provided a unifying characterization of diagnoses based on the idea of model counting. In the process, we compared and contrasted our formalization with the previous approaches — in many cases, removing the problems associated with them. Because model counting bridges the gap between probabilities and constraints and is well-defined for many representational forms of information available about the system, we believe that the model counting characterization of diagnoses is useful and general in the sense of not imposing any restrictions on the representational form of the system description.

As for our future work, we are in the process of investigating and developing computationally tractable algorithms based on the model counting characterization of diagnoses. Advances in model counting algorithms (approximate counting, structure-based methods etc.) seem to be encouraging towards this goal. We are also working on variants of the diagnosis problem (e.g. when we are interested in a set of candidate hypotheses rather than just one).

References

[Bayard and Pehoushek, 2000] Bayard R. J. and Pehoushek J. D. Counting Models using Connected Components. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*.

[Console *et al.*, 1989] Console L., Theseider D., and Torasso P. A Theory of Diagnosis for Incomplete Causal Models. *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Los Angeles, USA (1989) 1311-1317.*

[Console and Torasso, 1991] Console L. and Torasso P. A Spectrum of Logical Definitions of Model-Based Diagnosis. *Computational Intelligence 7(3): 133-141.*

[Darwiche, 2001] Darwiche A. On the Tractable Counting of Theory Models and its Applications to Belief Revision and Truth Maintenance. *To appear in Journal of Applied Non-Classical Logics.*

[Dechter, 1992] Dechter R. Constraint Networks. *Encyclopedia of Artificial Intelligence, second edition, Wiley and Sons, Pages: 276-285, 1992.*

[de Kleer, 1986] de Kleer J. An Assumption Based TMS. *Artificial Intelligence 28 (1986).*

[de Kleer *et al.*, 1992] de Kleer J., Mackworth A. K., and Reiter R. Characterizing Diagnoses and Systems. *Artificial Intelligence 56 (1992) 197-222.*

[de Kleer and Williams, 1987] de Kleer J. and Williams B. C. Diagnosing Multiple Faults. *Artificial Intelligence, 32:100-117, 1987.*

[de Kleer and Williams, 1989] de Kleer J. and Williams B. C. Diagnosis with Behavioral Modes. *In Proceedings of IJCAI'89. Pages: 104-109.*

[Forbus and de Kleer, 1992] Forbus K. D. and de Kleer J. Building Problem Solvers. *MIT Press, Cambridge, MA, 1992.*

[Hamscher *et al.*, 1992] Hamscher W., Console L., and de Kleer J. Readings in Model-Based Diagnosis. *Morgan Kaufmann, 1992.*

[Karp *et al.*, 1989] Karp R., Luby M., and Madras N. Monte-Carlo Approximation Algorithms for Enumeration Problems. *Journal of Algorithms 10 429-448. 1989.*

[Kohlas *et al.*, 1998] Kohlas J., Anrig B., Haenni R., and Monney P. A. Model-Based Diagnosis and Probabilistic Assumption-Based Reasoning. *Artificial Intelligence, 104 (1998) 71-106.*

[Kumar, 2001] Kumar T. K. S. QCBFS: Leveraging Qualitative Knowledge in Simulation-Based Diagnosis. *Proceedings of the Fifteenth International Workshop on Qualitative Reasoning (QR'01).*

[Kumar, 2002] Kumar T. K. S. An Information-Theoretic Characterization of Abstraction in Diagnosis and Hypothesis Selection. *Proceedings of the Fifth International Symposium on Abstraction, Reformulation and Approximation (SARA 2002).*

[Kurien and Nayak, 2000] Kurien J. and Nayak P. P. Back to the Future for Consistency-Based Trajectory Tracking. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00).*

[Lucas, 1998] Lucas P. J. F. Analysis of Notions of Diagnosis. *Artificial Intelligence, 105(1-2) (1998) 293-341*.

[Lucas, 2001] Lucas P. J. F. Bayesian Model-Based Diagnosis. *International Journal of Approximate Reasoning, 27 (2001) 99-119.*

[McIlraith, 1998] McIlraith S. Explanatory Diagnosis: Conjecturing Actions to Explain Observations. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98).*

[Mosterman and Biswas, 1999] Mosterman P. J. and Biswas G. Diagnosis of Continuous Valued Systems in Transient Operating Regions. *IEEE Transactions on Systems, Man, and Cybernetics, 1999. Vol. 29, no. 6, pp. 554-565, 1999.*

[Nayak and Williams, 1997] Nayak P. P. and Williams B. C. Fast Context Switching in Real-time Propositional Reasoning. *In Proceedings of AAAI-97.*

[Poole, 1990] Poole D. A Methodology for Using a Default and Abductive Reasoning System. *International Journal of Intelligent Systems 5(5) (1990) 521-548.*

[Poole, 1993] Poole D. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence, 64(1) (1993) 81-129.*

[Poole, 1994] Poole D. Representing Diagnosis Knowledge. *Annals of Mathematics and Artificial Intelligence 11 (1994) 33-50.*

[Raiman, 1989] Raiman O. Diagnosis as a Trial: The Alibi Principle. *IBM Scientific Center (1989).*

[Reiter, 1987] Reiter R. A Theory of Diagnosis from First Principles. *Artificial Intelligence 32 (1987) 57-95.*

[Shanahan, 1993] Shanahan M. Explanation in the Situation Calculus. *In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), 160-165.*

[Struss, 1988] Struss P. Extensions to ATMS-based Diagnosis. *In J. S. Gero (ed.), Artificial Intelligence in Engineering: Diagnosis and Learning, Southampton, 1988.*

[Struss and Dressler, 1989] Struss P. and Dressler O. "Physical Negation" - Integrating Fault Models into the General Diagnosis Engine. *In Proceedings of IJCAI-89. Pages: 1318-1323.*

[Williams and Nayak, 1996] Williams B. C. and Nayak P. P. A Model-Based Approach to Reactive Self-Configuring Systems. *In Proceedings of AAAI-96. Pages: 971-978.*

Computing Minimal Hitting Sets with Genetic Algorithm

Lin Li[1, 2] and **Jiang Yunfei**[1]

Abstract. A set S that has a non-empty intersection with every set in a collection of sets *C* is called a hitting set of *C*. If no element can be removed from *S* without violating the hitting set property, *S* is considered to be minimal. Several interesting problems can be partly formulated as ones that a minimal hitting set or more ones have to be found. Many of these problems are required for proper solutions, but sometimes the approximate solutions are enough. A genetic algorithm and advantaged algorithms were devised for computing minimal hitting sets. An improvement makes them get most minimal hitting sets efficiently. Furthermore, they are smaller, i.e. fewer rules.

1 INTRODUCTION

A lot of theoretical and practical problems, e.g., [1~8], can be partly reduced to an instance of the minimal hitting set or one of its relatives, such as the minimum set cover problem, model-based diagnosis [1~5,7~8], and teachers and courses problem.

Normally speaking, it is a problem of selecting a minimal set (e.g., of teachers) that has a non-empty intersection with each set (e.g., list of courses), That is to say, there is, at least, one teacher who can teach any courses, This is a formulation of the minimal hitting set problem, which, in general, is NP-hard [6].

Generally, there are a number of hitting sets, but sometimes we only need one or some of them. There are some algorithms [1~8] for computing all of the minimal hitting sets, the space and time efficiency are not ideal. We present a novel method based on the Genetic Algorithm (in short GA here) for calculating minimal hitting sets.

Definition 1. (Hitting sets)

Given a collection $C=\{S_i \mid i \square \mathbb{N}\}$ of sets of elements from some universe *U*, a hitting set is a set $S \subseteq U$ such that $S \cap S_i \neq \varnothing$, for all *i*, i.e., a set which contains, at least, one element from all sets in *C*. Let HS(*C*) denote the collection of all hitting subsets in HS(*C*). These are called the minimal hitting sets of *C*.

We introduce a minimizing operator μ [5], MHS(*C*)= μ (HS(*C*)). We will use μ to get minimal conflict(/hitting) sets from conflict(/hitting) sets.

Determining a minimal cardinality element of MHS(*C*) is called the minimal hitting set problem.

Example 1. Model-based diagnosis [1], as shown in Figure 1. Suppose conflict sets are {M1, M2, A1}, {M1, A1, A2, M3}. The minimal hitting sets (diagnosis) are {M1}, {A1}, {M2, A2}, {M2, M3}.

{M1}, {A1} are of minimal cardinality.

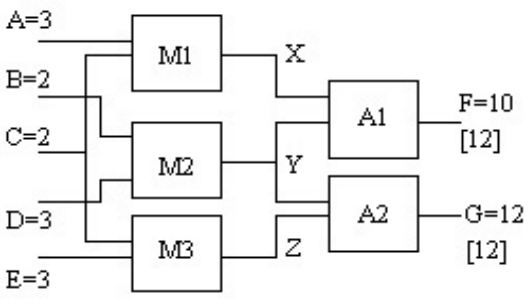


Figure 1 A simple circuit with 3 multipliers and 2 adders.

A minimal cardinality hitting set is a minimal hitting set of minimal cardinality.

In case of large sets of conflicts, the computation of the hitting sets will result both time and space consumption. Shown in Figure 2.

There are about millions of components, For example, in vehicles, computer systems, power plants, aircrafts, etc,. Therefore, we developed a novel efficient GA to compute minimal hitting sets. When the scale of conflicting sets is getting large, the GA method can still be used for computing the minimal hitting sets in a very short time.

[1] Sun Yat-sen university, Guangzhou, China.
[2] Jinan university, Guangzhou, China. email:linlionline@21cn.com.

2 GENETIC ALGORITHM

Genetic algorithm is a heuristic for the function optimization, where the extreme of the function (i.e., minimal or maximal) cannot be analytically established. A population of potential solution is refined iteratively by employing a strategy inspired by Darwinist evolution or natural selection. Genetic algorithms promote "survival of the fittest". This type of heuristic has been applied in many different fields, including construction of neural networks and multi-disorder diagnosis.

For the minimal hitting set problem, a straightforward choice of population is a set P of elements from 2^U, encoded as bit-vectors, where each bit indicates the presence of a particular element in the set.

Example 2. (Teacher and course problem) Let C denote a set cluster containing,

S_1={1, 2, 3, 4}, S_2={1, 2, 4}, S_3={1, 2}, S_4={2, 3}, S_5={4}.

It means that there are 5 courses {S_1, S_2, S_3, S_4, S_5 } and 4 teachers 1, 2, 3, 4. Teachers 1, 2, 3 and 4 can teach course S_1, teachers 1, 2, 4 can teach course S_2, … , teacher 4 can teach course S_5. We want to find the least teachers who can teach all of the 5 courses. This is a minimal hitting sets problem, and the minimal hitting sets are: H_1={1, 3, 4}, H_2={2, 4}.

We use bi-vectors to represent the sets and their hitting sets, these bi-vectors are called "chromosomes", each bit is called "gene", and all of the "chromosomes" are called "population".

If we use chromosome to represent the sets, they are represented as follow:

S_1={1, 1, 1, 1}, S_2={1, 1, 0, 1}, S_3={1, 1, 0, 0}, S_4={0, 1, 1, 0}, S_5={0, 0, 0, 1}.

The hitting sets are: H_1={1, 0, 1, 1}, H_2={0, 1, 0, 1}.

Here, $|S_i| \le |\square S_j|$, $|H_i| \le |\square S_j|$, so, the length of chromosomes equals to $|\square S_j|$.

Genetic operations include: "crossover", "mutation", "inversion", "selection" and "obtain".

Suppose that minimal conflict sets cluster is C={S_1, S_2, … , S_n}, $n=|\square S_k|$.

"Crossover" operator. Suppose that S_1={s_{11}, s_{12}, … , s_{1n}}, S_2={s_{21}, s_{22}, … , s_{2n}}, are two chromosomes, select that a random integer number $0<r<n$, S_3, S_4 is offspring of crossover(S_1, S_2),

S_3={s_i | if $i \le r$, $s_i \square S_1$, else $s_i \square S_2$},

S_4={s_i | if $i \le r$, $s_i \square S_2$, else $s_i \square S_1$}.

"Mutation" operator. Suppose that a chromosome S_1={s_{11}, s_{12}, … , s_{1n}}, selecting a random integer number $0<r \le n$, S_3 is mutation of S_1,

S_3={s_i | if $i \ne r$, then $s_i=s_{1i}$, else $s_i=1-s_{1i}$}.

"Inversion" operator. Suppose that chromosome S_1={s_{11}, s_{12}, … , s_{1r} , $s_{1, r+1}$, … , $s_{1, r+l}$, $s_{1,r+l+1}$, … , s_{1n} }, r, l are random numbers, S_2 is the inversion of S_1.

S_2={s_{11}, s_{12}, … , s_{1r} , $s_{1, r+l}$, … , $s_{1, r+1}$, $s_{1,r+l+1}$, … , s_{1n} }.

"Selection" operator. Suppose that there are m sets, we select [m/2] sets and eliminate other sets, the sets we selected are both "fitness" and "minimal", i.e. first, they intersect more sets than the other, and second, their cardinality is smaller.

"Obtain" operator. Suppose that there is a singleton set in the set cluster, then all hitting sets must hits this set, i.e. the gene stands for this set must be always kept as "1" , we refer to this operator as "obtain":

"Obtain" operator has no any influence on the result, it can improve the efficiency, such as a giraffe obtains "long neck". So they can be competed under the " long neck" condition.

Genetic algorithm.

1. InitializePopulation: Obtain $k*|C|*|\square S_i|$ population randomly, each chromosome is an n-length array, k is a const coefficient.

2. Testing if one of the stopping criteria (time, fitness, etc) holds. If it is yes, the procedure can be stopped, here,100 generations are gotten

3. Selection: Selecting one of chromosome; testing its fitness, here, being the number of sets it hits. Keeping the most fitness ones and deleting the bad ones.

4. Applying the genetic operator: such as "crossover", "mutation", "inversion" and "obtain" to the selected parents to form offspring.

5. Recombining the offspring and current population to form a new population with "selection" operator.

6. Repeating steps 2-5.

Also, we can use Genetic Algorithm to compute MINIMAL hitting sets from hitting sets.

In step 3. If we get hitting sets, we can undergo mutation operator just to change s_r from "1" to "0" in order to get its offspring, else, we undergo mutation operator just to change s_r from "0" to "1" in order to get its offspring. In the next selection operator we will go on keeping hitting sets because they are more fitting.

In the end, we will get 4 sets as follow:

1. Minimal hitting sets;

2. Both minimal hitting sets and their super-hitting sets; we will use operator μ to delete the super-hitting sets;

3. Hitting sets, but not minimal, their sub-hitting sets are not gotten;

4. No hitting sets, these sets will be deleted by "selection" operator.

But, in fact, the situation 3 is never gotten by GA test program.

We can get about 95 percent minimal hitting sets with GA. (shown in Figure 2)

3 COMPARISON

We have written a program to compare among HS-tree, BHS-tree [8] and GA; the result is shown in Figure 3 and Figure 4. The elements of every conflict sets are between 1 and 20.

In general, GA can get more than 95 percent minimal hitting sets in 100 generations, when the set cluster is big, then the HS-tree and BHS-tree can not run because of "Out of memory", but, GA can get almost all minimal hitting sets efficiently.

The space complexity of HS-tree is about $O(m^n)$, m is the average of $|S_i|$, n is $|C|$, That of BHS-tree is about $O(2^{|\cup S_i|})$,

that of GA is about $O(n|\sum S_i|)$. So the efficiency of GA is better than that of HS-tree and BHS-tree.

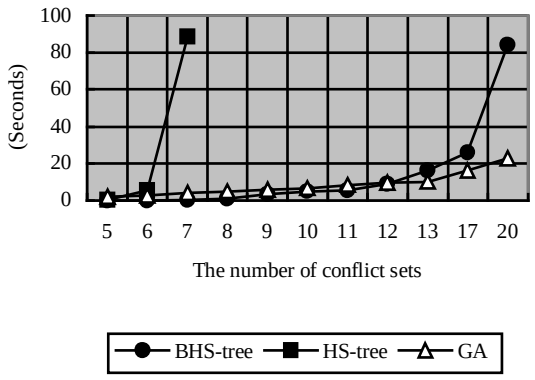


Figure 2 Running time among BHS-tree, HS-tree and GA. (CPU-PII 667, 128M main memory, C++, Windows'98)

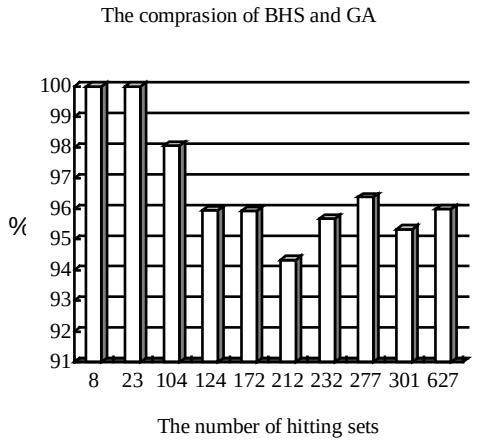


Figure 3 The hitting sets number and the percentage of GA gets.

4 CONCLUSIONS

In this paper, we have improved,

1. When the conflict sets scale gets big, This GA algorithm may get most of minimal hitting sets in a relative short time and small memory, but the other algorithm can't get the hitting sets because of "out of memory".

2. The GA algorithm can also get MINIMAL hitting sets. If a chromosome is not a hitting set, and the "mutation" operator just changes a random gene from "0" into "1", else change a random gene form "1" into "0" so that we can get minimal hitting set.

Example 3. (Continue to Example 2)

If we get H_3={1, 1, 0, 1} and know that it is a hitting set, then we undergo "mutation" operator to it, however, we only change "1" into "0" here.

H_3={1, 1, 0, 1}→{0, 1, 0, 1}, (minimal hitting set)
→{1, 0, 0, 1}, (no hitting set)
→{1, 1, 0, 0}. {no hitting set}

Underlined genes stand for "mutation" from parent genes.

3. Although this algorithm can't get all of the minimal hitting sets, but after we replace or repair these components we have computed, we can do next diagnosis step by step. The next research is GA used in choice of a repair/replace action on the set of suspects or choice of a next measurement.

This GA can be used in many other fields, e.g. a librarian can decide what kind of journals referred by researchers will be purchased under lack of funds. [6, pp124].

ACKNOWLEDGEMENTS

We are grateful to the referees for their comments, which helped us improve this paper.

REFERENCES

[1] R. Reiter, A theory of diagnosis from first principles. Artificial Intelligence, 1987, 32(1): 57~96.

[2] Greiner R, Smith B. A, Wilkerson R. W, A correction to the algorithm in Reiter's theory of diagnosis (research note). Artificial Intelligence, 1989, 41(1): 79~88.

[3] J. de Kleer J., A. K. Mackworth, R. Reiter, Characterizing diagnoses and systems. Artificial Intelligence, 1992, (56): 197~222.

[4] J. A. Reggia, D.S. Nau, P.Y. Wang, Diagnostic expert systems based on a set covering model, International Journal of Man-Machine Studies. 1983, (19): 437~460.

[5] Rolf Haenni, Generating diagnosis from conflict sets. 1997. http://www.aaai.org/

[6] Staal Vinterbo, Aleksander Ohrn, Minimal approximate hitting sets and rule templates, International Journal of Approximate Reasoning 2000, (25): 123~143

[7] Franz Wotawa, A variant of Reiter's hitting set algorithm. Information Processing Letters. 2001, (79): 45~51.

[8] Jiang Yunfei, Lin Li, Computing minimal hitting set from first principles with BHS-tree. Journal of software. Spring 2002. (Coming soon, In Chinese).

[9] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, Hitting sets derandomize BPP, In: Automata, languages and programming, number 1099, Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 357~368.

[10] J.A. Reggia, D.S. Nau, P.Y. Wang, Diagnostic expert systems based on a set covering model, International Journal of Man-Machine Studies 19 (1983) 437~460.

[11] S. Vinterbo, L. Ohno-Machado, A genetic algorithm approach to multidisorder diagnosis, Artificial Intelligence in Medicine, 18 (2) (2000) 117~132.
[12] J.A. Swets, R.M. Pickett, Evaluation of Diagnostic Systems. Methods from Signal Detection Theory, Academic Press, New York, 1982.
[13] F.M. Brown, Boolean Reasoning, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.
[14] J. Wroblewski, Finding minimal reducts using genetic algorithms, in: Proceedings of the Second Annual Join Conference on Information Sciences, October 1995, pp. 186~189.
[15] M. Mitchell, An Introduction to Genetic Algorithms, MIT press, Cambridge, 1996.
[16] D. whitley. An overview of evolutionary algorithm: practical issues and common pitfalls, Information and Software Technology. 2001(43): 817~831.
[17] Benjamin Han, Shie-Jue Lee. A genetic algorithm approach to measurement prescription in fault diagnosis. Information Science. 1999(120):223~237.

Model-Based Diagnosis for Information Survivability [1]

Howard Shrobe
NE43-839
Artificial Intelligence Laboratory
Massacusetts Institute of Technology
Cambridge, MA 02139
hes@ai.mit.edu

Abstract.

The Infrastructure of modern society is controlled by software systems that are vulnerable to attack. Successful attacks on these systems can lead to catastrophic results; the survivability of such information systems in the face of attacks is therefore an area of extreme importance to society. This paper presents model-based techniques for the diagnosis of potentially compromised software systems; these techniques can be used to aid the self-diagnosis and recovery from failure of critical software systems. It introduces *Information Survivability* as a new domain of application for model-baesed diagnosis and it presents new modeling and reasoning techniques relevant to the domain. In particular: 1) We develop techniques for the diagnosis of compromised *software* systems (previous work on model-base diagnosis has been primarily cconcerned with physical components); 2) We develop methods for dealing with model-based diagnosis as a mixture of symbolic and Bayesian inference; 3) We develop techniques for dealing with common-mode failures; 4) We develop unified representational techniques for reasoning about information attacks, the vulnerabilities and compromises of computational resources, and the observed behavior of computations; 5) We highlght additional information that should be part of the goal of model-based diagnosis.

1 Background and Motivation

The infrastructure of modern society is controlled by computational systems that are vulnerabile to information attacks. The system and application software of these systems possess vulnerabilities that enable attacks capable of compromising the resources used by the software systems. A skillful attack could lead to consequences as dire as those of modern warfare. In every exercise conducted by the government so far, the attacking team has managed to completely the target systems with little difficulty. There is a dire need for new approaches to protect the computational infrastructure from such attacks and to enable it to continue functioning even when attacks have been successfully launched.

Our presmise is that to protect the infrastructure we need to restructure these software systems as *Adaptive Survivable Systems*. In particular, we believe that a software system must be capable of detecting its own malfunction and it must be capable of repairing itself. But this means that it must first be able to *diagnose* the form of the failure; in particular, it must both localize and characterize the breakdown.

Our work is set in the difficult context in which there is a concerted and coordinated attack by a determined adversary. This context places an extra burden on the diagnostic component. It is no longer adequate merely to determine which component of a computation has failed to achieve its goal, in addition we wish to determine whether that failure is indicative of a *compromise* to the underlying infrastructure and whether that compromise is likely to lead to failures of other computations at other times. Furthermore, we wish to determine what kind of *attack* compromised the resource and whether this attack is likely to have compromised other resources that share a vulnerability. This paper focuses on the diagnostic component of self adaptive survivable systems.

2 Contributions of this Work

We build on previous work in Model-Based diagnosis [2, 3, 4, 5, 8]. However, the context of our research is significantly different from that of the prior research, leading us to confront several important issues that have not previously been addressed. In particular, we present several new advances in representation and reasoning techniques for model-based diagnosis:

1. We develop representation and reasoning techniques for describing and reasoning about the behaviors and failures of *software systems* (most previous work has focussed on hardware, particularly digital hardware).
2. We develop mixed symbolic and Bayesian reasoning technique for model-based diagnosis. The statistical component of the technique utilizes Bayesian networks to calculate accurate posterior probabilities.
3. We develop a unified framework for reasoning about the failures of the computations, about how these failures are related to compromises of the underlying resources, about the vulnerabilities of these resources and how these vulnerabilities enable attacks.
4. We develop techniques for reasoning about *common-mode* failures. A common-mode failure occurs when the probabilites of the failure modes of two or more components are not independent. This issue has not been substantially addressed in the previous literature on model-based diagnosis.

[1] This article describe research conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research was provided by the Information Systems Office of the Defense Advanced Research Projects Agency (DARPA) under Space and Naval Warfare Systems Center - San Diego Contract Number N66001-00-C-8078. The views presented are those of the author alone and do not represent the view of DARPA or SPAWAR.

5. We develop diagnostic techniques that lead to an estimate of the trustability of the computational resources that are used in a specific computation. These techniques also help us to assess which attacks have occurred and the likelihood that specific attacks have been successful.

These are crucial issues when failure is caused by a concerted and coordinated attack by a malicious opponent. There are many modes of attacking computational systems but the most pernicious attackers seek to avoid detection; therefore they attempt to scaffold the attack slowly, at a nearly undetectable rate. These scaffolding actions will typically appear as minor misbehaviors (i.e. they will cause the system to behave somewhat outside its normal range), but skillful attackers will space out the attacks so that the misbehaviors are infrequent and they will attempt to make the resulting misbehaviors seem as close to normal behavior as possible. This makes it crucial that our diagnostic techniques be capable of extracting information from low-frequency events that closely resemble normal modes of operation.

Attackers aim at high leverage points of the infrastructure, such as operating systems or middleware. This leads to common-mode faults, because once the operating system has been compromised all application components can be caused to fail simultaneously.

The paper first briefly reviews the current state of the art in model-based diagnosis; this work has mainly been concerned with breakdowns caused by the deterioration of hardware components. In particular, we adopt the framework in [4] where each component has models for each of several behavioral modes and each model is given a probability. We will then turn to the question of how to extend these techniques so as to apply them to the diagnosis of *software* systems. We extend our modeling framework to account for the fact that software systems are built in layers of infrastructure, with compromises to one layer affecting all higher levels. A software system has a great deal of hidden state; what we are actually capable of observing is the behavior of a specific *computation*; but this particular computation uses a variety of *resources* (e.g. the operating system and middleware, data-sets, etc.). These resources may have been subject to a variety of *compromises*, each of which might lead to a different misbehavior of the computation. Compromises to the resources occur because the resources possess *vulnerabilities* that allow specific *attacks* to take control of the resources for purposes other than those intended by the original designers.

We will finally present mixed symbolic and statistical diagnostic algorithms for assessing the posterior probabilities of the various behavior modes of each component in the model. We present an implementation and show an example of the reasoning process. Finally, we discuss the demands placed on the diagnostic component by our goal of self-adaptivity and conclude with suggestions for future research.

3 Related Research

Model-Based Diagnosis is a symptom directed technique; it is driven by the detection of discrepancies between the observations of actual behavior and the predictions of a model of the system. Almost all of the reported work in the area [2, 1, 3, 4, 5, 8] has been concerned with the diagnosis of physical systems subject to routine breakdown. Model-based diagnostic systems use simulation models that compute expected outputs given known inputs; they utilize dependency directed techniques to link each intermediate and final value to the selected behavioral model of any component of the system which was involved in producing that value.

The completeness of the diagnostic process is dependent on having bi-directional simulation models for each component of the system. Such models produce both a set of assertions recording what values are expected and a dependency network linking these assertions to one another and to assertions stating which components must be in a particular behavioral mode for those values to appear.

Our work builds on the framework in Sherlock [4] and on the probabilistic techniques in [8]. In Sherlock the description of a component includes multiple simulation models, one for each behavioral mode of the component. One distinguished mode is the normal mode, but behavioral models for known failure modes may also be provided. It is also typical to include a null model to account for unknown modes of behavior. Finally, each of the behavioral modes of a component is assigned an *a priori* probability. Sherlock uses these to guide a best first search for a set of behavioral modes, one for each component, such that the models for those modes predict the observed behavior. This is the most likely diagnosis. However, these techniques i depend on the assumption that the failure modes of the components are independent; as we will see this assumption doesn't hold in our environment. Later work [8] introduced techniques for applying Bayesian networks in the context of model-based diagnosis, allowing dependencies to be modeled; [10] presents techniques within this framework for generating several likely diagnoses in order of decreasing likelihood.

Because our focus is on detecting the intentional compromise of software components we are forced to face a number of new issues. These include: How to model software components in the spirit of model-based diagnosis; How to deal with the fact that a compromise to the computational infrastructure (e.g. the operating system) can manifest itself in the malfunction of many application components; How to deal with the fact that compromised components may behave in ways that are difficult to distinquish from normal behavior; How to reason about the system so as to extract as much information about possible compromises as we can. In particular, we deal with how to use both symbolic and Bayesian techniques.

4 Modeling Software Computations

Model-Based Diagnosis requires completely invertible models of the components in order to guarantee completeness of its analysis. But the components of a complex software system rarely have input-output relationship that are invertible. We therefore look for other, additional properties, that lead to more complete coverage. In particular, we concentrate here on descriptions of computational delay (or other *Quality Of Service* metrics). In our current implementation we use an interval of expected delay times (i.e. the computation should run no slower than x and no faster than y) as the behavioral models. Figure 1 shows the application of such models in a framework similar to Sherlock. When propagating in the forward direction we add the delay interval predicted by the behavioral model to the interval bounding the arrival time of the latest input. In the backward direction, we use interval subtraction (and only update the bounds on the last input to arrive). When more than one component predicts the bounds for a particular value (e.g. when a model for component A and a model for component C both predict bounds for the value labeled MID), we take the intersection of the two intervals to obtain the tightest bounds implied by the overall model. A discrepancy is detected when the lower bound of an interval exceeds the upper bound.

As in Sherlock we provide several behavioral models for each component, one characterizing normal behavior, others characteriz-

ing known failure modes and a null model to cover all other unexpected behaviors.

Notice that in Figure 1 there are six potential diagnoses, only one of which involves a single point of failure (in component C). The others involve multiple failues with one component running slower than expected and other components masking the fault at Out1 by running faster than expected. In the third diagnosis, component A runs in "negative time"! On the surface, such a diagnosis seems physically impossible and we might expect the diagnostic algorithm to reject it. But, the diagnosis algorithm is guided by our representational choices; the reason this diagnosis involves negative time is that the fast behavioral model of component A predicts a delay interval from -30 to +2.

Such behavior seems very unlikely, and indeed we assign a low likelihood to this model; however, it is not impossible. Suppose that both computations A and C are running on the same computer and further suppose that the computer has been compromised by an attacker. Under these circumstances, it's not impossible for component C to be delayed (because of a parasitic task inserted by the attacker) while component A has been accelerated, running in less than zero time because it has been hacked by the attacker to send out reasonable answers before it receives its inputs.

What we are able to observe is the progress of a computation; but the computation is itself just an abstraction. What an attacker can actually affect is something physical: the file representing the stored version of a program, the bits in main memory representing the running program, or other programs (such as the operating system) whose services are employed by the monitored application.

Thus, we require a more elaborated modeling framework detailing how the behavior of a computation is related to the state of the resources that it uses. In turn, we must represent the vulnerabilities of these resources and the attacks enabled by these vulnerabilities. Finally, we must represent how such attacks compromise the resources, causing them to behave in an undesired manner.

5 Common Mode Failures

A single compromise of an operating system component, such as the scheduler, can lead to anomalous behavior in several application components. This is an example of a *common mode failure*; intuitively, a common mode failure occurs when a single fault (e.g. an inaccurate power supply), leads to faults at several observable points in the systems (e.g. several transistors misbehave because their biasing power is incorrect). Another example comes from reliability studies of nuclear power plants where it was observed that the catostrophic failure of a turbine blade could sever several pipes as it flies off, leading to multiple cooling fluid leaks.

Formally, there is a common mode failure whenever the probabilities of the failure modes of two (or more) components are dependent. Early model-based diagnostic systems have assumed probabilistic independence of the behavior modes of different components [4] in order to simplify the assessment of posterior probabilities. Later work [8] allows for probabilistic dependence; however, it does not explore in detail how to model the causes of this dependence. We deal with common mode failures by extending our modeling framework to make explicit the mechanisms that couple the failure probabilites of different components.

We first extend our modeling framework, as shown in Figure 2, to include two kinds of objects: computational components (represented by a set of delay models one for each behavioral mode) and infrastructural components (represented by a set of modes, but no delay or other behavioral models). Connecting these two kinds of models are conditional probability links; each such link states how likely a particular behavioral mode of a computational component would be if the infrastructural component that supports that component were in a particular one of its modes (normal or abnormal). Each infrastructural component mode will usually project conditional probability links to more than one computational component behavioral mode, allowing us to say that normal behavior has some probability of being exhibited even if the infrastructural component has been compromised (however, for simplicity, figure 2 shows only a one-to-one mapping).

The model also includes *a priori* probabilities for the modes of the infrastructural components, representing our best estimates of the degree of compromise in each such piece of infrastructure. Following a session of diagnostic reasoning, these probabilities may be updated to the value of the *posterior* probabilities.

We next observe that resources are compromised by attacks. Attacks are enabled by vulnerabilities in the resources. For example, many systems in the Unix family are vulnerable to buffer-overflow attacks; most networked systems are vulnerable to packet-flood attacks. An attack is capable of compromising a resource in a variety of ways; for example, buffer overflow attacks are used both to gain control of a specific resource and to gain root access to the entire system. But the variety of compromises enabled by an attack are not equally likely (some are much more difficult than others). We therefore add a third tier to our model to describe the ensemble of attacks assumed to be available in the environment. We connect the attack layer to the resource layer with Conditional probability links that state the likelikhood of each mode of the compromised resource once the attack has been successful.

Our model of the computational environment therefore includes:

- The components of the computation that is being observed
- A set of behavioral models for each component, representing both normal and failure modes.
- The set of resources available to be used by the computational components
- A set of behavioral modes for each resource, representing both normal and compromised modes.
- A map stating which resources are used by each computational component.
- Conditional probabilties linking the modes of the computations to the modes of the resources employed by that component.
- A list of vulnerabilities possessed by each computational resource.
- A description of which attacks are enable by each vulnerability.
- A list of attack types that are believed to be active in the environment.
- A description of which compromised modes of each type of resource can be caused by a successful execution of each type of attack. This is provided as a set of conditional probabilities of the compromised mode given the execution of the attack.

Given this information, simple rule-based inferencing (implemented in the Joshua inference system) deduces which specific resources might have been compromised and with what probability. This information is then used to construct a Bayesian network (in the IDEAL system).

6 Diagnostic Reasoning

Figure 3 shows a model of a fictitious distributed financial system which we use to illustrate the reasoning process. The system con-

sists of five interconnected software modules (Web-server, Dollar-Monitor, Bond-Trader, Yen-Monitor, Currency-Trader) utilizing four underlying computational resources (WallSt-Server, JPMorgan, BondRUs, Trader-Joe).

For each computational component we show the conditional probability tables that describe how the behavioral modes of each computational resource probabilistically depend on the modes of the underlying resources (each resource has two modes, normal and hacked). Note that two computations (Dollar-Monitor and Yen-Monitor) are supported by a common resource (JPMorgan) and compromises to this underlying resource are likely to affect both computations. The failure modes of these two computations are no longer independent; this is indicated by the conditional probabilities connecting the behavior modes of the JPMorgan to those of both Dollar-Monitor and Yen-Monitor. The specific conditional probabilites supplied describe the degree of coupling.

Finally we show the *a priori* probabilities for the modes of the underlying resources. However, when attacks are present in the environment what matters is the conditional probabilities of the different modes of the resources given that an attack has taken place. We hypothesize that one or more attack types are present in the environment, leading to a three-tiered model as shown in figure 4. In this example, we show two attack types, buffer-overflow and packet-flood. Packet-floods can affect each of the resources because they are all networked systems; buffer-overflows affect only the 2 resources which are modeled as instances of a system type vulnerable to such attacks.

As in earlier techniques, diagnosis is initiated when a discrepancy is detected; in this case this means that the predicted production time of an output differs from those actually observed after an input has been presented. The goal of the diagnostic process is to infer as much as possible about where the computation failed (so that we may recover from the failure) and about what parts of the infrastructure may be compromised (so that we can avoid using them again until corrective action is taken). We are therefore looking for two things: the most likely explanation(s) of the observed discrepancies and updated probabilities for the modes of the infrastructural components.

To do this we use techniques similar to [4, 8]. We first identify all conflict sets, and then proceed to calculate the posterior probabilities of the modes of each of the computational components. We do these tasks by a mixture of symbolic and Bayesian techiques; symbolic model-based reasoning is used to predict the behavior of the system, given an assumed set of behavioral modes. Whenever the symbolic reasoning process discovers a conflict (an incompatible set of behavioral modes), it adds to the Bayesian network a new node corresponding to the conflict (see below). Bayesian techniques are then used to solve the extended network to get updated probabilities.

This approach involves an exhaustive enumeration of the combinations of the models of the computational components. This allows us to calculate the exact posterior probabilties. However, this is expensive and the precision may not be needed. It would be possible to instead use the techniques in [10] to generate only the most likely diagnoses and to use these to estimate the posterior probabilities; but we have not yet pursued this approach.

We instead follow the following approach: We alternate the finding of conflicts with the search for diagnoses. After each “conflict” node is added to the Bayesian network (see below) the network is solved; this gives us updated probabilities for each behavioral mode of each component. We can, therefore, examine the behavioral modes in the current conflict and pick that component whose current behavioral mode is least likely. We discard this mode, and pick the most likely alternative; we continue this process of detecting conflicts, discarding the least likely model in the conflict and picking its most likely alternative until a consistent set is found. This process is a good heuristic for finding the most likely diagnosis [2].

Our models of computational behavior (the delay models) are used to predict the behavior of the computational components and to compare the predictions with observations. When a discrepancy is detected, we use dependency tracing to find the conflict set underlying the discrepancy (i.e. a set of behavioral modes which are inconsistent). At this point a new (binary truth value) node is added to the Bayesian network representing the conflict as shown in Figure 5. This node has an incoming arc from every node that participates in the conflict. It has a conditional probability table corresponding to a pure ”logical and” i.e. its true state has a probability of 1.0 if all the incoming nodes are in their true states and it otherwise has probability 1.0 of being in its false state.

Since this node represents a logical contradiction, it is pinned in its false state. Adding this node to the network imposes a logical constraint on the probabilistic Bayesian network; the constraint imposed is that the conflict discovered by the symbolic, model-based behavioral simulation is impossible. We continue to explore other combinations of behavioral modes, until all possible minimal conflicts are discovered. Each of these conflicts extends the Bayesian network as before, The set of such conflicts constitutes the full set of logical constaints on the values taken on within the Bayesian network; thus, once we have augmented the Bayesian network with nodes corresponding to each conflict, the network has all the information available. [3].

At this point, we have found all the minimal conflicts and added conflict nodes to the Bayesian network for each. We therefore also know all the possible diagnoses since these are sets of behavioral modes (one for each component) which are not supersets of any conflict set. For each of these we create a node in the Bayesian network which is the logical-and of the nodes corresponding to the behavioral modes of the components. This node represents the probability of this particular diagnosis. The Bayesian network is then solved. This gives us updated probabilities for all possible diagnoses, for the behavioral modes of the computational components and for the modes of the underlying infrastructural components. Furthermore, these updated probabilities are those which are consistent with all the constraints we can obtain from the behavioral models. Thus, they represent as complete an assessment as is possible of the state of compromise in the infrastructure. These posterior estimates can be taken as priors in further diagnostic tasks and they can also be used as a “trust model” informing users of the system (including self adaptive computations) of the trustworthiness of the various pieces of infrastructure which they will need to use.

7 Results

The sample system shown in Figure 3 was run through several analyses including both those in which the outputs are within the expected range and those in which the outputs are unexpected. Figure 6 shows the results of an analysis in which the outputs are within the expected range. Figure 7 and 8 show the results of an analysis of an

[2] However since the probabilities of the failure modes of different components are not independent, this is only a heuristic

[3] [8] builds logical reasoning directly into the Bayesian network system because the logical inferences needed are simple enough to be accomodated. However, our inference needs are more complex and not easily amenable to this approach

abnormal case. Inputs are supplied at times 10 and 15 for the two inputs of Web-Server; each of the figures shows the times at which the the outputs of Currency-Trader and Bond-Trader are observed. There are four runs for each case, each with a different attack model. In the first, it is assumed that there are no attacks present and the *a priori* values are used for the probabilities of the different modes of each resource. The second run assumes only a buffer-overflow attack; the third run assumes only a packet-flood attack. The fourth run assumes both types of attacks. There are four columns in each of the results chart, one for each of these runs. The top chart in each figure shows the *a priori and posterior* probabilities for each resource being in its "hacked" mode. The middle chart shows the *posterior* probabilities for each mode of each computational component. The bottom bottom chart in each figure shows the *posterior* probabilites that each of the two types of attacks have occurred. [4].

There are more than two dozen possible diagnoses in the abnormal case. It should be noted that even the most likely diagnosis is actually not all that likely; in addition the next several diagnoses are nearly equally as likely. The most likely diagnosis is therefore not particularly informative for our two goals of recovering from the failure and steering away from compromised resources in the future. However, the posterior probabilities of the modes of the infrastructure components are, in fact, useful guides for the second of these goals. The posterior probabilities of the behavioral modes of the computational resources are useful guides for the first goal, because these probabilities aggregate the information contained in the individual diagnoses.

The most significant change is the increase in the probabilities that the resources named JPMorgan and Wallst-server are hacked. This changes the trustworthiness ordering of the resources: JPMorgan is *a posteriori* the least trustworthy resource, while the *a priori* listing ranks Trader-Joe followed by Bonds-R-US as the least trustworthy. This follows from the fact that the JPMorgan resources is utilized by the computations Yen-Monitor and Dollar-Monitor both of which are very likely to be in abnormal modes and the most likely explanation is that that JPMorgan causes a common-mode failure. Notice that in the last two columns when packet-flood attacks are possible, all the resources are much more likely to be hacked. Qualitatively, this is because all the resources are vulnerable to the packet-flood attack. The misbehavior of the computational components provides evidence that JPMorgan is hacked which in turn provides evidence of a packet flood attack. But since packet-flood attacks affect all the resources, this increases the likelihood that other resources are hacked as well. The Bayesian network carries out the quantitative version of this argument.

It is worth noting that this propagation of trust can carry over to resources not used in the misbehaving computation. For example, assume that the environment contains another resource (call it "newbie") that is subject to the same attacks as the ones (e.g. JPMorgan) that participated in the faulty computation. The misbehavior in the computation is evidence that JPMorgan is "hacked" and this, in turn, is evidence that an attacked succeeded. But this would lend weight to the conclusion that other resources (e.g. Newbie) subject to this same attack had also been compromised. The Bayesian network would propagate probabilities in exactly this fashion leading to a posterior assessment that Newbie has been hacked (although this probability will be lower than the probability that JPMorgan is hacked).

8 Conclusions and Future Work

The example above illustrates how model-based reasoning techniques can be used to extract information from a single run. Our example is intentionally fanciful since we are at the present concentrating on the development of the representational and reasoning frameworks. In future work we will explore realistic models of real systems.

The information extracted is probabilistic and it sheds light both on the question of where the computation might have failed, on what underlying resources might have been compromised and on what attacks might have succeeded.

It is notable that the identification of the most likely diagnosis is not particularly informative. For example, in the most likely diagnosis Yen-Monitor is in its Normal mode. However, the most likely behavioral mode for Yen-Monitor is its Slower mode which occurs in many of the remaining diagnoses. The posterior probabilites of the behavioral modes aggregate the probabilites from each of the possible diagnoses, producing an overall assessment that is more informative than any individual diagnosis. Of course, if there are very few diagnoses, or the most likely diagnosis is extremely probable, then the probabilities of its behavioral modes will approximate the overall posterior probabilities.

It is important to keep in mind why we are interested in the diagnoses at all. The goal of the system is to recover from the failure and to steer away from future trouble. To do this it needs to know how much of the computation has been completed successfully and how much remains to be done. Given such information the system would pick a rollback point for recovery that includes no failed part of the computation. Furthermore, the chosen rollback point would maximize the probability of continuing the restarted computation to completion. As we just saw, an individual diagnosis, even the most likely diagnosis, does not give us the information we need to do this. When the available evidence supports multiple diagnostic hypotheses, then our interest should shift from individual diagnoses to aggregate failure probabilities and this information is conveyed completely by the posterior probabilities of the failure modes. I.e. if the posterior probability that Yen-Monitor failed is high, then we don't actually care that there are multiple (multiple point of failure) diagnoses involving this failure nor do we care how likely each of these diagnoses is. Instead what we do care about is that it's very likely that Yen-monitor didn't do its job and that we should select a rollback point prior to its execution. Similarly, in choosing a recovery plan we should avoid using those resources whose posterior failure probabilities are highest. [5].

This is to say that the goal of the diagnostic process should be to assess the overall posterior probabilities of the behavioral modes of the computational and infrastructure components. These give us evidence for which computational resources are to be to be trusted during the recovery process and during subsequent computations. This is

[4] The implementation is in CommonLisp and uses the Joshua [7] rule-based reasoning system as well as the Ideal system [9] and in particular its implementation of the algorithm described in [6]. On a 300 MHz powerbook, the total solution time is under 1 minute. By far, the most expensive part of this is calculating the probabilities of the complete set of diagnoses. The most likely diagnosis and all conflict sets are located in less than 10 seconds)

[5] Of course, gathering further evidence might reduce the number of possible diagnoses leading to greater resolution. However, in our context there are two difficulties with attempting to do this. First, it would take time and there might be tight timeliness contraints on the failed computation (e.g. suppose the computation was processing sensor data which must be acted on very quickly). Second, any attempt to gather more data would involve running the same, or similar, computations again when we know that something is compromised; this might lead to loss or destruction of data. Making this tradeoff correctly involves estimating the expected cost of new information and it expected benefit. It is possible that such and analysis would suggest that acting on the available diagnostic evidence is the best course of action

a different definition of the goal of diagnostic activity than has been used in previous research on model-based diagnosis.

We have not yet addressed the details of how the system should use this information in forming a recovery plan. The general outline is that when assigning a computation to a resource it should choose that resource which is most likely to be in n mode that will successfully complete the computation. But the probabilities of the modes of different resources are not independent; they are linked by the Bayesian network. Having decided to use a particular resource because it's likely to be in an acceptable mode, the system should pin the Bayesian network into a state where the resource is believed to be in the desired state and re-solve the network. Subsequent choices should be made in light of the updated probabilities.

We have also not yet addressed the question of what actions the system might take to obtain more information in future runs. The Minimum Entropy approach in [3] provides a useful framework. However, the current context provides more degrees of freedom; in addition to making new observations, we can also change the assignment of resources to computational components in a way that will maximize the expected gain in information. The details of this remain for future research.

REFERENCES

[1] Randall Davis, 'Diagnostic reasoning based on structure and behavior', *Artificial Intelligence*, **24**, 347–410, (December 1984).

[2] Randall Davis and Howard Shrobe, 'Diagnosis based on structure and function', in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 137–142. AAAI, (1982).

[3] Johan deKleer and Brian Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).

[4] Johan deKleer and Brian Williams, 'Diagnosis with behavior modes', in *Proceedings of the International Joint Conference on Artificial Intelligence*, (1989).

[5] Walter Hamscher and Randall Davis, 'Model-based reasoning: Troubleshooting', in *Exploring Artificial Intelligence*, ed., Howard Shrobe, 297–346, AAAI, (1988).

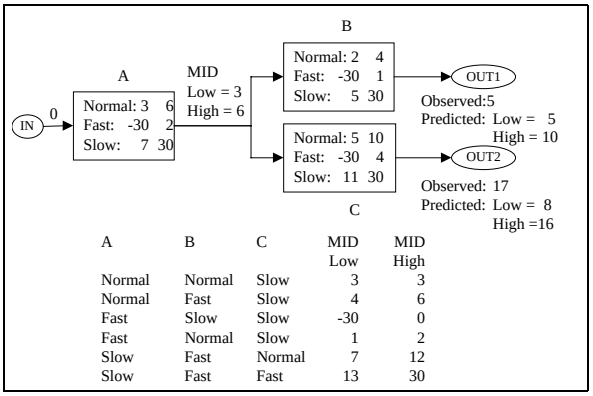
[6] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen, 'Bayesian updating in causal probablistic networks by local computations', *Computational Statistics Quarterly*, **4**, 269–282, (1990).

[7] S. Rowley, H. Shrobe, R. Cassels, and W. Hamscher, 'Joshua: Uniform access to heterogeneous knowledge structures (or why joshing is better than conniving or planning)', in *National Conference on Artificial Intelligence*, pp. 48–52. AAAI, (1987).

[8] Sampath Srinivas, 'Modeling techinques and algorithms for probablistic model-based diagnosis and repair', Technical Report STAN-CS-TR-95-1553, Stanford University, Stanford, CA, (July 1995).

[9] Sampath Srinivas and Jack Breese, 'Ideal: A software package for analysis of influence diagrams', in *Proceedings of CUAI-90*, pp. 212–219, (1990).

[10] Sampath Srinivas and Pandurang Nayak, 'Efficient enumeration of instantiations in bayesian networks', in *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pp. 500–508, Portland, Oregon, (1996).



A	B	C	MID Low	MID High
Normal	Normal	Slow	3	3
Normal	Fast	Slow	4	6
Fast	Slow	Slow	-30	0
Fast	Normal	Slow	1	2
Slow	Fast	Normal	7	12
Slow	Fast	Fast	13	30

Figure 1. Reasoning About Software Component Delay

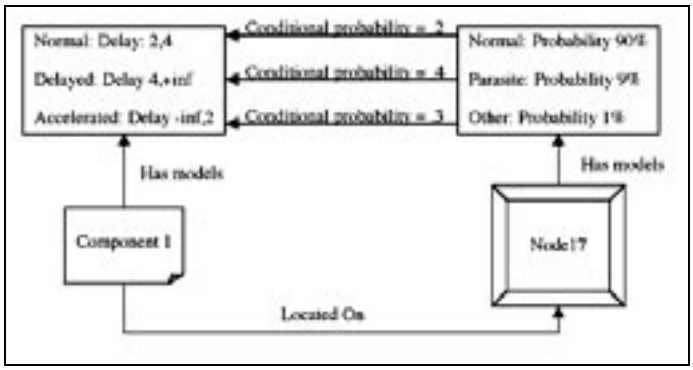


Figure 2. Modeling Computational and Infrastructure Components

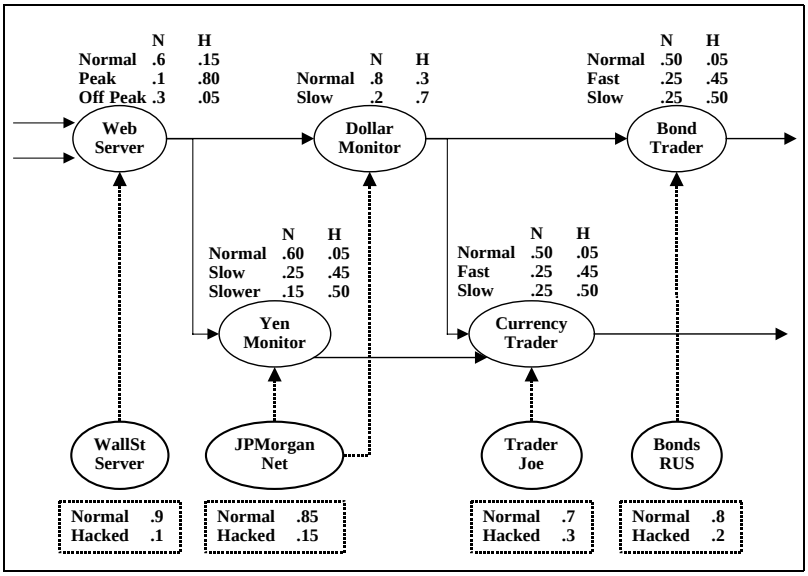


Figure 3. An Example of the Extended System Modeling Framework

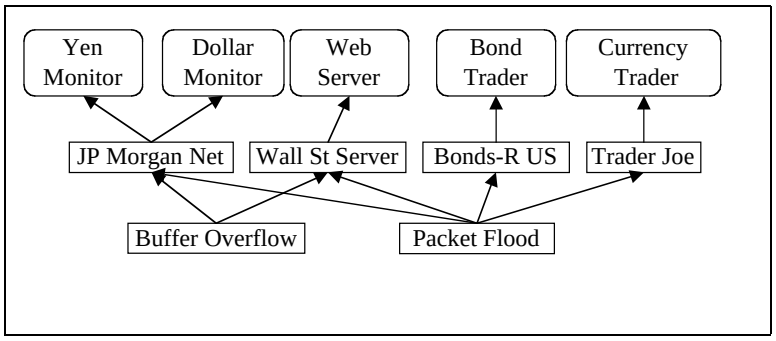


Figure 4. An Example of the Three Tiered System Modeling Framework

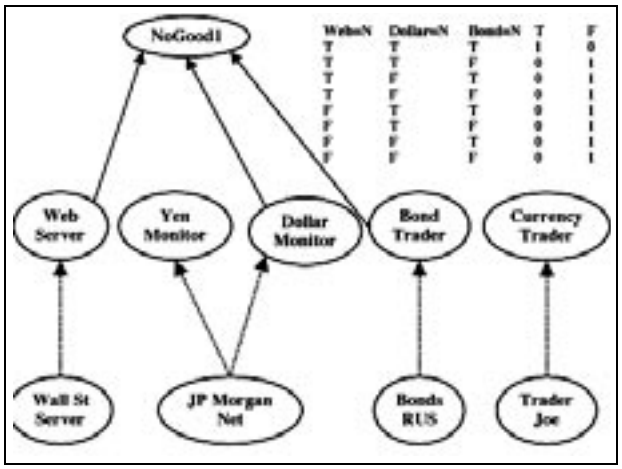


Figure 5. Adding a Conflict Node to the Bayesian Network

Normal Case: 48 “Diagnoses”
30 Minimal Conflicts
Output of Bond-Trader Observed at 25
Output of Current-trader Observed at 28

Name	Prior	Posterior			
Wallst	.10	.04	.07	.09	17
JPMorgan	.15	.07	.08	.11	.19
Bonds-R-Us	.20	.18	.18	.15	.17
Trader-Joe	.30	.28	.28	.16	.18

Computations Using Each Resource					
Web-Server	Off-Peak	.40	.40	.40	.40
	Peak	.04	.05	.05	.08
	Normal	.55	.56	.55	.52
Dollar-Monitor	Slow	.23	.23	.25	.29
	Normal	.77	.77	.75	.71
Yen-Monitor	Really-Slow	.03	.04	.04	.05
	Slow	.21	.21	.21	.25
	Normal	.76	.75	.75	.70
Bond-Trader	Slow	.29	.29	.27	.26
	Fast	.23	.23	.24	.26
	Normal	.48	.48	.49	.48
Currency-Trader	Slow	.09	.09	.07	.06
	Fast	.52	.52	.48	.51
	Normal	.40	.39	.45	.43

Attack Types		Attacks Possible			
Name	Prior	None	Buffer-Overflow	Packet-Flood	Both
Buffer-Overflow	.4	0	.28	0	.30
Packet-Flood	.5	0	0	.23	.25

Figure 6. Updated Probabilities

Slow Fault on both outputs 25 “Diagnoses”
34 Minimal Conflicts
Output of Bond-Trader Observed at 35
Output of Current-trader Observed at 45

Name	Prior	Posterior			
Wallst	.1	.27	.58	.75	.80
JPMorgan	.15	.45	.62	.74	.81
Bonds-R-Us	.20	21	.20	.61	.50
Trader-Joe	.30	.32	.31	.62	.50

Computations Using Each Resource					
Web-Server	Off-Peak	.03	.02	.02	.02
	Peak	.54	.70	.78	.80
	Normal	.43	.28	.20	.18
Dollar-Monitor	Slow	.74	.76	.73	.76
	Normal	.26	.24	.27	.24
Yen-Monitor	Really-Slow	.52	.54	.56	.58
	Slow	.34	.35	.34	.34
	Normal	.14	.11	.10	.08
Bond-Trader	Slow	.59	.57	.76	.70
	Fast	0	0	0	0
	Normal	.41	.43	.24	.30
Currency-Trader	Slow	.61	.54	.62	.56
	Fast	.07	.11	.16	.16
	Normal	.32	.35	.22	.28

Attack Types		Attacks Possible			
Name	Prior	None	.Buffer-Overflow	Packet-Flood	Both
Buffer-Overflow	.4	0	.82	0	.58
Packet-Flood	.5	0	0	.89	.73

Figure 7. Updated Probabilities

Prob ability	Currency Trader	Bond Trader	Yen Monitor	Dollar Monitor	Web Server
.0898	Slow	Slow	Normal	Normal	Peak
.0876	Slow	Normal	Slow	Slow	Normal
.0855	Normal	Normal	slower	Slow	Normal
.0762	Slow	Normal	Really-Slow	Slow	Normal
.0641	Slow	Slow	Slow	Slow	Normal
.0626	Normal	Slow	Really-Slow	Slow	Normal
.0557	Slow	Slow	Really-Slow	Slow	Normal
.0468	Normal	Slow	Slow	Normal	Peak
.0416	Slow	Slow	Slow	Normal	Peak
.0321	Slow	Normal	Normal	Slow	Peak
.0306	Normal	Slow	slower	Normal	Peak
.0301	Normal	Normal	Slow	Slow	Peak
.0276	Slow	Slow	slower	Slow	Off-Peak
.0272	Slow	Slow	slower	Normal	Peak
.0268	Slow	Normal	Slow	Slow	Peak
.0262	Normal	Normal	slower	Slow	Peak
.0260	Fast	Slow	slower	Normal	Peak
.0235	Slow	Slow	Normal	Slow	Peak
.0233	Slow	Normal	slower	Slow	Peak
.0223	Fast	Normal	slower	Slow	Peak
.0221	Normal	Slow	Slow	Slow	Peak
.0196	Slow	Slow	Slow	Slow	Peak
.0192	Normal	Slow	slower	Slow	Peak
.0171	Slow	Slow	slower	Slow	Peak
.0163	Fast	Slow	slower	Slow	Peak

Figure 8. diagnoses

Observations and Results Gained from the Jade Project

Wolfgang Mayer[*] and **Markus Stumptner**[†] and **Dominik Wieland**[*] and **Franz Wotawa**[‡ §]

Abstract

This paper summarizes the work done in the course of the Jade project, which deals with automatic debugging of Java programs. Besides a brief introduction to the Jade project, models developed to debug Java programs are evaluated and results are presented. Furthermore, insights gained from the results are discussed and topics for further research are identified.

1 Introduction

For the last three years the Jade project has examined the applicability of model-based diagnosis (MBD) techniques to the software debugging domain. In particular, the goals of Jade were (1) to establish a general theory of model-based software debugging with a focus on object-oriented programming languages, (2) to describe the semantics of the Java programming language in terms of logical models usable for diagnosis, and (3) to develop an intelligent debugging environment for Java programs based on theoretic results.

The main practical achievement of the Jade project is the interactive debugging environment, which allows us to efficiently locate bugs in faulty Java programs. Currently, this debugger is fully functional with regard to nearly all aspects of the Java programming language and comes complete with a user-friendly GUI, the diagnosis system being integrated into a "normal" interactive debugger interface. The Jade debugger limits the search space of bug candidates by computing diagnoses for a given (incorrect) input/output behavior. This is done by using model-based diagnosis techniques, which in some cases have been adapted to suit the needs of an object-oriented debugging environment. Furthermore, the debugger can be used to unambiguously locate faults through an interactive debugging process, which is based on the iterative computation of diagnoses, measurement selection, and input of additional observations by the user.

This work is organized as follows: The next section briefly describes the program models used by the Jade debugging environment. Section 3 presents results obtained with the models introduced in Section 2. Section 4 analyzes the results from Section 3 and discusses some properties of the models. In Section 5, we point out interesting topics for further research. Section 6 briefly compares our approach to related work. Finally, we conclude the paper.

2 Program models

Since model-based diagnosis relies on the existence of a logical model description of the underlying target system, one of the most important components of the Jade system are its models. Currently, the Jade debugger makes use of two model classes, dependency-based models and value-based models. This section briefly describes these model types. More comprehensive descriptions can be found in [Stumptner *et al.*, 2001; Wieland, 2001; Mayer, 2000; 2001].

Dependency-based models are based on the collection of all data and control dependencies of a given Java program. As an example, we look at a single statement S_i, e.g., `int x=a*b;`. Informally, the variable dependencies arising from this statement can be specified by $S_i : x \leftarrow \{a, b\}$. A formal logical model can now automatically be derived from this dependency. For our example it reads $\neg AB(S_i) \wedge ok(a) \wedge ok(b) \Rightarrow ok(x)$, where the predicate AB stands for the assumption that a certain statement is incorrect, i.e., behaves abnormally. The predicate $ok(v)$ specifies that the value of variable v is correct without making use of the concrete value of v. Observations for such a model can be expressed by specifying the correctness or incorrectness of a certain variable, e.g., $\neg ok(x)$ in the above example. In the course of the Jade project different dependency-based models have been created that vary in their levels of abstraction and the amount of information used during their creation. These models are:

ETFDM: A dependency-based model, which makes use of a concrete execution trace [Wieland, 2001].

[*]Vienna University of Technology, Institute for Information Systems, Database and Artificial Intelligence Group, Favoritenstrasse 9-11, A-1040 Vienna, Austria, email: {mayer,wieland}@dbai.tuwien.ac.at

[†]University of South Australia, Advanced Computing Research Center, 5095 Mawson Lakes (Adelaide) SA, Australia, email: mst@cs.unisa.edu.au

[‡]Graz University of Technology, Institute for Software Technology, Inffeldgasse 16b/II, A-8010 Graz, Austria, email: wotawa@ist.tu-graz.ac.at

[§]Authors are listed in alphabetical order

#	Test series	#TC	Diagnosis					Debugging					
			$\varnothing S_1$	$\varnothing D_1$	$\varnothing D_1(\%)$	$\varnothing D_2$	$\varnothing D_2(\%)$	$\varnothing S_2$	$\varnothing R$	$\varnothing T_1$	$\varnothing T_1(\%)$	$\varnothing T_2$	$\varnothing T_2(\%)$
1	Adder	14	17	8.14	48	8.14	48	17	10	3.9	39	3.9	39
2	IfTest	10	3.5	2.2 (1.9)	63 (54)	2.0	57	6.3	4.9	3	61	2.8	57
3	WhileTest	10	5.6	3.3	59	2.5	47	11.7	5.4	5.1	94	3.9	72
4	Numeric	9	4.6	4.6	100	4.6	100	6.2	3.6	4.4	120	5.3	147
5	Trafficlight	4	5	3	60	3	60	14	7.25	6.25	86	6.25	86
6	Library	5	26	20.6 (18)	79 (69)	20	77	33	18.6	7.8	42	7.6	41
$\varnothing$		1	10	6.3 (5.9)	**63 (59)**	6	**60**	13.4	7.6	4.6	**60**	4.5	**59**

Table 1: Diagnosis and debugging results of the dependency-based models

DFDM: A dependency-based model, which only makes use of static (compile-time) information, such as the Java source code and the programming language semantics [Stumptner *et al.*, 2001; Wieland, 2001].

SFDM: Another dependency-based model, which is based on either the ETFDM or the DFDM and involves a higher level of abstraction by removing the distinction between object locations and references [Stumptner *et al.*, 2001; Wieland, 2001].

Value-based models are models which make use of concrete execution values and propagate these values from the model's inputs to its outputs and (if possible) from the model's outputs to its inputs. A simple value-based model for the above example statement reads $\neg AB(S_i) \vdash x = a * b$, where x, a, and b stand for concrete variable values as computed at run-time. In the case of value-based models observations can be expressed by specifying the concrete value of a certain variable, e.g., $x = 6$ in the above example. The Jade system currently operates on the following two value-based model types:

VBM: A value-based model, which makes use of not only the underlying program dependencies, but also concrete evaluation values and the full programming language semantics [Mayer, 2000].

LF-VBM: A second value-based model, which is based on the unfolded source code for a particular program run [Mayer, 2001]. In particular, the loops are expanded into a set of nested conditional statements, where the conditional statements are modeled specially in order to provide better backward reasoning capabilities.

Although the expressiveness of the individual models is not exactly the same, all models support a considerable subset of the Java programming language. Currently, exception handling and programs using multiple threads are not supported. Furthermore, the value-based models do not support recursive method calls. The models are designed to locate functional faults, e.g. wrong operators or reversed conditions. They cannot reliably locate structural faults or more severe defects, such as wrong algorithms or data structures.

3 Results

In this section we describe results obtained by applying the models introduced above to a set of example programs and compare them with respect to their debugging and diagnostic accuracy. The tests were separated into two test sets, where one test set was used to compare the dependency-based models, whereas the other set was used to evaluate the value-based models. A comparison between the dependency-based models and the value-based models can be found in [Stumptner *et al.*, 2001]. Most of the example programs can be obtained from `http://www.dbai.tuwien.ac.at/proj/Jade/`.

3.1 Dependency-based models

The first test series aims at evaluating the performance of the used dependency-based models, i.e., DFDMs, ETFDMs, and SFDMs. Furthermore, we compare the results scored by these model types. In particular, the test series has two main goals: (1) to examine the ability of the Jade debugger to reduce the search space of bug candidates. In other words, we test which parts of a Java program can automatically be excluded from the fault localization process in a single diagnosis step and which parts of the search space remain for further debugging actions. (2) to evaluate the debugging performance of the Jade tool, i.e., determining the amount of user interaction needed to unambiguously locate a fault in a Java program.

In order to carry out these tests we implement a couple of test programs demonstrating simple variable dependencies (simulating a binary adder, numeric examples), making use of control structures (if and while statements), and finally multiple objects and instance fields together with linked lists and general processing (a small library application). We then construct test cases for each program P by specifying the correct input/output behavior of P and installing a single-fault into P. Overall 52 test cases are constructed and used for the evaluation of the system's performance. Table 1 shows all tests carried out with each row summarizing all tests performed in a single test series. Column $\#TC$ denotes the number of tests of the respective test series.

The diagnostic performance of the Jade system in the context of dependency-based models is given in columns 4 to 8 of Table 1. Column $\varnothing S_1$ shows the average number of top-level statements of the tested programs in a single test series. Since the Jade tool performs hierarchical debugging, only these top-level statements (this excludes statements nested in loops and selection statements) can be identified as the source of a fault in a single diagnosis step. Columns $\varnothing D_1$ and $\varnothing D_2$ present the number of top-level statements, which remain as possible fault candidates after a single diagnosis step has been performed using DFDMs and ETFDMs, re-

spectively. In other words, the difference between $\varnothing S_1$ and $\varnothing D_1$ ($\varnothing D_2$) shows the number of statements, which can be eliminated from the debugging scope in a single diagnosis step. Columns $\varnothing D_1(\%)$ and $\varnothing D_2(\%)$ show the number of remaining statements for both model types in relation to the total number of top-level statement, i.e., $\varnothing S_1$. These columns present the percentage of statements, which remain as possible fault candidates for further debugging actions. All tests are also performed with the simplified versions of the test programs' DFDMs. In cases where these tests yield results different from tests with the full DFDMs, the results obtained from the SFDMs are given in brackets. Note that no tests are carried out with simplified versions of ETFDMs, since these models are not yet fully supported by the Jade debugging tool.

The right side of Table 1 (columns 9 to 14) depicts the debugging performance of the Jade debugging environment. Since we are now interested in the exact localization of faults, we no longer deal with top-level statement only, but also take statements nested in loop and selection statements into consideration. Column $\varnothing S_2$ shows the average number of all statements of the respective tested program. Column $\varnothing R$ includes the average indices of those statements, in which the single fault has been installed during the test design phase. If we argue that with traditional debugging tools one has to step through the code manually statement by statement until the bug is located, the values in column $\varnothing R$ provide a reasonable reference value for the amount of user interaction needed by the Jade system to exactly locate a fault. The latter is presented in columns $\varnothing T_1$ (DFDMs) and $\varnothing T_2$ (ETFDMs). Columns $\varnothing T_1(\%)$ and $\varnothing T_2(\%)$ show the average number of user interaction relative to the average index of the buggy statement, i.e., $\varnothing T_1(\%) = \varnothing T_1/\varnothing R$ and $\varnothing T_2(\%) = \varnothing T_2/\varnothing R$.

3.2 Value-based models

In a second step we test the diagnostic performance of the more detailed and semantically stronger value-based models, i.e., VBMs and LF-VBMs. For this task we implement a second set of example programs which is designed especially to investigate the specific advantages and disadvantages of the value-based model variants. Whereas some examples are small and specifically designed to demonstrate different aspects of the models, most of the example programs implement well-known algorithms which could be part of larger programs. For example, programs executing a binary search procedure, computing the Huffman encoding of an array of characters, or applying Gauss elimination are part of this test suite. Similar to the tests carried out with the dependency-based models, faults were seeded into each program such that each test case is influenced by one fault. Again, we assume that the faulty program is a close variant of the correct program. We do not deal with wrong choice of algorithms, data structures or similar major design defects.

The diagnostic experiments are performed by specifying the inputs of the program together with the expected results as observations. A summary report of the obtained results for each example program is depicted in Table 2. Several aspects of the examples are listed: Stm denotes the number of statements in the program, C represents the number of components in the generated model. D stands for the number of diagnoses of minimal cardinality that are obtained and H represents the number of diagnoses from D that actually include the seeded fault. S denotes the cardinality at which the diagnostic process is stopped because the seeded fault has been located. Finally, the %-column lists the percentage of the statements that have to be examined in the worst case until the seeded fault is found. Here it is assumed that the diagnoses are presented with increasing cardinality. Note that these numbers can further be improved by suitable heuristics, which present the diagnoses according to their 'likelihood' to explain the faults. For the VBM, the columns H and S are omitted because their value is always equal to one. Numbers in parentheses denote cases where the faults cannot be located because the maximum time allowed for diagnosis is exceeded. In these cases the numbers are lower bounds to the actual results that would be obtained when continuing the diagnostic process to its completion.

Program	Stm	VBM			LF-VBM				
		C	D	%	C	D	H	S	%
BinSearch	27	16	6	63	43	1	1	2	8
Binomial	76	26	9	42	255	24	1	1	32
BoundedSum	16	14	4	38	19	1	0	2	38
BubbleSort	15	10	6	93	34	7	1	1	47
FindPair	5	4	4	100	10	1	0	2	80
FindPositive2	17	13	3	41	20	2	1	1	12
FindPositive3	17	13	3	41	20	2	1	1	12
Hamming	27	19	11	70	95	9	1	1	33
Huffman	64	22	9	80	161	9	0	(2)	(25)
Huffman	64	22	6	59	164	12	1	1	19
Intersection	95	31	12	84	155	8	1	1	5
Library	24	21	6	38	36	5	0	2	34
Matrix	71	21	21	100	127	37	1	1	52
MaxSearch2	21	16	3	38	37	2	0	2	19
MultLoops	21	12	2	19	27	4	2	3	24
MultiSet	97	55	8	28	283	1	0	(2)	(11)
Permutation	24	17	14	96	29	3	1	1	13
Permutation0	26	19	12	69	33	1	1	1	4
Permutation1	26	19	12	69	32	8	0	3	100
Permutation2	26	19	15	85	33	9	1	1	35
Permutation3	24	19	12	67	33	2	0	3	50
Polynom	120	64	14	24	189	26	0	(3)	(13)
SearchTree	84	41	41	100	140	45	0	(1)	(54)
SkipEqual	5	4	4	100	11	2	1	1	40
Stat	23	17	3	39	42	2	0	4	48
Sum	5	4	3	80	10	3	1	1	40
SumPowers	21	12	8	81	36	5	1	1	24
∅	39	20	9	**65**	77	8	0.6	(1.6)	(**32**)

Table 2: Diagnosis results of the value-based models

4 Discussion

Based on the results from Section 3, in this section we discuss some important properties of the proposed models and present insights gained during the Jade project.

From the results it can be seen that the amount of code that has to be analyzed in order to locate a fault can be reduced significantly with all models. If we look at Table 1 we find that in the test series carried out with dependency-based models approximately 40% of the top-level statements can be eliminated from the debugging scope, leaving some 60% for further debugging actions. Interestingly, the average results obtained with different dependency-based model types were quite similar with slight advantages to ETFDMs (in comparison to DFDMs) and full model versions (in comparison to SFDMs). In the case of value-based models, the results lie in the same order of magnitude. In particular, between 40 and 80% of all statements have to be checked, with the average being at 65%. Note that this does not indicate a better performance of dependency-based models in comparison to value-based models, since completely different test programs were used to evaluate the different model types. In particular, the test series with the value-based variants in general used longer and more complex test methods. These methods result in only very few statements being removed from the suspect code in case of dependency-based models, but still yield remarkable results with VBMs. For a more detailed comparison of dependency-based and value-based models see [Stumptner *et al.*, 2001].

Dependency-based models One major advantage of dependency-based models is that they can be constructed and applied to actual diagnosis problems very quickly. This is also true for medium- to large-size programs. They are also easier to handle than their value-based counterparts, since they require observations only to state whether the value of a certain variable is correct or not, whereas with value-based models concrete execution values are needed. Generally, the use of ETFDMs results in fewer single diagnoses, because concrete execution traces are used during the collection of the dependencies. This becomes especially apparent for programs, which include loop and selection statements or recursive method calls. The improved debugging performance of ETFDMs in comparison to DFDMs comes with longer modeling times, since now the creation of a model not only depends on the underlying source code, but also on the existence of an execution trace, whose creation requires running the program. It was also shown that the full versions of DFDMs and ETFDMs are superior to their simplified counterparts. This is, because they model object locations and object references by separate model constructs and thus provide a finer-grained model architecture. On the other hand the computation of diagnoses with full model versions is computationally more expensive. Further on, the specification of observations is easier with simplified model versions.

The Value-Based Model However, dependency-based models did not prove to be an optimal solution for all tested programs due to their lack of run-time information. Note that even ETFDMs do not make use of concrete evaluation values directly, but only extract information about executed branches and numbers of iterations of loops from concrete execution traces. Therefore, the VBM was developed, which makes use of the full programming language semantics and propagates concrete evaluation values through the system. As already mentioned, in many cases VBMs score satisfying results with programs, which can hardly be diagnosed using dependency-based approaches only. [Stumptner *et al.*, 2001] indicates that in general value-based models are superior to their dependency-based counterparts. Therefore, although VBMs have the drawbacks of their high computational requirements, VBMs have proved as satisfying general-purpose alternatives and complements to dependency-based models.

Loop Handling A negative aspect of the dependency-based models and the VBM is the fact that these models provide good results for programs without loops but fail to compute satisfying diagnoses for programs that consist of large loop statements. This is due to the fact that loop statements are modeled hierarchically and discrimination between statements inside the loops is not possible. To overcome these problems, the LF-VBM expands loops into a set of nested conditional statements, with separate assumption variables for each statement. The number of conditional statements is derived from the initial execution of the test cases. Therefore, the model is able to reason about the statements inside the loop independently, without considering the whole loop as an entity. This provides a finer-grained resolution, which avoids the problem of large diagnosis entities mentioned above.

As can be seen in Table 2, switching from the VBM to the LF-VBM leads to much better results. In particular, the percentage of statements that has to be considered until the fault is located is reduced to 32-43%[1] on average, which is quite low compared to the percentage of statements that was computed by the VBM. For the LF-VBM it is no longer the case that every faulty statement is included in a diagnosis of cardinality one (as with the VBM). Therefore, the cardinality up to which diagnoses have to be computed is likely to be greater than one, depending on the type of fault and the program structure. For most example programs the diagnosis cardinality required to locate a fault is less than or equal to two, which is usually computationally feasible when considering small- to medium-sized programs. Another aspect of the LF-VBM that keeps the model from being blindly applicable is the fact that the strong fault modes of the conditional statements decouple the selection of the conditional branch to be executed from the evaluation of the selection condition. Therefore, faults in the condition cannot be located using the LF-VBM. Fortunately, such faults can in many cases be found with the VBM alone and do not require the LF-VBM to be applied.

In case of dependency-based models additional tests have been carried out to examine the overall debugging performance of the Jade tool. As Table 1 indicates, the average number of user interactions needed by the Jade tool is significantly smaller than the amount of user interactions needed by traditional debugging tools. On average some 40% of user interactions can be saved using the Jade tool. In general, the direct comparison of user interactions is problematic, since different user interactions require different types of inputs from the user, which vary in time, complexity, and knowledge

[1] 43% is obtained when assuming the whole program has to be examined for the examples where no exact solution was found. Better estimates (37%) are obtained when taking the percentages obtained with the VBM as upper bounds.

needed by the user. The numbers given in Table 1 therefore include all user interaction performed by the Jade system. If only variable queries, i.e., the input of a new observation in the form of the value of a certain variable at a given source code position, are counted, the average amount of user interaction amounts to only 35% of the user interaction needed by traditional debugging tools. Since strictly speaking all other kinds of user interactions are not included in the reference value of traditional debuggers, this lower value probably provides a more accurate measurement of the debugging performance of the Jade system.

Comparison If we compare the results obtained with the Jade system to results obtained with other approaches for program analysis, it can be seen that the approaches described herein are comparable and in many cases even superior to other techniques. When comparing our approach to slicing [Weiser, 1984], we find that with dependency-based models we yield similar results to those obtained by slicing techniques. When value-based models are used, our results are much better, because for most of the example programs used during the evaluation of the value-based variants, static slicing is not able to eliminate any statement. This can be explained by the different levels of abstraction applied by dependency-based models and slicing on the one hand and value-based diagnosis techniques on the other hand. The value-based approach is somewhat closer to the actual execution semantics of the program than with both, program slicing and dependency-based models. Another improvement with respect to slicing is that we can provide more information to the user, if a loop has to be executed a different number of times to explain a fault. Those examples where no statements of the program can be eliminated are programs that are either very short (consisting of only an initialization statement and a loop) or programs where almost every part of the program depends on every other part (for example a binary search tree, where the program execution depends on the values that were inserted previously).

5 Ongoing Work

Although the results presented in the previous section are already promising, there remain topics for further research. This section discusses possible enhancements of the models, to avoid some of the drawbacks mentioned in Section 4.

First, no single model is able to efficiently locate faults. Rather, a combination of models has to be applied to perform efficient reasoning. This multi-model-reasoning approach is not only applicable to a single level of abstraction, as in the case of the VBM and the LF-VBM, but can also be applied using multiple levels of abstraction or types of models. For example, the dependency-based models can be used to narrow the region of interest and then apply combinations of the VBM and the LF-VBM to exactly locate the fault. Also, models dealing with structural faults [Jackson, 1995; Wotawa, 2000] or various special-purpose models (e.g., to locate faults in loops, selection statements, etc...) could be incorporated in such a framework.

For this approach to be applicable, suitable strategies to decide under which conditions to apply certain kinds of models have to be developed and evaluated. Based on these criteria, the most efficient model can be selected based on the program structure, the test cases and the diagnoses computed so far. This approach overcomes the drawbacks of the models, as well as reduces the computational complexity of the diagnostic process, because models are only instantiated when needed. To select candidates for further inspection, suitable criteria for ranking diagnoses according to their likelihood to explain the fault have to be developed.

As far as the fault classes which can be located with the Jade environment are concerned, it should already have become clear that we are interested in source code bugs which become observable as failures or output errors and manifest themselves as logical faults in the analyzed source code. This explicitly excludes compile-time and run-time failures as well as faults leading to the non-termination of a program. For a discussion about the fault classes handled by the Jade system we divide the class of analyzed faults into functional and structural faults. Functional faults are all faults, which result in a certain variable storing an incorrect value in at least one possible evaluation trace. In particular, these faults include the use of incorrect operators or the specification of incorrect literals, such as integer or boolean constants. Since these faults do not alter the structure of the program, faults belonging to this class can generally be found with the Jade debugging environment, once they become observable through a test case leading to an incorrect variable value.

Structural faults, on the other hand, are source code bugs which alter the structure of the underlying program. This is the case if the dependency graph [Ferrante *et al.*, 1987] of the program is not structurally equivalent to the dependency graph of the correct program. The result of these faults is that the system description, i.e., the model, differs from the system description obtained by the correct program. At the moment structural faults can only be located under certain circumstances. A discussion about different classes of structural faults and how they are handled by the Jade tool is given in [Wieland, 2001]. In the future special-purpose models have to be developed that handle different kinds of structural faults. As already discussed, these models then have to be combined with the general-purpose models described herein to increase not only the performance of the Jade debugger, but also the number of fault classes handled by the tool.

To aid the programmer in correcting a fault, an intelligent debugging environment should be able to provide possible corrections for a faulty part of a program. As described in [Stumptner and Wotawa, 1999], after a single diagnosis has been selected for further investigation, possible replacement expressions for the faulty expression can be inferred and presented as corrections.

Finally, intuitive means for specifying the expected behavior of a program have to be developed. This includes the construction of an intuitive graphical user-interface through which the user can easily switch between different levels of abstraction, test case specification, and other representations of the program (e.g., visualizations of heap structures, etc.).

6 Related Work

This section briefly summarizes related research in the area of program debugging and compares the approaches to our work.

Weiser's slicing approach [Weiser, 1984] is probably the most widely known approach to improve program debugging. His approach relies on the program dependencies and tries to eliminate those parts of a program that cannot contribute to an observed faulty program behavior. This approach is comparable to the dependency-based models presented here. Details on the relationship between these approaches can be found in [Wotawa, 2001].

Shapiro [Shapiro, 1983] introduces a theoretical framework for algorithmic program debugging and several algorithms suited to debug logic programs. However, the approach suffers from heavy user interaction, which is undesirable when debugging larger programs. In addition, the algorithms cannot locate faults inside procedures.

In [Console *et al.*, 1993] the application of model-based diagnosis to the software domain has been proposed for the first time. This paper introduces a way of using MBD by removing and adding Horn clauses to Prolog programs. Extensions of this approach were developed in [Bond, 1994].

Liver [Liver, 1994] discusses the use of a functional representation in the debugging of software to reduce the problem of structural faults in software, where statements are missing or superfluous parts of a program are the source of errors. The approach relies on symbolic execution of a functional specification, which has to be provided by the user.

Hunt [Hunt, 1998] applies the idea of MBD to the domain of object-oriented languages by building models for programs written in Smalltalk. The model used in this work is based on dependencies between instance variables and method calls that modify them. In contrast to our approach, [Hunt, 1998] is limited to single faults.

MBD concepts have also been applied to VLSI design languages, in particular VHDL [Friedrich *et al.*, 1999], using papers describe (abstract) models used for locating a concurrent statement, e.g., a VHDL process, responsible for a detected misbehavior. The Jade project builds on this work, but extends the previous approaches by modeling of object-oriented features.

Finally, Burnell and Horvitz [Burnell and Horvitz, 1995] present another approach to program debugging using probability measurements to guide diagnosis. As this approach relies on belief networks, which have to be initialized by domain experts, it is doubtable whether this approach can be applied to arbitrary programs.

7 Conclusion

Building intelligent debugging aids for programmers is an important goal repeatedly attacked by researchers during the last decades. Unfortunately, no generally applicable solution has been found so far. In this paper we summarize the work done during the Jade project and discuss some results obtained using the introduced model types. Besides the results, specific advantages and disadvantages of each of the models are discussed. Incorporating these models in a system with multi-model reasoning capability and ranking criteria for diagnoses holds the promise of wider applicability and even better discrimination. As our approach clearly outperforms classical debugging techniques for many example programs, the model-based approach can be considered a promising technique that should be further researched to obtain a generally applicable debugging tool.

Acknowledgments

This work was partially supported by the Austrian Science Fund project P12344-INF.

References

[Bond, 1994] Gregory W. Bond. *Logic Programs for Consistency-Based Diagnosis*. PhD thesis, Carleton University, Faculty of Engineering, Ottawa, Canada, 1994.

[Burnell and Horvitz, 1995] Lisa Burnell and Eric Horvitz. Structure and Chance: Melding Logic and Probability for Software Debugging. *Communications of the ACM*, 38(3):31 – 41, 1995.

[Console *et al.*, 1993] Luca Console, Gerhard Friedrich, and Daniele Theseider Dupré. Model-based diagnosis meets error diagnosis in logic programs. In *Proceedings* 13^{th} *International Joint Conf. on Artificial Intelligence*, pages 1494–1499, Chambery, August 1993.

[Ferrante *et al.*, 1987] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987.

[Friedrich *et al.*, 1999] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(2):3–39, July 1999.

[Hunt, 1998] John Hunt. Model-Based Software Diagnosis. *Applied Artificial Intelligence*, 12(4):289–308, 1998.

[Jackson, 1995] Daniel Jackson. Aspect: Detecting Bugs with Abstract Dependences. *ACM Transactions on Software Engineering and Methodology*, 4(2):109–145, April 1995.

[Liver, 1994] Beat Liver. Modeling software systems for diagnosis. In *Proceedings of the Fifth International Workshop on Principles of Diagnosis*, pages 179–184, New Paltz, NY, October 1994.

[Mayer, 2000] Wolfgang Mayer. Modellbasierte Diagnose von Java-Programmen, Entwurf und Implementierung eines wertbasierten Modells. Master's thesis, Institut für Informationssysteme, Abteilung für Datenbanken und Artificial Intelligence, TU Wien, 2000. (only available in German).

[Mayer, 2001] Wolfgang Mayer. Evaluation of Value-Based Models for Java Debugging. Technical report, Technische Universität Wien, Institut für Informationssysteme 184/2, Paniglgasse 16, A-1040 Wien, Austria, 2001.

[Shapiro, 1983] Ehud Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, Massachusetts, 1983.

[Stumptner and Wotawa, 1999] Markus Stumptner and Franz Wotawa. Debugging Functional Programs. In *Proceedings* 16^{th} *International Joint Conf. on Artificial Intelligence*, pages 1074–1079, Stockholm, Sweden, August 1999.

[Stumptner *et al.*, 2001] Markus Stumptner, Dominik Wieland, and Franz Wotawa. Comparing Two Models for Software Debugging. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI)*, Vienna, Austria, 2001.

[Weiser, 1984] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[Wieland, 2001] Dominik Wieland. *Model-Based Debugging of Java Programs Using Dependencies*. PhD thesis, Vienna University of Technology, Computer Science Department, Institute of Information Systems (184), Database and Artificial Intelligence Group (184/2), November 2001.

[Wotawa, 2000] Franz Wotawa. Debugging VHDL Designs using Model-Based Reasoning. *Artificial Intelligence in Engineering*, 14(4):331–351, 2000.

[Wotawa, 2001] Franz Wotawa. On the Relationship between Model-based Debugging and Programm Mutation. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis*, Sansicario, Italy, 2001.

Hybrid Diagnosis with Unknown Behavioral Modes

Michael W. Hofbaur[1] and **Brian C. Williams**[2]

Abstract. A novel capability of discrete model-based diagnosis methods is the ability to handle *unknown modes* where no assumption is made about the behavior of one or several components of the system. This paper incorporates this novel capability of model-based diagnosis into a hybrid estimation scheme by calculating partial filters. The filters are based on causal and structural analysis of the specified components and their interconnection within the hybrid automaton model. Incorporating unknown modes provides a robust estimation scheme that can cope, unlike other hybrid estimation and multi-model estimation schemes, with unmodeled situations and partial information.

1 Introduction

Modern technology is increasingly leading to complex artifacts with high demands on performance and availability. As a consequence, fault-tolerant control and an underlying monitoring and diagnosis capability plays an important role in achieving these requirements. Monitoring and diagnosis systems that build upon the discrete model-based reasoning paradigm[8] can cope well with complexity in modern artifacts. As an example, the Livingstone system[22] successfully monitored and diagnosed the DS-1 space probe in flight, a system with approximately 4^{80} modes of operation. However, a widespread application of discrete model-based systems is hindered by their difficulty to reason about the continuous dynamics of an artifact in a comprehensive manner. Continuous behaviors are difficult to capture by the pure qualitative models that are used by the reasoning engines. Nevertheless, additional reasoning in terms of the continuous dynamics is vital for detecting functional failures, as well as low-level incipient (i.e slowly developing) faults and subtle component degradation.

Hybrid systems theory provides a modeling paradigm that integrates both, continuous state evolution and discrete mode changes in a comprehensive manner. Recent work in hybrid estimation[14, 16, 24, 9] attempts to overcome the shortcomings of discrete model-based diagnosis cited above and provides schemes that integrate model-based approaches with techniques from fault detection and isolation (FDI)[23, 4] and multi-model adaptive filtering[13, 11, 10]. The hybrid estimation schemes, as well as their FDI and multi-model filtering ancestors, work well whenever the underlying model(s) are 'close' mathematical descriptions of the physical artifact. They can fail severely whenever unforeseen situations occur. Therefore, it is essential to provide models that capture the entire spectrum of possible behaviors/modes whenever we use the hybrid estimate for closed loop control, for instance. Model-based diagnosis, in contrast, does not impose such a strong modeling assumption. Its concept of the *unknown mode* allows diagnosis of systems where no assumption is made about the behavior of one or several components of the system. In this way, it captures unspecified and unforeseen behaviors of the system under investigation. This paper provides an approach to incorporate the concept of an unknown mode into our hybrid estimation scheme[9]. As a result we obtain an estimation capability that can detect unforeseen situations. Furthermore, it allows us to continue estimation on a degraded basis. We achieve this by causal analysis[17, 20], structural analysis[7] and decomposition of the system.

This paper starts with a brief introduction to our hybrid systems modeling and estimation scheme. Upon this foundation, we extend hybrid estimation to incorporate the unknown mode and demonstrate the underlying structural analysis and decomposition task. Finally, an experimental evaluation with computer simulated data for a Martian live support system demonstrates the advantages of this extended hybrid estimation scheme.

2 Hybrid Systems

The hybrid automaton model used throughout this paper is based on [9] and can be seen as a model that merges hidden Markov models (HMM) with continuous discrete-time dynamical system models (we present the model on the level of detail sufficient for this work and refer the reader to the reference cited above for more detail).

2.1 Concurrent Hybrid Automata

Definition 1 A *discrete-time probabilistic hybrid automaton (PHA)* $\mathcal{A}$ is described as a tuple $\langle \mathbf{x}, \mathbf{w}, F, T, \mathcal{X}_d, T_s \rangle$:

- $\mathbf{x}$ denotes the hybrid *state variables* of the automaton[3], composed of $\mathbf{x} = \{x_d\} \cup \mathbf{x}_c$. The discrete variable x_d denotes the *mode* of the automaton and has finite domain $\mathcal{X}_d$. The *continuous state variables* $\mathbf{x}_c$ capture the dynamic evolution of the automaton. $\mathbf{x}$ denotes the *hybrid state* of the automaton, while $\mathbf{x}_c$ denotes the *continuous state*.
- The set of *I/O variables* $\mathbf{w} = \mathbf{u}_d \cup \mathbf{u}_c \cup \mathbf{y}_c$ of the automaton is composed of disjoint sets of discrete input variables $\mathbf{u}_d$ (called *command variables*), continuous *input variables* $\mathbf{u}_c$, and continuous *output variables* $\mathbf{y}_c$.
- $F : \mathcal{X}_d \rightarrow F_{DE} \cup F_{AE}$ specifies the *continuous evolution* of the automaton in terms of *discrete-time difference equations* F_{DE} and *algebraic equations* F_{AE} for each mode $x_d \in \mathcal{X}_d$. T_s denotes the sampling period of the discrete-time difference equations.

[1] Department of Automatic Control, Graz University of Technology, A-8010 Graz, Austria, email: hofbaur@irt.tu-graz.ac.at

[2] MIT Space Systems and AI Laboratories, 77 Massachusetts Ave., Rm. 37-381, Cambridge, MA 02139 USA, email: williams@mit.edu

[3] When clear from context, we use lowercase bold symbols, such as $\mathbf{v}$, to denote a *set* of variables $\{v_1, \ldots, v_l\}$, as well as a *vector* $[v_1, \ldots, v_l]^T$ with components v_i.

- The finite set, T, of *transitions* specifies the probabilistic discrete evolution of the automaton.

Complex systems are modeled as a composition of concurrently operating PHA that represent the individual system components. A *concurrent probabilistic hybrid automata (cPHA)* specifies this composition as well as its interconnection to the outside world:

Definition 2 A *concurrent probabilistic hybrid automaton (cPHA)* $\mathcal{CA}$ is described as a tuple $\langle A, \mathbf{u}, \mathbf{y}_c, \mathbf{v}_s, \mathbf{v}_o, N_x, N_y \rangle$:

- $A = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_l\}$ denotes the finite set of PHAs that represent the components $\mathcal{A}_i$ of the cPHA (we denote the components of a PHA $\mathcal{A}_i$ by $x_{di}, \mathbf{x}_{ci}, \mathbf{u}_{di}, \mathbf{u}_{ci}, \mathbf{y}_{ci}, F_i, \mathcal{X}_{di}$).
- The *input variables* $\mathbf{u} = \mathbf{u}_d \cup \mathbf{u}_c$ of the automaton consists of the sets of discrete input variables $\mathbf{u}_d = \mathbf{u}_{d1} \cup \ldots \cup \mathbf{u}_{dl}$ (command variables) and continuous input variables $\mathbf{u}_c \subseteq \mathbf{u}_{c1} \cup \ldots \cup \mathbf{u}_{cl}$.
- The *output variables* $\mathbf{y}_c \subseteq \mathbf{y}_{c1} \cup \ldots \cup \mathbf{y}_{cl}$ specify the observed output variables of the cPHA.
- The observation process is subject to additive, zero mean Gaussian *sensor noise*. $N_y : \mathcal{X}_d \rightarrow \mathbb{R}^{m \times m}$ specifies the mode dependent[4] disturbance $\mathbf{v}_o$ in terms of the covariance matrix $\mathbf{R} = \text{diag}(r_i)$.
- N_x specifies additive, zero mean Gaussian *disturbances* that act upon the continuous state variables $\mathbf{x}_c = \mathbf{x}_{c1} \cup \ldots \cup \mathbf{x}_{cl}$. $N_x : \mathcal{X}_d \rightarrow \mathbb{R}^{n \times n}$ specifies the mode dependent disturbance $\mathbf{v}_s$ in terms of the covariance matrix $\mathbf{Q}$.

Definition 3 The *hybrid state* $\mathbf{x}_{(k)}$ of a cPHA at time-step k specifies the mode assignment $\mathbf{x}_{d,(k)}$ of the mode variables $\mathbf{x}_d = \{x_{d1}, \ldots, x_{dl}\}$ and the continuous state assignment $\mathbf{x}_{c,(k)}$ of the continuous state variables $\mathbf{x}_c = \mathbf{x}_{c1} \cup \ldots \cup \mathbf{x}_{cl}$.

Interconnection among the cPHA components $\mathcal{A}_i$ is achieved via shared continuous I/O variables $w_c \in \mathbf{u}_{ci} \cup \mathbf{y}_{ci}$ only. Fig. 1 illustrates a simple example composed of 3 PHAs.

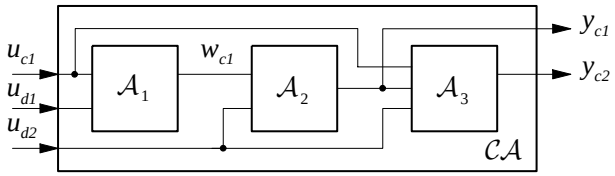


Figure 1. Example cPHA composed of three PHAs

A cPHA specifies a mode dependent discrete-time model for a plant with command inputs $\mathbf{u}_d$, continuous inputs $\mathbf{u}_c$, continuous outputs $\mathbf{y}_c$, mode $\mathbf{x}_d$, continuous state variables $\mathbf{x}_c$ and additive, zero mean Gaussian disturbances $\mathbf{v}_s$, $\mathbf{v}_o$. The discrete-time evolution of $\mathbf{x}_c$ and $\mathbf{y}_c$ is described by the nonlinear system of difference equations (sampling period T_s)

$$\begin{aligned} \mathbf{x}_{c,(k)} &= \mathbf{f}_{(k)}(\mathbf{x}_{c,(k-1)}, \mathbf{u}_{c,(k-1)}) + \mathbf{v}_{s,(k-1)} \\ \mathbf{y}_{c,(k)} &= \mathbf{g}_{(k)}(\mathbf{x}_{c,(k)}, \mathbf{u}_{c,(k)}) + \mathbf{v}_{o,(k)}. \end{aligned} \tag{1}$$

The functions $\mathbf{f}_{(k)}$ and $\mathbf{g}_{(k)}$ are obtained by symbolically solving[5] the set of equations $F_1(x_{d1,(k)}) \cup \ldots \cup F_l(x_{dl,(k)})$ given the mode $\mathbf{x}_{d,(k)} = [x_{d1,(k)}, \ldots, x_{dl,(k)}]^T$.

[4] E.g. sensors can experience different magnitudes of disturbances for different modes.

[5] Our symbolic solver restricts the algebraic equations and nonlinear functions to ones that can be solved explicitly and utilizes a Gröbner Basis approach[3] to derive a set of equations of form (1).

Consider the illustrative cPHA in Fig. 1 with

$$\begin{aligned} \mathcal{A}_1 &= \langle \{x_{d1}\}, \{u_{d1}, u_{c1}, w_{c1}\}, F_1, T_1, \{m_{11}, m_{12}\} \ldots \rangle \\ \mathcal{A}_2 &= \langle \{x_{d2}, x_{c1}\}, \{u_{d2}, w_{c1}, y_{c1}\}, F_2, T_2, \{m_{21}, m_{22}\} \ldots \rangle \\ \mathcal{A}_3 &= \langle \{x_{d3}, x_{c2}, x_{c3}\}, \{u_{d2}, u_{c1}, y_{c1}, y_{c2}\}, F_3, T_3, \{m_{31}\} \ldots \rangle. \end{aligned}$$

F_1, F_2 and F_3 provide for a cPHA mode $\mathbf{x}_{d,(k)} = [m_{11}, m_{21}, m_{31}]^T$ the equations

$$\begin{aligned} F_1(m_{11}) &= \{u_{c1} = 5.0\, w_{c1}\} \\ F_2(m_{21}) &= \{x_{c1,(k)} = 0.8\, x_{c1,(k-1)} + w_{c1,(k-1)}, \\ &\qquad y_{c1} = x_{c1}\} \\ F_3(m_{31}) &= \{x_{c2,(k)} = x_{c3,(k-1)} + y_{c1,(k-1)}, \\ &\qquad x_{c3,(k)} = 0.4\, x_{c2,(k-1)} + 0.5\, u_{c1,(k-1)}, \\ &\qquad y_{c2} = 2.0\, x_{c2} + x_{c3}\}. \end{aligned} \tag{2}$$

This leads to the discrete-time model:

$$\begin{aligned} x_{c1,(k)} &= 0.8\, x_{c1,(k-1)} + 0.2\, u_{c1,(k-1)} + v_{s1,(k-1)} \\ x_{c2,(k)} &= x_{c1,(k-1)} + x_{c3,(k-1)} + v_{s2,(k-1)} \\ x_{c3,(k)} &= 0.4\, x_{c2,(k-1)} + 0.5\, u_{c1,(k-1)} + v_{s3,(k-1)} \\ y_{c1,(k)} &= x_{c1,(k)} + v_{o1,(k)} \\ y_{c2,(k)} &= 2.0\, x_{c2,(k)} + x_{c3,(k)} + v_{o2,(k)} \end{aligned} \tag{3}$$

2.2 Estimation of Hybrid Systems

To detect the onset of subtle failures, it is essential that a monitoring and diagnosis system is able to accurately extract the hybrid state of a system from a signal that may be hidden among disturbances, such as measurement noise. This is the role of a hybrid observer. More precisely:

Hybrid Estimation Problem: Given a cPHA $\mathcal{CA}$, a sequences of observations $\{\mathbf{y}_{c,(0)}, \mathbf{y}_{c,(1)}, \ldots, \mathbf{y}_{c,(k)}\}$ and control inputs $\{\mathbf{u}_{(0)}, \mathbf{u}_{(1)}, \ldots, \mathbf{u}_{(k)}\}$, estimate the most likely hybrid state $\hat{\mathbf{x}}_{(k)}$ at time-step k.

A *hybrid state estimate* $\hat{\mathbf{x}}_{(k)}$ consists of a *continuous state estimate*, together with the associated *mode*. We denote this by the tuple

$$\hat{\mathbf{x}}_{(k)} := \langle \mathbf{x}_{d,(k)}, \hat{\mathbf{x}}_{c,(k)}, \mathbf{P}_{(k)} \rangle,$$

where $\hat{\mathbf{x}}_{c,(k)}$ specifies the mean and $\mathbf{P}_{(k)}$ the covariance for the continuous state variables $\mathbf{x}_c$. The likelihood of an estimate $\hat{\mathbf{x}}_{(k)}$ is denoted by the *hybrid belief-state* $h_{(k)}[\hat{\mathbf{x}}]$.

We perform hybrid estimation as extended version of HMM-style belief-state update that accounts for the influence of the continuous dynamics upon the system's discrete modes. A major difference between hybrid estimation and an HMM-style belief-state update, as well as multi-model estimation, is, however, that hybrid estimation tracks a set of trajectories, whereas standard belief-state update and multi-model estimation aggregate trajectories which share the same mode. This difference is reflected in the first of the following two recursive functions which define our hybrid estimation scheme:

$$h_{(\bullet k)}[\hat{\mathbf{x}}_i] = P_{\mathcal{T}}(\mathbf{m}_i | \hat{\mathbf{x}}_{j,(k-1)}, \mathbf{u}_{d,(k-1)}) h_{(k-1)}[\hat{\mathbf{x}}_j] \tag{4}$$

$$h_{(k)}[\hat{\mathbf{x}}_i] = \frac{h_{(\bullet k)}[\hat{\mathbf{x}}_i] P_{\mathcal{O}}(\mathbf{y}_{c,(k)} | \hat{\mathbf{x}}_{i,(k)}, \mathbf{u}_{c,(k)})}{\sum_j h_{(\bullet k)}[\hat{\mathbf{x}}_j] P_{\mathcal{O}}(\mathbf{y}_{c,(k)} | \hat{\mathbf{x}}_{j,(k)}, \mathbf{u}_{c,(k)})} \tag{5}$$

$h_{(\bullet k)}[\hat{\mathbf{x}}_i]$ denotes an intermediate hybrid belief-state, based on transition probabilities only. Hybrid estimation determines for each

$\hat{\mathbf{x}}_{j,(k-1)}$ at the previous time-step $k-1$ the possible transitions, thus specifying candidate successor states to be tracked. Consecutive filtering provides the new hybrid state $\hat{\mathbf{x}}_{i,(k)}$ and adjusts the hybrid belief-state $h_{(k)}[\hat{\mathbf{x}}_i]$ based on the hybrid probabilistic observation function $P_{\mathcal{O}}(\mathbf{y}_{c,(k)}|\hat{\mathbf{x}}_{i,(k)}, \mathbf{u}_{c,(k)})$. The estimate $\hat{\mathbf{x}}_{j,(k)}$ with the highest belief-state $h_{(k)}[\hat{\mathbf{x}}_j] = \max_i(h_{(k)}[\hat{\mathbf{x}}_i])$ is taken as the hybrid estimate at time-step k.

Tracking all possible trajectories of the system is almost always intractable because the number of trajectories becomes too large after only a few time-steps. In [9] we present an approximative anytime anyspace algorithm that copes with the exponential growth, as well as the large number of modes in a typical concurrent hybrid automaton model.

Hybrid estimation and other multi-model estimation schemes have in common that they require models that are 'close' mathematical descriptions of the system. They can fail severely whenever unforeseen, i.e. unmodeled, situations occur. As a consequence, we have to provide models for all operational modes as well as an exhaustive set of models for possible failure modes. Providing all possible failure models can be problematic even under the assumption of an exhaustive failure mode effect analysis (FMEA). For instance, consider an incipient fault in a servo valve that causes the valve to drift off its nominal opening value. The drift (positive, negative, slow, fast...) is subject to the fault. It is surely difficult to provide a mathematical model with the correct parameter values that captures all possible drift situations. Nor is it helpful to introduce a sufficiently large set of modes that captures possible situations of the drift fault as this would introduce additional complexity for hybrid estimation by increasing the number of modes unnecessarily.

This requirement of hybrid mode estimation is in contrast to discrete model-based diagnosis schemes, such as GDE (e.g. [5, 6, 19]). Model-based diagnosis deduces the possible mode of the system based on nominal models, and few specified fault models only. The onset of possible fault scenarios are covered by the so called *unknown mode* which does not impose any constraints on the system's variables.

The next section provides an approach that systematically incorporates the concept of the unknown mode into our hybrid estimation scheme.

3 Estimation with Unknown Modes

The estimation scheme [9] requires a fully specified mode assignment $\mathbf{x}_{di,(k)}$ for each candidate trajectory that is tracked in the course of hybrid estimation. Only a fully specified mode allows us to deduce the mathematical model (1) for the overall system. This model is the basis for the dynamic filter (e.g. extended Kalman filter) that is used in the course of hybrid estimation.

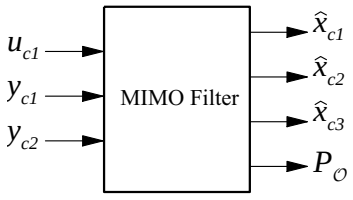


Figure 2. MIMO filter (e.g. extended Kalman filter) for the cPHA example

For our illustrative 3 component example introduced above this would mean that hybrid estimation calculates a multi-input multi-output (MIMO) filter (see Fig. 2) for mode $\mathbf{x}_{di,(k)} = [m_{11}, m_{21}, m_{31}]^T$ based on the mathematical model (3). This filter provides the hybrid state estimate $\hat{\mathbf{x}}_{i,(k)}$ as well as the value for the hybrid probabilistic observation function $P_{\mathcal{O}}(\mathbf{y}_{c,(k)}|\hat{\mathbf{x}}_{i,(k)}, \mathbf{u}_{c,(k)})$ for the hybrid estimator (see Appendix A for the extended Kalman filter estimation details).

Let us assume the mode $\mathbf{x}_{di,(k)} = [?, m_{21}, m_{31}]^T$ which specifies that component 1 ($\mathcal{A}_1$) is in *unknown mode*. A component in unknown mode imposes no constraints (equations) among its variables (u_{c1} and the internal variable w_{c1}, in our case). As a consequence, we cannot deduce an overall mathematical model of the form (1) and fail to provide the basis for the hybrid estimation scheme, the MIMO filter for mode $\mathbf{x}_{di,(k)} = [?, m_{21}, m_{31}]^T$.

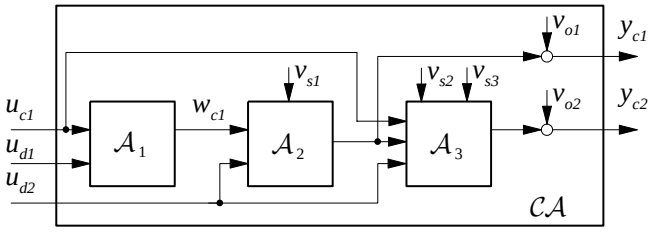


Figure 3. Example cPHA with explicit noise inputs

However, a close look on the PHA interconnection (Fig. 3 - the figure extends Fig. 1 by including the implicit noise inputs, as well as indicating the causality for the internal I/O variables) reveals that we can still estimate component 3 by its observed output y_{c2} and the observation y_{c1} as a substitute for the value of its input. This intuitive approach utilizes a decomposition of the cPHA as shown in Fig. 4.

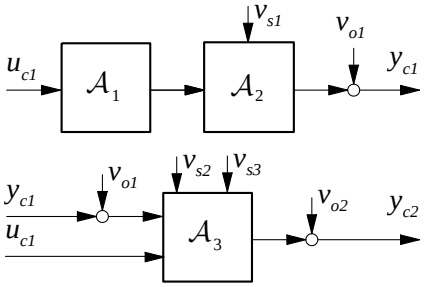


Figure 4. Decomposed cPHA

The decomposition allows us to treat the concurrent parts of the system independently and calculate a *filter cluster* consisting of 2 independent filters. However, when calculating the individual filters for the cluster, we have to take into account that we use the *measurement* of the input to the third component (y_{c1}) in replacement to its true value. This can be interpreted as having additional additive noise at the component's input as indicated in Fig. 4. The following modification of the covariance matrix $\mathbf{Q}_3$ for the state variables of $\mathcal{A}_3$ takes this into account:

$$\tilde{\mathbf{Q}}_3 = \mathbf{b}_3 r_1 \mathbf{b}_3^T + \mathbf{Q}_3, \tag{6}$$

where r_1 denotes the variance of disturbance v_{o1} and $\mathbf{b}_3 = [0,\ 1]^T$

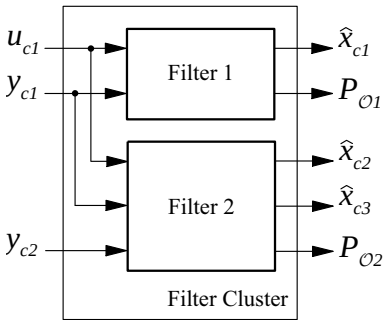


Figure 5. Decomposed filter

denotes the input vector[6] of $\mathcal{A}_3$ with respect to y_{c1}.

A filter cluster consisting of extended Kalman filters and the MIMO extended Kalman filter are interchangeable as they provide the same expected value for the continuous state ($E(\hat{\mathbf{x}}_c)$) whenever the mode of the automaton is fully specified. However, the decomposed filter has the advantage that the probabilistic observation function $P_{\mathcal{O}}$ of the overall system is given by

$$P_{\mathcal{O}} = \prod_j P_{\mathcal{O}j}, \tag{7}$$

where $P_{\mathcal{O}j}$ denotes the probabilistic observation function of the j'th filter in the filter cluster.

This factorization of the probabilistic observation function allows us to calculate an upper bound for $P_{\mathcal{O}}$ whenever one or more components of the system are in unknown mode. We simply take the product over the remaining filters in the cluster. This is equivalent with considering the upper bounds of the inequalities $P_{\mathcal{O}j} \leq 1$ for each unknown filter j. In our example with unknown component $\mathcal{A}_1$ this would mean:

$$P_{\mathcal{O}} \leq P_{\mathcal{O}2},$$

where $P_{\mathcal{O}2}$ denotes the observation function for the filter that estimates the continuous state of component $\mathcal{A}_3$.

The following subsection provides a graph-based approach for filer cluster deduction that grounds the informally introduced decomposition on a more versatile basis.

3.1 System Decomposition and Filter Cluster Calculation

Starting point for the decomposition of the system for a cPHA mode $\mathbf{x}_d$ is the set of equations

$$F_1(x_{d1,(k)}) \cup \ldots \cup F_l(x_{dl,(k)}) =: \mathcal{F}(\mathbf{x}_d), \tag{8}$$

where $F_j(x_{dj,(k)})$ returns the appropriate set of equations for a component $\mathcal{A}_i$ whenever $x_{dj,(k)} \in \mathcal{X}_{dj}$ or the empty set whenever the component is in unknown mode, i.e. $x_{dj,(k)} =?$. Although we still have to solve the set of equations to arrive at the mathematical model of form (1) we can interpret the set of equations (8) as the *raw model* for the system given mode $\mathbf{x}_d$. The following decomposition performs a structural analysis of the raw model-based on causal analysis[17, 20], structural observability analysis[7] and graph decomposition[1].

A cPHA model does not impose a fixed causal structure that specifies directionality of automaton interconnections. Causality is implicitly specified by the set of equations. This increases the expressiveness of the modeling framework but requires us to perform a causal analysis of the raw model (8) as a first step. The deduction of the causal dependencies is done by applying the bipartite-matching based algorithm presented in [17]. The resulting directed graph records the causal dependencies among the variables of the system (Fig. 6 shows the graph for the the illustrative 3 PHA example). Each vertex of the graph represents one equation $e_i \in \mathcal{F}$

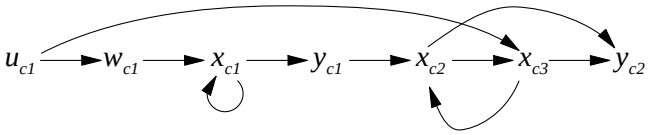


Figure 6. Causal graph for the cPHA example

or an exogenous variable specification (e.g. u_{c1}) and is labeled by its dependent variable which also specifies the outgoing edge (in the following, we will use the variable name to refer to the corresponding vertex in the graph). Vertices without incoming edges specify the *exogenous* variables.

Definition 4 A *causal graph* of a cPHA $\mathcal{CA}$ at a mode $\mathbf{x}_d$ is a directed graph that records the causal dependencies among the variables $v \in \bigcup_i \mathbf{x}_{ci} \cup \mathbf{u}_{ci} \cup \mathbf{y}_{ci}$ of $\mathcal{CA}$. We denote the causal graph by $\mathcal{CG}(\mathcal{CA}, \mathbf{x}_d)$ and sometimes omit arguments where no confusion seems likely.

Goal of our analysis is to obtain a set of independent subsystems that utilize observed variables as virtual inputs. Therefore, we slice the graph at observed variable vertices with outgoing edges, insert a new vertex to represent a virtual input and re-map the sliced outgoing edges to this vertex. Fig. 7 demonstrates this re-mapping for the causal graph of Fig. 6. The observed variables are y_{c1} and y_{c2}. Only the vertex with dependent variable y_{c1} has an outgoing edge, thus we slice the graph at $y_{c1} \rightarrow x_{c2}$ and re-map the edge to the virtual input uy_{c1}.

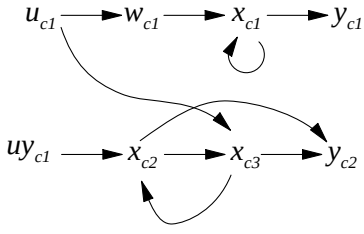


Figure 7. Remapped causal graph for the cPHA example

A dynamic filter (e.g. extended Kalman filter) can only estimate the observable part of the model. Therefore, it is essential to perform

[6] In the general case, we have to calculate $\mathbf{b}_j$ for a cPHA component $\mathcal{A}_j$ and observed inputs $\mathbf{uy}_c$ by linearization, more specifically: $\mathbf{b}_{j,(k)} = \partial \mathbf{f}_j / \partial \mathbf{uy}_c|_{\hat{\mathbf{x}}_{cj,(k-1)}, \mathbf{u}_{cj,(k-1)}}$, where $\mathbf{f}_j$ denotes the right-hand side of the difference equation for component $\mathcal{A}_j$, $\mathbf{uy}_c$ refers to the observed variables that are used as inputs to the component (i.e. $\mathbf{uy}_c \subset \mathbf{y}_c$) and $\hat{\mathbf{x}}_{cj,(k-1)}$ as well as $\mathbf{u}_{cj,(k-1)}$ represent the state estimate and the continuous input for component $\mathcal{A}_j$ at the previous time-step, respectively.

an observability analysis prior calculating the filter so that non observable parts of the model are excluded. We perform this analysis on a structural basis[7].

Definition 5 We call a variable v of a cPHA $\mathcal{CA}$ at mode $\mathbf{x}_d$ *structurally observable (SO)* whenever it is directly observed, i.e. $v \in \mathbf{y}_c$, or there exists at least one path in the causal graph $\mathcal{CG}(\mathcal{CA}, \mathbf{x}_d)$ that connects the variable z to an output variable $y_c \in \mathbf{y}_c$ of $\mathcal{CA}$.

A filter estimates the state variables $\mathbf{x}_c$ of a dynamic system based on observations $\mathbf{y}_c$ and the inputs $\mathbf{u}_c$ that act upon the state variables $\mathbf{x}_c$. The required knowledge about the inputs $\mathbf{u}_c$ indicates that the structural observability criteria is not yet sufficient to determine the submodel for estimation. We have to make sure, that no unknown exogenous input influences a variable. To illustrate this, consider again the 3 PHA example with mode $\mathbf{x}_d = [?, m_{21}, m_{31}]^T$. Component 1 in unknown mode omits the equation that relates the variables u_{c1} and w_{c1}. This leads to a causal graph $\tilde{\mathcal{CG}}$ (Fig. 8), where w_{c1} is labeled as exogenous (no incoming edges). This unknown exogenous input influences the state variable x_{c1} and, as a consequence, prevents us from estimating it!

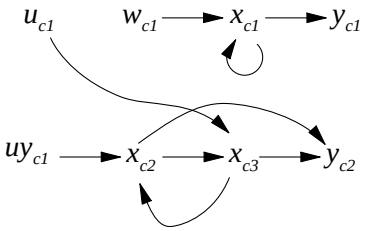


Figure 8. Remapped causal graph for the cPHA example with unknown component $\mathcal{A}_1$

We extend our structural analysis of the causal graph by the following criteria:

Definition 6 We call a variable v of a cPHA $\mathcal{CA}$ at mode $\mathbf{x}_d$ *structurally determined (SD)* whenever it is an input variable of the automaton, i.e. $v \in \mathbf{u}_c$, or there does not exist a path in the causal graph $\mathcal{CG}(\mathcal{CA}, \mathbf{x}_d)$ that connects an exogenous variable $u_e \notin \mathbf{u}_c$ with v.

Furthermore, it is helpful to eliminate loops in the causal graph prior checking variables against both structural criteria. For this purpose, we calculate the *strongly connected components* of the causal graph[1].

Definition 7 A *strongly connected component (SCC)* of the causal graph $\mathcal{CG}$ is a maximal set $\mathcal{SCC}$ of variables in which there is a path from any one variable in the set to another variable in the set.

Fig. 9 shows the remapped causal graph for the 3 PHA example after grouping variables into strongly connected components.

The strong interconnection among variables in an SCC implies that:

1. Structural observability of variables in an SCC follows directly from structural observability of at least one variable in the SCC.

[7] Throughout the paper we assume that loss of observability is caused by a structural defect of the model. Otherwise, it is necessary to perform an additional numerical observability test [18] as structural observability only provides a *necessary* condition for observability.

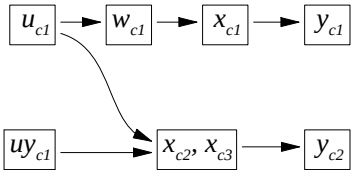


Figure 9. Causal SCC graph for cPHA example

2. A variable in an SCC is structurally determined, if and only if all variables in the SCC are structurally determined.

As a consequence, we can apply our structural analysis to strongly connected components directly and operate on the SCC graph, i.e a causal graph without loops. The analysis of a strongly connected component with respect to structural observability and structural determination (SOD) can be outlined as follows:

---

function determine-SOD-of-SCC($\mathcal{SCC}, \mathbf{u}_c, k$)
  when SOD-undetermined?($\mathcal{SCC}$)
    if exogenous?($\mathcal{SCC}$)
      then $v_i \leftarrow$ independent-var($\mathcal{SCC}$)
        if $v_i \in \mathbf{u}_c$ **then** SD($\mathcal{SCC}$) $\leftarrow$ **True**
          else SD($\mathcal{SCC}$) $\leftarrow$ **False**
      else $\mathcal{V} \leftarrow$ uplink-SCCs($\mathcal{SCC}$)
        loop for $\mathcal{SCC}_i$ in $\mathcal{V}$
          do determine-SOD-of-SCC($\mathcal{SCC}_i, \mathbf{u_c}, k$)
        SO($\mathcal{SCC}$) $\leftarrow$ **True**
        SD($\mathcal{SCC}$) $\leftarrow$ all-uplink-SCCs-are-SD?($\mathcal{V}$)
        cluster-index($\mathcal{SCC}$) $\leftarrow k \cup$ cluster-indices($\mathcal{V}$)
    SOD-determined($\mathcal{SCC}$) $\leftarrow$ **True**
  return Nil

Our structural analysis algorithm determines structural observability and determination (SOD) of a variable by traversing the SCC graph backwards from the observed variables towards the inputs. In the course of this analysis we label non-exogenous strongly connected components with an index that refers to their cluster membership. This indexing scheme allows us to cluster the variables into non-overlapping clusters with respect to the observed variables. The direct relation between a variable, its determining equation, and the cPHA component that specified this equation leads to the component clusters sought. The structural analysis can be summarized as follows:

---

function component-clustering($\mathcal{CA}, \mathbf{x}_d$)
  returns a set of cPHA component clusters
  $\mathbf{y}_c \leftarrow$ observed-vars($\mathcal{CA}$)
  $\tilde{\mathcal{CG}} \leftarrow$ remap-causal-graph($\mathcal{CG}(\mathcal{CA}, \mathbf{x}_d), \mathbf{y}_c$)
  $\mathbf{u}_c \leftarrow$ virtual-inputs($\tilde{\mathcal{CG}}$) $\cup$ input-vars($\mathcal{CA}$)
  $\mathcal{CG}_{SCC} \leftarrow$ strongly-connected-component-graph($\tilde{\mathcal{CG}}$)
  $k \leftarrow 0$
  loop for $\mathcal{SCC}_i$ in output-SCCs($\mathcal{CG}_{SCC}, \mathbf{y}_c$)
    do determine-SOD-of-SCC($\mathcal{SCC}_i, \mathbf{u}_c, k$)
      $k \leftarrow k + 1$
  graph-clusters $\leftarrow$ get-SOD-SSC-clusters($\mathcal{CG}_{SCC}$)
  return automaton-clusters($\mathcal{CA}$, *graph-clusters*)

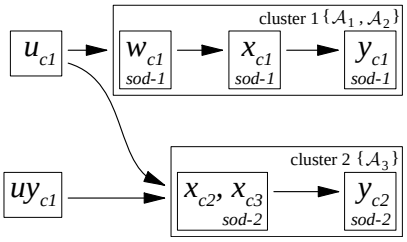


Figure 10. Labeled and partitioned causal SCC graph for the 3 cPHA example

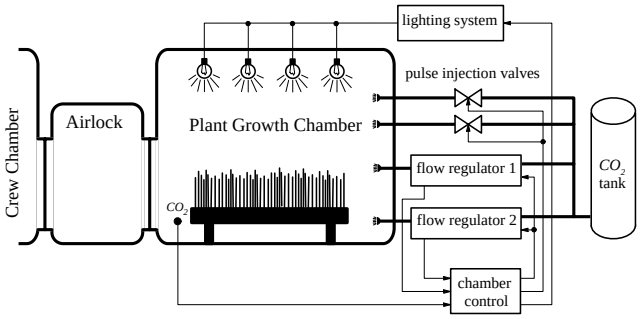


Figure 11. BIO-Plex plant growth chamber

Each component cluster defines the observable and determined raw model for a subsystem of the cPHA. This raw model can be solved symbolically and provides the nonlinear system of difference equations (a model similar to (1), but with the additional virtual inputs) that is the basis for the corresponding filter in the filter cluster. In this way we exclude the unobservable and/or undetermined parts of the overall system from estimation.

Whenever a state variable x_{cj} becomes unobservable and/or undetermined (e.g. due to a mode change) during hybrid estimation, we hold the value for the mean at its last known estimate $\hat{x}_{cj}$ and increase its variance $\sigma_j^2 = p_{jj}$ by a constant factor at each hybrid estimation step. This reflects a continuously decreasing confidence in the estimate $\hat{x}_{cj}$ and allows us to restart estimation whenever the variable becomes observable and determined again[8].

4 Example - BIO-Plex

Our application is the BIO-Plex Test Complex at NASA Johnson Space Center, a five chamber facility for evaluating biological and physiochemical Martian life support technologies. It is an artificial, biosphere-type, closed environment, which must robustly provide all the air, water, and most of the food for a crew of four without interruption. Plants are grown in plant growth chambers, where they provide food for the crew, and convert the exhaled CO_2 into O_2. In order to maintain a closed-loop system, it is necessary to control the resource exchange between the chambers without endangering the crew. For the scope of this paper, we restrict our evaluation to the sub-system dealing with CO_2 control in the plant growth chamber (PGC), shown in Fig. 11.

The system is composed of several components, such as redundant flow regulators (FR1, FR2) that provide continuous CO_2 supply, redundant pulse injection valves (PIV1, PIV2) that provide a means for increasing the CO_2 concentration rapidly, a lighting system (LS) and the plant growth chamber (PGC), itself. The control system maintains a plant growth optimal CO_2 concentration of 1200 ppm during the day phase of the system (20 hours/day).

Hybrid estimation schemes are key to tracking system operational modes, as well as, detecting subtle failures and performing diagnoses. For example, we simulate a failure of the second flow regulator. The regulator becomes off-line and drifts slowly towards its positive limit. This fault situation is difficult to capture by an explicit fault model as we do not know, in advance, whether the regulator drifts towards its postitive or negative limit, nor do we know the magnitude of the drift. A fault of this type, which develops slowly and whose symptom is hidden among the noise in the system is a typical candidate for our unknown-mode detection capability. However, we also provide explicit failure models that describe typical situations. For example, the PGC has 4 plant trays with one illumination bank for each tray. A black out of one illumination bank can be interpreted as a 25% loss in light intensity. This situation can be modeled explicitly by a dynamical model that takes this reduced light intensity into account.

In the following we describe the outcome of a simulated experiment where the flow regulator fault with drifting symptom is injected at time point $k = 700$ and an additional light fault, that harms one of the four illumination banks, is injected at $k = 900$. The faults are 'repaired' at $k = 1100$ and $k = 1300$ for the flow regulator fault and the lighting fault, respectively. This experiment illustrates unknown mode detection and recovery from it, nominal failure mode detection, and the multiple fault detection capability of our approach.

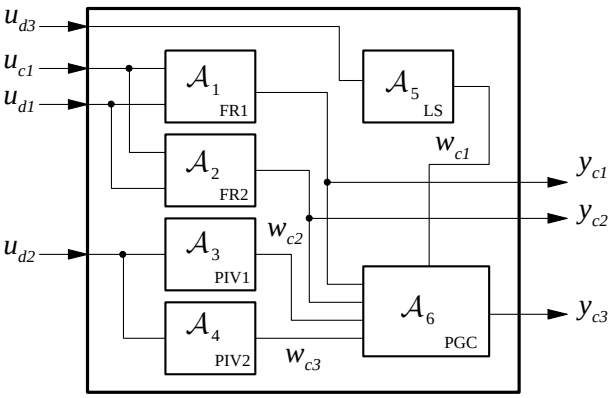


Figure 12. BIO-Plex cPHA model

The simulated data is gathered from the execution of a refined subset of NASA's JSC's CONFIG model for the BIO-Plex system[12]. Hybrid estimation utilizes a cPHA model that consists of 6 components as shown in Fig. 12. To illustrate the complexity of the hybrid estimation problem we should note, that the concurrent automaton has approximately $5^6 \approx 15000$ modes. Each mode describes the dynamic evolution of the chamber system by a third order system of difference equations. For example, the nominal operational condition for plant growth is characterized by the mode

[8] Whenever a state variable x_{cj} is directly observed we also can utilize an alternative approach suggested in [15] that restarts the estimator with the observed value, thus improving the observer convergence time.

$\mathbf{x}_d = [m_{r2}, m_{r2}, m_{v1}, m_{v1}, m_{l2}, m_{p2}]$, where m_{r2} characterizes an partially open flow regulator, m_{v1} a closed pulse injection valve, m_{l2} 100% light on, and m_{p2} plant growth mode at 1200 ppm, respectively. This mode specifies the raw model:

$$\begin{aligned}
F_1(m_{r2}) &= \{x_{c1,(k)} = 0.5\, u_{c1,(k-1)},\ y_{c1} = x_{c1}\} \\
F_2(m_{r2}) &= \{x_{c2,(k)} = 0.5\, u_{c1,(k-1)},\ y_{c2} = x_{c2}\} \\
F_3(m_{v1}) &= \{w_{c2} = 0.0\} \\
F_4(m_{v1}) &= \{w_{c3} = 0.0\} \\
F_5(m_{l2}) &= \{w_{c1} = 1204.0\} \\
F_6(m_{p2}) &= \{x_{c3,(k)} = x_{c3,(k-1)} + 20.163\cdot \\
&\qquad [-1.516 \cdot 10^{-4} f_1(w_{c1,(k-1)}) f_2(x_{c3,(k-1)}) + \\
&\qquad y_{c1,(k-1)} + y_{c2,(k-1)} + w_{c1,(k-1)} + w_{c2,(k-1)}], \\
&\qquad y_{c3} = x_{c3}\},
\end{aligned} \tag{9}$$

where f_1 and f_2 denotes

$$\begin{aligned}
f_1(w_{c1}) &:= -7.615 + 0.111\, w_{c1} - 2.149 \cdot 10^{-5}\, w_{c1}^2 \\
f_2(x_{c3}) &:= 72.0 - 78.89\, e^{-x_{c3}/400.0}.
\end{aligned} \tag{10}$$

$x_{c1,(k)}$ and $x_{c2,(k)}$ denote the gas flow ([g/min]) of flow regulator 1 and 2, respectively and $x_{c3,(k)}$ denotes the CO_2 gas concentration ([ppm]) in the plant growth chamber. $w_{c1,(k)}$ and $w_{c2,(k)}$ denote the gas flow ([g/min]) of the pulse injection valves and $w_{c3,(k)}$ denotes the photosynthetic photon flux ([μ-mol/m^{2}s]) of the lights above the plant trays. The nonlinear expression

$$-1.516 \cdot 10^{-4} f_1(w_{c1,(k-1)}) f_2(x_{c3,(k-1)})$$

approximates the CO_2 gas production [g/min] due to photosynthesis according to the CO_2 gas concentration and chamber illumination[12]. This raw model defines a third order system of discrete-time difference equations with sampling period $Ts = 1$ [min]:

$$\begin{aligned}
x_{c1,(k)} &= 0.5\, u_{c1,(k-1)} + v_{s1,(k-1)} \\
x_{c2,(k)} &= 0.5\, u_{c1,(k-1)} + v_{s2,(k-1)} \\
x_{c3,(k)} &= x_{c3,(k-1)} + 20.163[-1.041 + \\
&\qquad 1.141 e^{-x_{c3,(k)}/400.0} + x_{c1,(k-1)} + x_{c2,(k-1)}] + v_{s3,(k-1)} \\
y_{c1,(k)} &= x_{c1,(k)} + v_{o1,(k)} \\
y_{c2,(k)} &= x_{c2,(k)} + v_{o2,(k)} \\
y_{c2,(k)} &= x_{c3,(k)} + v_{o3,(k)},
\end{aligned} \tag{11}$$

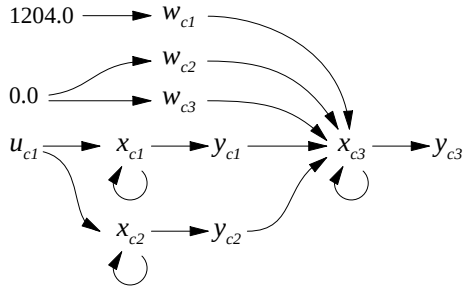


Figure 13. Causal graph of the BIO-Plex cPHA raw model (9)

The causal graph (Fig. 13) of the raw model (9) leads to the decomposition of the system as shown in Fig. 14 (our implementation of the causal analysis and decomposition algorithms treats constant values, such as the value 1204.0 for the photosynthetic photon flux, as known exogenous inputs with constant value). The decomposition of the model leads to a filter cluster with 3 extended Kalman filters - one for each flow regulator and one for the remaining system (pulse injection valves, lighting system and plant growth chamber). This enables us to estimate the mode and continuous state of the flow regulators independent of the remaining system. As a consequence, an unknown mode in a flow regulator does not cause any implications on the estimation of the remaining system.

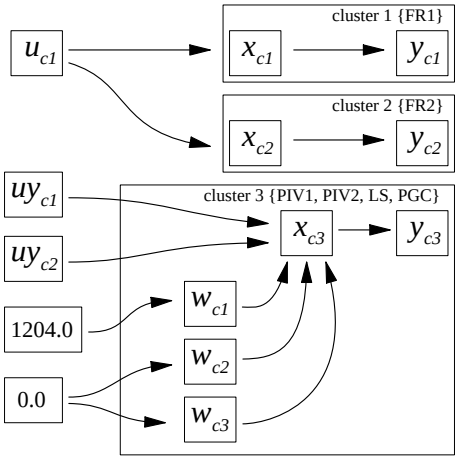


Figure 14. Partitioned causal SCC graph of the BIO-Plex cPHA model

Fig. 15 shows the continuous input (control signal) u_{c1}, observed flow rates for flow regulator 1 and 2 and the CO_2 concentration for the experiment. Both flow regulators provide half of the requested gas injection rate up to $k = 700$. At this time point, the second flow regulator starts to slowly drift towards its positive limit which it will reach at approximately $k = 800$. The camber control system reacts immediately and lowers the control signal in order to keep the CO_2 concentration at the requested 1200 ppm concentration. This transient behavior causes a slight bump in the CO_2 concentration as shown in Fig. 15-b. Our hybrid mode estimation system detects this unmodeled fault at $k = 727$ and declares flow regulator 2 to be in an unknown mode (we indicate the unknown mode by the mode number 0 in Fig. 16). The flow regulator mode *stuck-open* (m_{r5}) be-

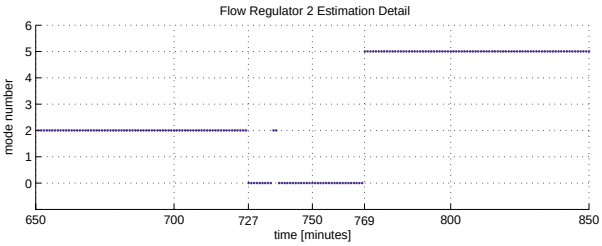
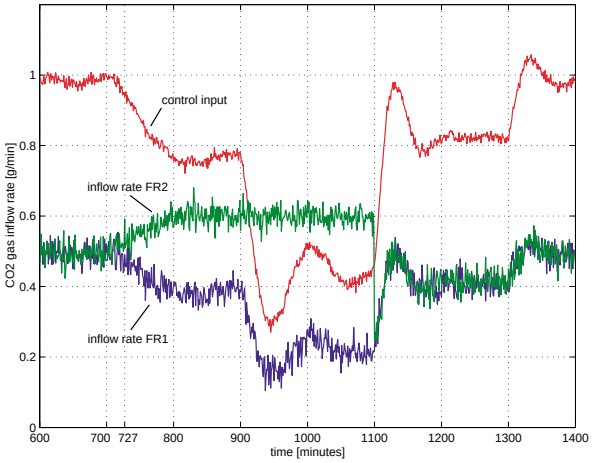
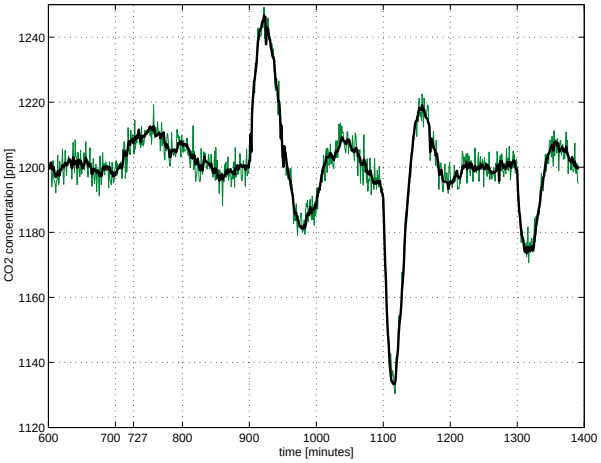


Figure 16. Mode estimate detail for flow regulator 2

comes more and more likely as the regulator drifts towards its open position. Hybrid mode estimation prefers this mode as symptom ex-



(a) Control input u_c and measured CO_2 input flow rates



(b) CO_2 level in PGC (measurement - gray/green, estimate - black)

Figure 15. Observed data and continuous estimation of the CO_2 concentration in plant growth chamber

planation from $k = 769$ onwards, although flow regulator 2 goes into saturation a little bit later at $k = 800$.

The light fault at $k = 900$ is detected almost instantly at $k = 904$ (m_{l4}). This good discrimination among the pre-specified modes (failure and nominal) is further demonstrated at the termination points of the faults. Repairs of the flow regulator 2 and the lighting system are detected immediately at $k = 1101$ and $k = 1301$, respectively. Fig. 17 shows the mode estimation result for the lighting system and flow regulator 2 over the entire experiment horizon.

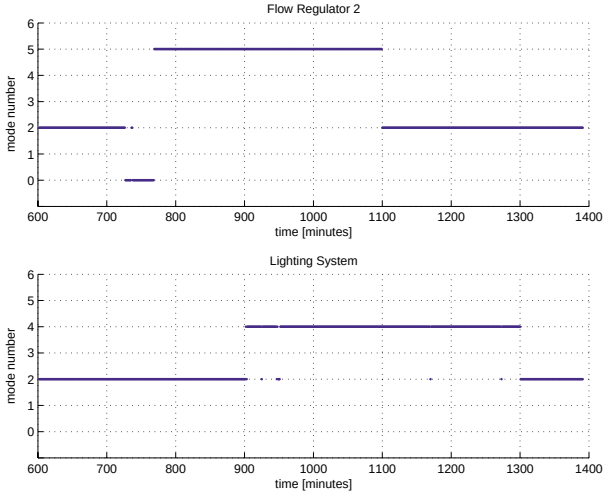


Figure 17. Mode estimates for flow regulator 2 and lighting system

5 Implementation and Discussion

The implementation of our hybrid estimation scheme extends previous work on hybrid estimation [9] and is written in Common LISP. The hybrid estimator uses a cPHA description and performs decomposition and estimation, as outlined above. Decomposition is done on-line according to the mode hypotheses that are tested in the course of hybrid estimation. In general, it can be assumed that the the mode in the system evolves on a lower rate than the hybrid estimation rate, which operates on the sampling period T_s. Therefore, we cache recent decompositions and their corresponding filters for re-use as a compromise between a-priori calculation (space complexity) and pure on-line deduction (time complexity).

Optimized model-based estimation schemes, such as Livingstone[22], utilize *conflicts* to focus the underlying search operation. A conflict is a (partial) mode assignment that makes a hypothesis very unlikely. This requires a more general treatment of unknown modes compared to the filter decomposition task introduced above. The decompositional model-based learning system Moriarty[21] introduced continuous variants of conflicts, so-called *dissents*. We are currently reformulating these dissents for hybrid systems and investigate their incorporation to improve the underlying search scheme. This will lead to an overall framework that unifies our previous work on Livingstone, Moriarty and hybrid estimation.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[2] B. Anderson and J. Moore, *Optimal Filtering*, Information and System Sciences Series, Prentice Hall, 1979.

[3] *Gröbner Bases and Applications*, eds., B. Buchberger and F. Winkler, Cambridge Univ. Press, 1998.

[4] J. Chen and R. Patton, *Robust Model-Based Fault Diagnosis for Dynamic Systems*, Kluwer, 1999.

[5] J. de Kleer and B. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).

[6] J. de Kleer and B. Williams, 'Diagnosis with behavioral modes', in *Proceedings of IJCAI-89*, pp. 1324–1330, (1989).

[7] A. Gehin, M. Assas, and M. Staroswiecki, 'Structural analysis of sys-

tem reconfigurability', in *Preprints of the 4th IFAC SAFEPROCESS Symposium*, volume 1, pp. 292–297, (2000).
[8] *Readings in Model-Based Diagnosis*, eds., W. Hamscher, L. Console, and J. de Kleer, Morgan Kaufmann, San Mateo, CA, 1992.
[9] M. Hofbaur and B.C. Williams, 'Mode estimation of probabilistic hybrid systems', in *Hybrid Systems: Computation and Control, HSCC 2002*, eds., C.J. Tomlin and M.R. Greenstreet, volume 2289 of *Lecture Notes in Computer Science*, 253–266, Springer Verlag, (2002).
[10] P. Li and V. Kadirkamanathan, 'Particle filtering based likelyhood ratio approach to fault diagnosis in nonlinear stochastic systems', *IEEE Transactions on Systems, Man, and Cybernetics - Part C*, **31**(3), 337–343, (2001).
[11] X.R. Li and Y. Bar-Shalom, 'Multiple-model estimation with variable structure', *IEEE Transactions on Automatic Control*, **41**, 478–493, (1996).
[12] J. T. Malin, L. Fleming, and T. R. Hatfield, 'Interactive simulation-based testing of product gas transfer integrated monitoring and control software for the lunar mars life support phase III test', in *SAE 28th International Conference on Environmental Systems, Danvers MA*, (July, 1998).
[13] P. Maybeck and R.D. Stevens, 'Reconfigurable flight control via multiple model adaptive control methods', *IEEE Transactions on Aerospace and Electronic Systems*, **27**(3), 470–480, (1991).
[14] S. McIlraith, 'Diagnosing hybrid systems: a bayseian model selection approach', in *Proceedings of the 11th International Workshop on Principles of Diagnosis (DX00)*, pp. 140–146, (June 2000).
[15] P.J. Mosterman and G. Biswas, 'Building hybrid observers for complex dynamic systems using model abstractions', in *Hybrid Systems: Computation and Control (HSCC'99)*, eds., F. Vaandrager and J. Schuppen, volume 1569 of *LNCS*, 178–192, Springer Verlag, (1999).
[16] S. Narasimhan and G. Biswas, 'Efficient diagnosis of hybrid systems using models of the supervisory controller', in *Proceedings of the 12th International Workshop on Principles of Diagnosis (DX01)*, pp. 127–134, (March 2001).
[17] P. Nayak, *Automated Modelling of Physical Systems*, Lecture Notes in Artificial Intelligence, Springer, 1995.
[18] E. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, Springer, New York, Berlin, Heidelberg, 2 edn., 1998.
[19] P. Struss and O. Dressler, 'Physical negation: Integrating fault models into the general diagnostic engine', in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, pp. 1318–1323, (1989).
[20] L. Travé-Massuyès and R. Pons, 'Causal ordering for multiple mode systems', in *Proceedings of the 11th International Workshop on Qualitative Reasoning (QR97)*, pp. 203–214, (1997).
[21] B. Williams and B. Millar, 'Decompositional, model-based learning and its analogy to diagnosis', in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, (1998).
[22] B. Williams and P. Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, (1996).
[23] A. S. Willsky, 'A survey of design methods for failure detection in dynamic systems', *Automatica*, **12**(6), 601–611, (1974).
[24] F. Zhao, X. Koutsoukos, H. Haussecker, J. Reich, and P. Cheung, 'Distributed monitoring of hybrid systems: A model-directed approach', in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01)*, pp. 557–564, (2001).

Acknowledgments

In part supported by NASA under contract NAG2-1388.

A Extended Kalman Filter

The disturbances and imprecise knowledge about the initial state $\mathbf{x}_{c,(0)}$ make it necessary to estimate the state by its mean $\hat{\mathbf{x}}_{c,(k)}$ and covariance matrix $\mathbf{P}_{(k)}$. We use an extended Kalman filter[2] for this purpose, which updates its current state, like an HMM observer, in two steps. The first step uses the model to predict mean for the state $\hat{\mathbf{x}}_{c,(\bullet k)}$ and its covariance $\mathbf{P}_{(\bullet k)}$, based on the previous estimate $\langle \hat{\mathbf{x}}_{c,(k-1)}, \mathbf{P}_{(k-1)} \rangle$, and the control input $\mathbf{u}_{c,(k-1)}$:

$$\hat{\mathbf{x}}_{c,(\bullet k)} = \mathbf{f}(\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{u}_{c,(k-1)}) \quad (12)$$

$$\mathbf{A}_{(k-1)} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{u}_{c,(k-1)}} \quad (13)$$

$$\mathbf{P}_{(\bullet k)} = \mathbf{A}_{(k-1)} \mathbf{P}_{(k-1)} \mathbf{A}_{(k-1)}^T + \mathbf{Q}. \quad (14)$$

This one-step ahead prediction leads to a prediction residual $\mathbf{r}_{(k)}$ with covariance matrix $\mathbf{S}_{(k)}$

$$\mathbf{r}_{(k)} = \mathbf{y}_{c,(k)} - \mathbf{g}(\hat{\mathbf{x}}_{c,(\bullet k)}, \mathbf{u}_{c,(k)}) \quad (15)$$

$$\mathbf{C}_{(k)} = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{c,(\bullet k)}, \mathbf{u}_{c,(k)}} \quad (16)$$

$$\mathbf{S}_{(k)} = \mathbf{C}_{(k)} \mathbf{P}_{(\bullet k)} \mathbf{C}_{(k)}^T + \mathbf{R}. \quad (17)$$

The second filter step calculates the Kalman filter gain $\mathbf{K}_{(k)}$, and refines the prediction as follows:

$$\mathbf{K}_{(k)} = \mathbf{P}_{(\bullet k)} \mathbf{C}_{(k)}^T \mathbf{S}_{(k)}^{-1} \quad (18)$$

$$\hat{\mathbf{x}}_{c,(k)} = \hat{\mathbf{x}}_{c,(\bullet k)} + \mathbf{K}_{(k)} \mathbf{r}_{(k)} \quad (19)$$

$$\mathbf{P}_{(k)} = \left[\mathbf{I} - \mathbf{K}_{(k)} \mathbf{C}_{(k)}\right] \mathbf{P}_{(\bullet k)}. \quad (20)$$

The output of the extended Kalman filter, as used in our hybrid estimation system, is a sequence of mean/covariance pairs $\langle \hat{\mathbf{x}}_{c,(k)}, \mathbf{P}_{(k)} \rangle$ for $\mathbf{x}_{c,(k)}$ as well as the hybrid probabilistic observation function

$$P_{\mathcal{O}}(\mathbf{y}_{(k)} | \hat{\mathbf{x}}_{(k)}, \mathbf{u}_{c,(k)}) = e^{-\mathbf{r}_{(k)}^T \mathbf{S}_{(k)}^{-1} \mathbf{r}_{(k)}/2}. \quad (21)$$

State Tracking of Uncertain Hybrid Concurrent Systems[1]

Emmanuel Benazera[2] and **Louise Travé-Massuyès**[3] and **Philippe Dague**[4]

Abstract. In this paper we propose a component-based hybrid formalism, that represents physical phenomena by combining concurrent automata with continuous uncertain dynamic models. The formalism eases the modeling of complex physical systems, and adds concurrency to the supervision of hybrid systems. Uncertainties in the model are integrated as probabilities at the discrete level and intervals at the continuous level. Our modeling framework is rather generic while focusing on the construction of intelligent autonomous supervisors by integrating a continuous/discrete interface able to reason on-line in any region of the physical system state-space, for behavior simulation, diagnosis and system tracking.

1 INTRODUCTION

In the past few years, numerous works have been presented to model embedded systems with hybrid models and reason about them for simulation, diagnosis [9] or verification [1] purposes. The modeling framework usually expresses the different operating modes of the system as a set of finite automata and associates to each mode continuous knowledge encoded through standard numeric differential equations. In this paper we propose a component-based hybrid formalism, that represents physical phenomena by combining concurrent automata with continuous uncertain dynamic models. However it is not sufficient to add continuous knowledge to automata, because moving between operating modes requires the automatic construction of the structure of the newly assembled continuous model. It means computing both the characterization of the region of the state-space of the operating mode (denoted as a *configuration*), and a proper causal ordering between the active variables in that mode. No pre-study of the behavior of the physical system is required to determine the state-space regions associated with the current system configuration(s) because the search at continuous level is casted into a boolean constraint satisfaction problem. A reasoning continuous/discrete interface (C/D I) is thus added, which provides an online generation of the characterization of the new model structure by making use of enhanced Truth Maintenance techniques [18] on the logical model. This is keypoint to achieve the diagnosis of the hybrid system for which detection is provided by the continuous layer and state identification is performed at the discrete logical level by searching for the current configuration consistent with observations. At the same time, the logical framework allows the description of purely discrete component behavior in the same manner as in [17]. Section 2 describes the discrete and the continuous layers; Section 3 presents the interface that integrates both layers together; Section 4 presents the algorithms required to reason about hybrid models and to track multiple trajectories in both simulation and diagnosis; Section 5 discusses our research, compares and references some related work.

[1] This work is supported by CNES (French Space Research Center) and ASTRIUM.
[2] Laboratory for Analysis and Architecture of Systems, Toulouse, France
[3] Laboratory for Analysis and Architecture of Systems, Toulouse, France
[4] LIPN - UMR 7030 Université Paris 13, France

2 Hybrid System Formulation

2.1 Hybrid Systems as Transition Systems

The set of all components of the physical system to be modeled is denoted by $Comps$. Every component in that set is described by a hybrid transition system. The set of all variables used to describe a component is denoted V and is partitioned in the following manner:

- $\Pi = \Pi_M \cup \Pi_C \cup \Pi_{Cond} \cup \Pi_D$ — set of discrete variables of 4 distinct types (Mode, Command, Conditional, Dependent),
- $\Xi = \Xi_I \cup \Xi_D$ — set of continuous variables of 2 distinct types (Input, Dependent).

Mode variables Π_M represent components nominal or faulty modes, such as *on* or *stuck*. Command variables Π_C are endogeneous and exogeneous commands modeled as discrete events to the system (e.g. software commands). Continuous input variables Ξ_I are exogeneous continuous signals to the system determined by its environment (e.g. known inputs or disturbances). Conditional variables Π_{Cond} are specific discrete variables that represent conditions on continuous variables. Discrete and continuous dependent variables are all other variables. Finally the set Obs contains observable variables of the physical system. Each observable signal has an explicit sampling period. Our hybrid transition system is an extension of the standard transition system [8] that adds (qualitative or quantitative) constraints to the states.

Definition 1 (Hybrid Transition System – HTS) *A Hybrid Transition System HTS is a tuple (V, Σ, T, C, Θ) with:*

- $V = \Pi \cup \Xi$ *— set of all variables.* $\forall v \in V$*, the domain of* v *is* $D[v]$*, finite for variables in* Π*, intervals or real values in* $\Re$ *otherwise.*
- Σ *— set of all interpretations over* V*.*
  Each state in Σ *assigns a value from its domain to any variable* $v \in V$*.*
- T *— finite set of transition variables.*
  Each variable τ_m *in* T *ranges over its domain* $D[\tau_m]$ *of possible transitions of the mode variable* $m \in \Pi_M$*. Each* τ_m^i *in* $D[\tau_m]$ *is a function* $\tau_m^i : \Sigma \rightarrow 2^\Sigma$*, associated to a mapping function* $l_{\tau_m^i}$ *.*
- C *— set of (qualitative or quantitative) continuous constraints over* V*.*
  Each constraint c *in* C *at least depends on one mode variable in* Π_M*.* $\forall m \in \Pi_M$*, we note* $C[m]$ *the set of constraints associated to the variable* m*.*

- Θ — *set of initial conditions.*
  Θ *is a set of assertions over* V *such that they define the set of initial possible states, i.e. the set of states* s *in* Σ *such that* $s \models \Theta$.

Note that in a HTS, due to the continuous constraints in C, some transitions can trigger according to conditions over continuous variables. At the discrete/continuous interface level, these conditions have a corresponding discrete variable in Π_{Cond}, which captures their truth value. Throughout this paper we illustrate the formalism and later on the diagnosis operation on a simple example: figure 1 shows the HTS of a room R submitted to a temperature source. It has two nominal modes: *open* (a door or a window is opened), *closed*, and a faulty *unknown* mode. The room temperature x is influenced by the temperature of the source x_e according to a first-order differential equation which accounts for the room characteristics Q_c (closed) and Q_o (open). The actions that move the room from one mode to another are modeled as observed single discrete commands $cmd = open$ and $cmd = close$. Figure 2 presents the model of a thermostat T, with faulty modes *stuck_on*, *stuck_off* and *unknown*, as well as required transitions. This thermostat switches according to the room temperature x (it should be in its *on* mode when the temperature $x \leq m$ to warm up the room, and back to its *off* mode when $x \geq M$ to cool it down). x is hence influenced by the heater setting temperature h (in mode *on*) or by the outside temperature x_{ext} (in mode *off*). The temperature variation $\dot{x}$ is observed through a sensor with additive noise $\dot{x}_{noi}$. Initially, $x = x_{ext}$, the room is *closed* and the thermostat is *on*. Variables of both HTS are:

$$
\begin{aligned}
R.mode \in \Pi_M &= (closed, open, unknown)\\
R.cmd \in \Pi_C &= (none, open, close)\\
R.c \in \Pi_{cond} &= (R.x \leq m, R.x > m \wedge R.x < M, R.x \geq M)\\
R.x \in \Xi_D &\in [-\infty, +\infty]\\
R.\dot{x} \in \Xi_D &\in [-\infty, +\infty]\\
R.\Delta x \in \Xi_D &\in [-\infty, +\infty]\\
R.\dot{x}_{noi} \in \Xi_I &\in [-1, 1]\\
R.Qc &\in [0.05, 0.15]\\
R.Qo &\in [0.02, 0.05]\\
R.a &\in [0.9, 1.1]\\
R.x_{ext} &= 4\\
T.mode \in \Pi_M &= (off, on, stuck\_on, stuck\_off, unknown)\\
T.M &= 17\\
T.m &= 10\\
T.h &= 20\\
Obs &= \{\dot{x}\}
\end{aligned}
$$

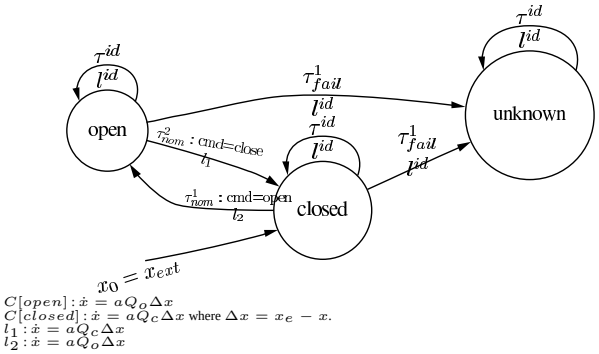


Figure 1. room HTS with unknown mode

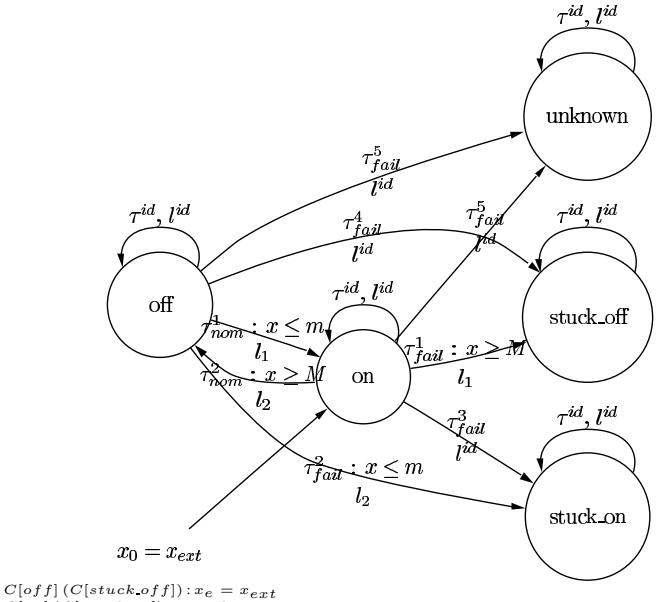


Figure 2. thermostat HTS with fault modes

2.1.1 States and Time

Considerations about time are central because both the discrete and the continuous frameworks use time representations that are different. At the continuous level, time is explicit in the equations that represent the physical system behavior, we call it *physical time* θ. Physical time is discretized according to the highest frequency sensor, providing the HTS reference sampling period T_s. $x(kT_s)$, or $x(k)$ for short, specifies the value of the continuous vector of state-variables in Ξ at physical time kT_s. We call *abstract time* the time at the discrete level. It is dated according to the occurrence of discrete events. At date t, the discrete state π_t of a HTS is the tuple (M_t, Q_t), where M_t is the vector of instances of mode variables, and Q_t the vector of instances of variables of Π in qualitative constraints. Discrete state-variables are in $\Pi \setminus \Pi_{Cond}$. Abstract time dates are indexed on physical time, which informs about how long a component has been in a given discrete state. If $t = kT_s$, then we write the indexed date t^k. When there is no ambiguity it is simply denoted by t. The *hybrid state* s_{tk} of a HTS is the tuple $(\pi_{tk}, x(k))$.

2.1.2 Transitions

Transitions describe changes between modes over time. The transition variable associated to a mode variable m is denoted τ_m such that its domain is $D[\tau_m] = \{\tau^i_m \in T_N\} \cup \{\tau^j_m \in T_F\} \cup \{\tau^{id}\}$, with:

- T_N the set of *nominal* transitions that express switches from one nominal mode to another,
- T_F the set of *faulty* transitions that move the HTS into a faulty mode,
- τ^{id} the *identity* transition that allows a HTS to stay in its current mode.

Because transitions cannot always be considered as instantaneous against the frequency of the sensors, we introduce delays on nominal transitions. Delay $d_{\tau^i_m}$ is such that once a transition τ^i_m is *enabled* it is triggered after $d_{\tau^i_m} T_s$, i.e. after $d_{\tau^i_m}$ physical time units.

While a transition is *enabled* and waiting for its delay to expire, it is said to be in *standby*. For a matter of simplification, the delay will be referred as d when there is no ambiguity. A delay on transition can also be modeled by adding modes and clocks to the hybrid transition system [4]. We do not use this representation here because we think that it does not enforce the easy representation of a component as a transition system by creating modes that are irrelevant for the diagnosis purpose. To model faults, we define fault modes of which we know the behavior, such as *stuck_on* or *stuck_off*, and a unique mode *unknown* that is rather specific because it has no constraints and covers all interpretations in Σ. Modeled faults are often abrupt faults in the sense that they do not represent tenuous parameter changes. Thus fault transitions have no delay, i.e. their duration is one physical time unit.

Definition 2 (pre and post assertions) *For a given transition τ_m^i and a given state $s_{t^k} \in \Sigma$, we note assertions $pre(\tau_m^i) = m^j \wedge \phi^i_{\Pi_{C\cup Cond}}$ and $post(\tau_m^i) = m^{j'}$ where:*

- *m^j and $m^{j'}$ are two instances of the mode variable m,*
- *$\phi^i_{\Pi_{C\cup Cond}}$ is a logical condition over instances of variables of both Π_C and Π_{Cond}.*

We refer to the *guard* of a transition as the condition statement $\phi^i_{\Pi_{C\cup Cond}}$ that triggers the transition. Only fault transitions can be spontaneous, so their guard can be always true. Traditionnally, probabilities are also attached to every nominal and faulty transitions. In our example, T is represented as follows ($\bigcirc$ is the next operator from temporal logic):

$$\begin{aligned}
R.\tau^1_{nom} &: R.mode = closed \wedge R.cmd = open \quad \bigcirc \quad R.mode = open \\
R.\tau^2_{nom} &: R.mode = open \wedge R.cmd = close \quad \bigcirc \quad R.mode = closed \\
R.\tau^1_{fail} &: R.mode = open \vee R.mode = closed \quad \bigcirc \quad R.mode = unknown
\end{aligned}$$

$$\begin{aligned}
T.\tau^1_{nom} &: T.mode = off \wedge R.x \leq m \quad \bigcirc \quad T.mode = on \\
T.\tau^2_{nom} &: T.mode = on \wedge R.x \geq M \quad \bigcirc \quad T.mode = off \\
T.\tau^1_{fail} &: T.mode = on \wedge R.x \geq M \quad \bigcirc \quad T.mode = stuck_off \\
T.\tau^2_{fail} &: T.mode = off \wedge R.x \leq m \quad \bigcirc \quad T.mode = stuck_on \\
T.\tau^3_{fail} &: T.mode = on \quad \bigcirc \quad T.mode = stuck_on \\
T.\tau^4_{fail} &: T.mode = off \quad \bigcirc \quad T.mode = stuck_off \\
T.\tau^5_{fail} &: T.mode = on \vee T.mode = off \quad \bigcirc \quad T.mode = unknown
\end{aligned}$$

There is no delay when the thermostat (room) switches between *on* (*open*) and *off* (*closed*) modes.

2.2 Moving between modes

When a transition triggers, the component switches from one mode to another, the corresponding HTS needs to transfer its continuous state vector x as well. For that reason each transition τ_m^i is associated with a *mapping function* $l_{\tau_m^i} : \Sigma \rightarrow \Sigma$ over the dependent variables in V. It initializes the value of a subset of variables in the hybrid state resulting from applying τ_m^i to $s_{t_l^k}$ where l is the abstract time index. Other variables in $s_{t_l^k}$ keep their previous value. The identity mapping function is denoted l^{id}. Triggering a transition is a two steps operation [1]. First, mode change is performed by applying the transition τ_m^i to the current hybrid state and moving to the resulting mode after its delay has expired (*transition relation* $\xrightarrow{\tau_m^i}$):

$$\frac{\tau_m^i \in T, (s_{t_l^k}, s_{t_{l+1}^{k+d}}) \in \Sigma^2, s_{t_l^k} \models pre(\tau_m^i)}{s_{t_l^k} \xrightarrow{\tau_m^i} s_{t_{l+1}^{k+d}}} \quad (1)$$

Second, initialization is performed by making use of the mapping function, and physical time goes on (*time-step relation* $\xrightarrow{\theta}$):

$$\frac{(\pi_{t_{l+1}}, x(k+d)) = l_{\tau_m^i}(s_{t_l^k})}{(\pi_{t_{l+1}}, x(k+d)) \xrightarrow{\theta} (\pi_{t_{l+1}}, x(\theta))} \quad (2)$$

where $x(\theta)$ is the continuous state associated to the discrete state $\pi_{t_{l+1}}$ over the continuous time θ. In the systems we are interested in, most of the discontinuities are driven by controller actions and preserve the state variables continuity. In our example, the temperature is obviously continuous when the thermostat switches from *on* to *off* and we use the temperature $T.M$ at this point to compute $\dot{x} = aQ_c(x_e - T.M)$. However it has been shown in [10] that in specific cases, retrieving a mapping function from the models of both considered modes is far from trivial and requires deep understanding of the physics of the phenomena abstracted in the discontinuity.

2.3 Component modes behavior

We described how transitions express component's dynamics between modes. At this point we want to represent each intra-mode behavior with two goals in mind: on the one hand the representation must encode the available qualitative or quantitative knowledge; on the other hand it must be suitable for efficient reasoning. For purely discrete components, usually software drivers as well as complex electronic devices, the behavioral model is given by a set of boolean constraints over $\Pi_C \cup \Pi_D$ that are associated to each mode variable value in the same manner as in [17]. For continuous components, the continuous behavior is expressed by discrete-time continuous constraints over Ξ. Each constraint is attached to a mode of the transition system. The discrete-time continuous constraints are of the following standard form:

$$\left\{ \begin{array}{lcl} x(k+1) & = & Ax(k) + \sum_{j=0,\ldots,r} B_j u(k-j) \\ y(k+1) & = & Cx(k+1) \end{array} \right. \quad (3)$$

where $x(k)$, $y(k)$, and $u(k)$ represent the continuous state vector of dimension n, ouput (observed) variables vector of dimension p and input (control) variables of dimension q at time kT_s, respectively; A, B_j and C are matrices of appropriate dimensions. To provide a suitable framework for reasoning, continuous constraints are encoded in a specific two levels formalism [15] which includes a causal model and an analytical constraint level. The causal model is obtained from equation (3) by expressing it as a set of *causal influences* among the (state, input or output) variables. Influences may be of different types: dynamic, integral, static and constant. The following definition expresses first and second order dynamic influences:

Definition 3 (Dynamic influence) *A dynamic influence i_{ij} is a tuple $(\xi_i, \xi_j, K, T_d, T_r, cond)$ for first order differential relations and $(\xi_i, \xi_j, K, T_d, \zeta, w, cond)$ for second order relations with :*

- *$\xi_i \in \Xi$ and $\xi_j \in \Xi$ are two continuous variables such that ξ_i influences ξ_j,*
- *K is the parameter* gain*, representing the static gain of the influence,*
- *T_d is the parameter* delay*, representing the time needed by ξ_j to react to ξ_i,*
- *T_r is the parameter* response time *representing the time needed by ξ_j to get to a new equilibrium state after having been perturbed,*
- *ζ is the* damping ratio *of the system,*
- *w is the* undamped natural frequency *of the system,*

- *cond is the parameter* condition *which specifies the logical condition under which the influence is active.* $cond$ *ranges over elements of* V.

The underlying operational model of dynamic influences is provided by the following equation:

$$\xi_j(k+1) = \sum_{p=0,\ldots,n-1} a_p \xi_j(k-p) + \sum_{q=0,\ldots,m} b_q \xi_i(k+1-q) \quad (4)$$

where ξ_i and ξ_j are continuous variables, n is the influence order and $m \leq n$ (causal link). Usually an equation is modeled by a set of influences. When necessary, uncertainties can be taken into account in the influence parameters and as additive disturbances. The first are represented by considering that parameters a_p and b_q have time independent bounded values, i.e. they are given an interval value. The latter can be introduced as a bounded value constant influence acting on ξ_j. From the superposition theorem that applies to the linear case, the computation of the updated value of variable $\xi_j \in \Xi$ in an equation eq consists in processing the sum of the activated influences from eq having exerted on ξ_j during the last time-interval. The prediction update of all the state and observed variables $x(k)$ and $y(k)$ from the knowledge of control variables $u(k)$ and influence activation conditions is performed along the causal model structure. Our representation of uncertainties leads to the prediction of continuous variable trajectories in the form of bounded envelopes. In other words, the system state $x(k)$ at every time instant $t = kT_s$ is provided in the form of a rectangle of dimension n.

Definition 4 (Causal system description – CD) *The causal system description associated to the set of continuous constraints of a* HTS *is a directed graph* $G = (\Xi, I)$ *where* I *is a set of edges supporting the influences among variables in* Ξ*, with their associated conditions and delays.*

The numerical intervals obtained from equation (4) are refined at the analytical model level with global constraints by performing a tolerance propagation algorithm [6] on the set of variables. Back to the example, the feasible continuous states of Σ are specified by the influences in each HTS:

$$R.i_1\ (static) : \text{if } (R.mode = closed) \text{ then } R.\Delta x \overset{gain=Q_c}{\longrightarrow} R.\dot{x}$$
$$R.i_2\ (static) : \text{if } (R.mode = open) \text{ then } R.\Delta x \overset{gain=Q_o}{\longrightarrow} R.\dot{x}$$
$$R.i_3\ (integral) : R.\dot{x} \overset{gain=a}{\longrightarrow} R.x$$
$$R.i_4\ (static) : R.x \overset{gain=-1,delay=1}{\longrightarrow} R.\Delta x$$
$$T.i_1\ (constant) : \text{if } (T.mode = on \vee T.mode = stuck\_on) \text{ then } T.h \longrightarrow R.\Delta x$$
$$T.i_2\ (constant) : \text{if } (T.mode = off \vee T.mode = stuck\_off) \text{ then } R.x_{ext} \longrightarrow R.\Delta x$$
$$T.i_3\ (constant) : T.\dot{x}_{noi} \longrightarrow R.\dot{x}$$

Influences without explicit conditions are valid in all modes except in the *unknown* mode. Figure 3 presents the nominal CD for the room and the thermostat.

2.4 Hybrid Component System

Once components have been modeled as HTS, constituting a generic reusable database of models, they need to be assembled in a *Hybrid Component System* to model the entire physical plant. Components are hence instantiated. Within the whole plant model, components are concurrent, i.e. able to evolve independently which allows us to reason on subparts of the model.

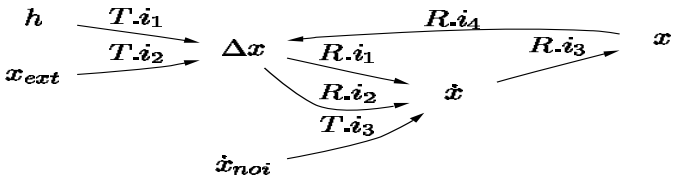


Figure 3. Causal nominal system description of the thermostat and room example

Definition 5 (Hybrid Component System – HCS) *A Hybrid Component System* HCS *is a tuple* $(Comps, V, \Sigma, T, C, \Theta)$ *with* $Comps$ *being a set of* n *components modeled as concurrent hybrid transition systems* $H_i = (V_i, \Sigma_i, T_i, C_i, \Theta_i)$, $\left(\bigcup_{i=1,\cdots,n} V_i\right) = V$, $\Sigma \subseteq \bigotimes_i \Sigma_i$, $T = \bigcup_i T_i$, $C = \bigcup_i C_i$, $\Theta = \bigcup_i \Theta_i$.

We track the evolution of a HCS over a temporal window in the form of a trajectory as a succession of states. At each time-step, constraints and commands first synchronize on shared variables in Π_D, Π_C and Ξ (the room and the thermostat share Δx). Shared variables serve as time-dated communication channels between automata. The automata must nevertheless synchronize between states. The synchronization uses transitions and is such that given components of the HCS:

- HTS that received a command synchronize on the corresponding nominal transition,
- non commanded HTS synchronize on the identity transition τ^{id}.

When synchronized, HTS instances are introduced into the trajectory whereas other HTS are not copied at each time-step. Intuitively we want to only introduce the minimal subset of the HTS necessary for tracking and diagnosis purposes. In [11] and for discrete-only models, this subset is computed using a pre-compilation of prime implicants of mode variables. In our implementation, transitions synchronize a posteriori, and only when needed by the reasoner to operate. This saves big amounts of memory as when tracking a physical system in its nominal long-term state, very few components need to be reintroduced.

The concurrency process is complexified by the introduction of delays on transitions. Figure 4 presents an example of the synchronization of four concurrent HTS, H_1 to H_4. Four transitions are enabled on shared variables at time-step t_l^k and synchronize over the three next time-steps with different delays, except for d_{τ_2} and d_{τ_4} that are equal. H_1 and H_2, as well as H_3 and H_2 have constraints that share variables. Due to different commands, the concurrence makes the four HTS change mode at time t_l^k whereas other HTS in the model stay inactive (they are not represented on the figure). Then the synchronization effort takes into account delays of triggered transitions as well as the links between HTS through shared variables:

- H_2 and H_4 have the same delay and thus participate a same hybrid state at time-step $t_{l+1}^{k+d_{\tau_2}}$,
- H_1 and H_2 synchronize at $t_{l+2}^{k+d_{\tau_1}}$. This is done with the identity transition on H_2.
- H_1 (or H_2) and H_4 don't synchronize at $t_{l+2}^{k+d_{\tau_1}}$ because they don't share any variables,
- H_1 and H_2 share variables but *don't synchronize* at $t_{l+1}^{k+d_{\tau_2}}$ because τ_1 is already in *standby*.

The last remark is of importance because it relies on the hypothesis that *we cannot track or diagnose a physical component while it is switching from one mode to another*, i.e. when one of its transitions is in *standby*, as the required transient models are often unknown or too complex. The consequence is that components only synchronize in their non-standby states.

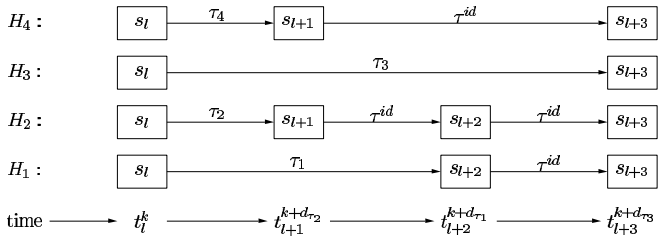


Figure 4. synchronization over 3 states of four HTS.

3 Continuous/Discrete Interface

3.1 Configurations

Depending on the mode at a given time, a HCS has its hybrid state that ranges over several continuous regions. These regions are known to be difficult to determine and compute, if not undecidable. We propose an on-line mechanism to keep track of the state-space partition by sheltering every continuous functional piece with a conjunction of logical conditions we denote as a *configuration.*

Definition 6 (HCS configuration) *A configuration for a HCS at time-step t^k is a logical conjunction $\delta_{t^k} = (\bigwedge_i m^i) \wedge (\bigwedge_j \Pi^j_{Cond})$ where the m^i are instanciations of component modes in Π_M and the Π^j_{Cond} are variables of Π_{Cond}.*

The configurations are automatically drawn from conditions on both transition guards and influences that define structural changes in the model. A configuration can be attached to one or more modes in Π_M. In our example, the continuous state is easily partitioned by the thermostat's transitions into three regions determined by the three conditions on variable x, defining 27 configurations:

$$
\begin{array}{lcl}
\mathcal{C}_1 & : & R.mode = closed \wedge T.mode = on \wedge R.x \leq m \\
\mathcal{C}_2 & : & R.mode = closed \wedge T.mode = on \wedge (R.x > m \wedge R.x < M) \\
\mathcal{C}_3 & : & R.mode = closed \wedge T.mode = off \wedge (R.x > m \wedge R.x < M) \\
\mathcal{C}_4 & : & R.mode = closed \wedge T.mode = off \wedge R.x \geq M \\
\ldots &&
\end{array}
$$

Whatever the complexity of the conditions defining the regions of the physical system, it is easy to logically express any condition as a boolean variable of Π_{Cond}, whose 1/0 corresponds to the condition and its negation. This however leads to a number of partitions that is not optimal relatively to the exact number of state-space regions in which the physical system evolves. Note that the configuration associated to the *unknown* mode encompasses the overall state-space.

3.2 Causal ordering for static equations

When switching from one mode to another, some equations and variables are added or retracted according to the new configuration. Consequently, due to the possible presence of static continuous equations in the model, a proper causal ordering of variables is to be found when entering the new mode. A brute force approach would consist in generating a new causal structure for every different mode. The problem of performing an on-line incremental generation of the causal structure has been previously addressed [16] but it is solved here in a slightly different manner. This is done by first casting the problem into a boolean constraint satisfaction problem: every continuous equation and variable in the HCS is associated to boolean variables in Π whose truth values state if the variables or equations are active or not. Rules over the boolean variables are automatically built to represent the conditions of these activations and form a logical representation of the causal-ordering problem.

3.3 Overview

The previous configuration and causal ordering problems are solved on-line by using a truth maintenance system (TMS) to reason on the corresponding boolean constraint satisfaction problems. We use the context switching algorithms of [18] because we are not interested in generating all configurations of the physical system but to switch from one to another as fast as possible. The HCS reacts to events,

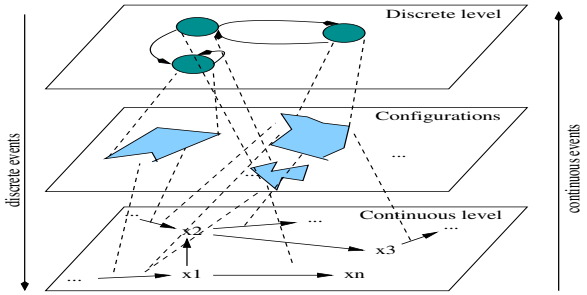


Figure 5. 3-layers interactions

i.e. observations from sensors as well as commands, and propagates them to the model's discrete and continuous levels through the logical interface and the way back. Figure 5 sums up these interactions. The C/D I, made of the variables in Π_{Cond} associated to influence conditions and transition guards, as well as the causal ordering logical model, ensures the logical consistency of the changes triggered by the flow of events.

4 Simulation and Diagnosis of a Hybrid Component System

4.1 Simulation

A HCS simulation is a run of concurrent hybrid transition systems that generates possible nominal trajectories of the HCS according to issued commands and inputs over the time. The uncertainty on the continuous constraint parameters determines the precision of the computed envelopes that enclose the observed behavior of the physical system at each time step.

Sometimes the truth value of a condition in a configuration may be undetermined when checked against a rectangular enclosing of the continuous state-variables. The problem arises from the fact that some variables over which configurations rely are not measured. When the computed bounds of such a continuous variable ξ_i span over more than one configuration region relying on that variable, we

say that the current *configuration is splitting the continuous state on variable* ξ_i. Figure 6 shows a configuration split for the thermostat

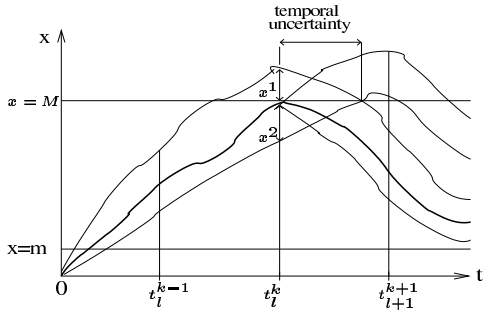


Figure 6. Transition guard split

example when crossing $x = M$. The current configuration splits on regions x^1 and x^2 and the two possible trajectories are tracked simultaneously. In applications, this situation happens rather frequently and multiple consecutive splits of a guard on the same variable can occur because sensor frequencies are usually beneath the *temporal uncertainty* induced by the envelopes. We first want to split the continuous state into logical branches then refine consequently the bounds on all continuous variables in every explored branch. For a given continuous variable ξ_i, the logical split of a configuration δ_{t^k} returns the set of possible configurations to be tracked:

$$[\delta_{t^k}](\xi_i) = \bigvee_j \left(\Pi^j_{Cond_{\xi_i}} \wedge \left(\bigwedge_n \Pi^n_{Cond} \right) \right) \tag{5}$$

where $\Pi^j_{Cond_{\xi_i}}$ are variables of Π_{Cond} relying on ξ_i and Π^n_{Cond} other conditions in δ_{t^k}. Relation (5) is used to compute the splitted areas because it is much faster than exploring the overall continuous state space. The following algorithm is applied on every tracked trajectory:

1. The configuration δ_{t^k} is checked against the rectangular region defined by variables' predicted envelopes to find a variable ξ_i over which it is splitting the state-space,
2. The state-space is logically splitted with relation (5). For each configuration $\delta^j_{t^k}$ in $[\delta_{t^k}](\xi_i)$, its corresponding continuous region is denoted $x^j_{\xi_i}(k)$ and its corresponding discrete state $\pi^j_{t^k,\xi_i}$.
3. Envelopes over variables in Ξ are refined in every region $x^j_{\xi_i}(k)$ by filtering them on the constraints defined by the conditions in the configuration [6].
4. $(\pi^j_{t^k,\xi_i}, x^j_{\xi_i}(k))$ constitute new hybrid states enclosed in new trajectories to be tracked.

The three preceding steps are applied for remaining variables on the growing set of generated trajectories. Finally the resulting set of computed hybrid states is:

$$[s_{t^k}] = \bigotimes_{i,j} (\pi^j_{t^k,\xi_i}, x^j_{\xi_i}(k)) \tag{6}$$

In our example, the thermostat's configurations only split on the temperature x. On figure 6, until time-step t^k_l, the configuration of the HCS is

$$\mathcal{C}_2 : R.mode = closed \wedge T.mode = on \wedge R.x > m \wedge R.x < M$$

At time-step t^k_l, due to the crossing of $x = M$, the current configuration is splitted on x. A new partial hybrid state comes from equation (5):

$$R.mode = closed \wedge T.mode = on \wedge R.x \geq M$$

Then bounds of variable x are refined in each configuration by filtering the values with respective constraints $R.x > m \wedge R.x < M$ and $R.x \geq M$. As transition $T.\tau^2_{nom}$ turns *enabled* with the second configuration, the configuration is instantaneously ($T.\tau^2_{nom}$ has no delay) updated to:

$$\mathcal{C}_4 : R.mode = closed \wedge T.mode = off \wedge R.x \geq M \tag{7}$$

From that point the system tracks two distinct trajectories.

4.2 Fault Detection

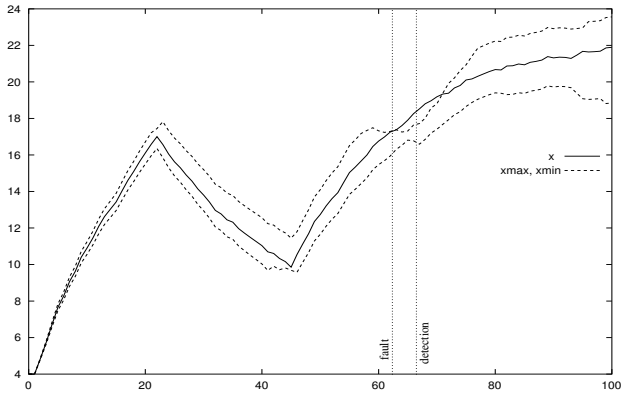
The detection algorithm then uses the above prediction of the endogeneous continuous variable values to obtain robust decisions about the existence of faults, based on *adaptive thresholds* provided by the envelopes' upper and lower bounds. This is performed by comparing the predicted and observed values of variables across time. The adaptive thresholds principle fairly reduces the possibility of false alarms when tracking the system. However, to achieve better robustness, we usually mark a variable as mibehaving after it has been outside of its bounds for at least n_{misb} physical time-steps. After that delay, the diagnosis operation is triggered.

For dynamic influences, the algorithm sensitivity relies on a mixed strategy which combines an observer type strategy (*closed-loop* mode, i.e. the measure of a variable y at time t is used to elaborate the prediction of y at time $t+1$) with a pure simulation strategy (*open-loop* mode, i.e. the prediction of y at time $t+1$ is obtained from the prediction of y at time t) to determine the thresholds and further assess variable states. We call this strategy a *semi-closed loop* (SCL) strategy [13]. The mode control (open-loop or closed-loop) depends on whether the observed value of a variable y is in the predicted envelope (normal situation) or out of it (alarming situation). As soon as the variable becomes alarming, running on a closed-loop mode might drive the prediction to follow the fault, turning the detection procedure insensitive to the fault. The prediction temporal window is hence scaled up by switching to the open-loop mode. Note that the fault detection mechanism is very efficient at ruling out wrong trajectories issued from multiple successive splits on the same boundary constraint.

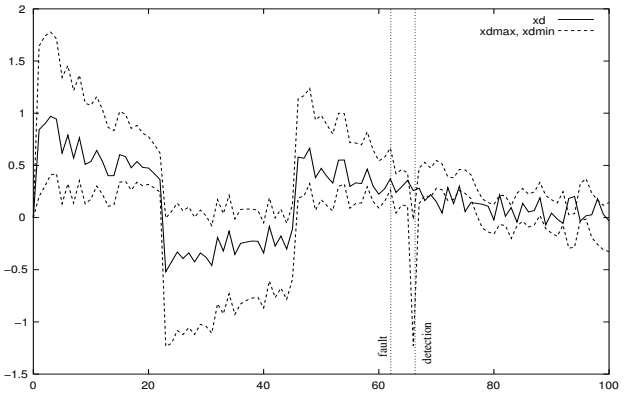
Figure 7 shows three scenarios with faults where detection is applied. On the first scenario the thermostat fails to switch at time-step 63 and sticks to its *on* mode. In the second scenario the constant $T.h$ is degraded from time-step 46 to a lower value, so the heater is slower to warm the room. Scenario three presents a fault characterized by an abrupt structural change in the thermostat model. For all scenarios, $n_{misb} = 1$.

4.3 Diagnosis

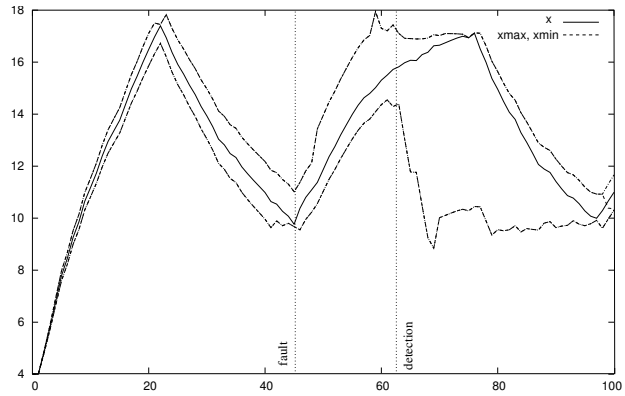
When a fault is detected, a diagnosis comes back to find the current configuration of the HCS according to observations, inputs and commands. This must be performed over a finite temporal window [11], but because of the fault detection at a continuous level the problem of losing solutions is strongly reduced. The temporal window is usually set up to the physical time that corresponds to the longest chain of non-repeated transitions. In our example 20 physical time-steps cover an *on-off* complete sequence.



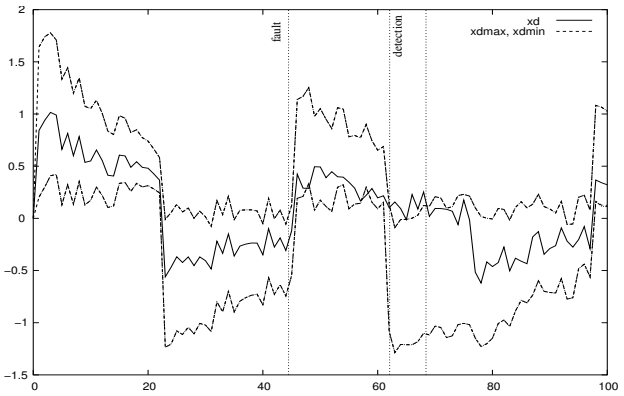
(a) Scenario 1, x: After detection and diagnosis, a few more time-steps are necessary for the prediction to catch up with the physical system. This comes from the fact that the estimation of the time of the fault is not accurate enough: because of the time uncertainty due to the envelopes, the estimation is a few time-steps late.



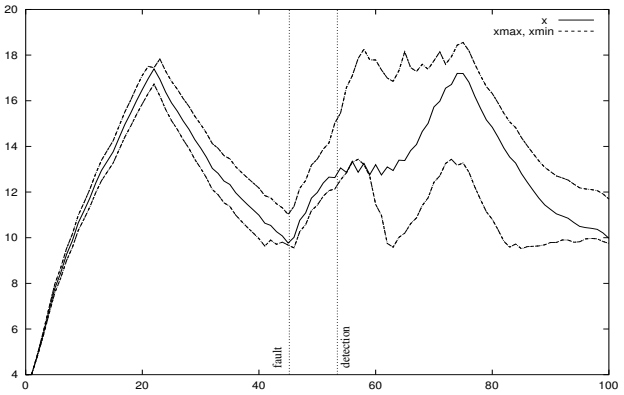
(b) Scenario 1, $\dot{x}$: the fault is detected at time-step 68.



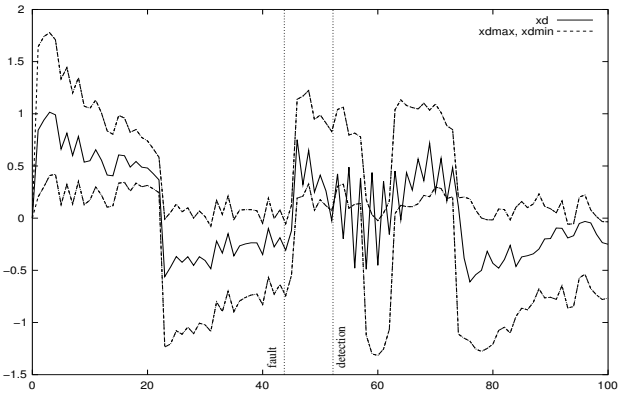
(c) Scenario 2, x: After the fault is diagnosed, the *blind state-tracking method* uses the nominal behavior of the thermostat and predicts all possible switches at each time-step: the very wide envelope shows that it is not sure if the thermostat is *on* or *off*.



(d) Scenario 2, $\dot{x}$: The fault is not so abrupt as to be detected instantaneously. Measures go *in* the predicted bounds again at time-step 69. This is due to the fact that when using the *blind state-tracking method*, the thermostat's controller model is still switching on valid thresholds.



(e) Scenario 3, x: The thermostat switches on valid thresholds and the *blind state-tracking method* keeps a relatively good tracking of the temperature after the fault occured. This is due to the fact that the physical model of the room is still valid.



(f) Scenario 3, $\dot{x}$: After a thermostat's structure change, the heater setting temperature $T.h$ is oscillating. When turned *off*, T keeps its nominal behavior.

Figure 7. Three fault scenarios

Definition 7 (HCS **Diagnosis)** *A diagnosis* $diag(t)$ *over* m *time-steps for a* HCS *is such that* $diag(t) = \{\delta_t\}_{t=1,\cdots,m}$ *with the consitency of:*

$$HCS \cup Obs_{t=1,\ldots,m} \cup \left(\bigcup_{t=1,\cdots,m} \delta_t \right) \quad (8)$$

Solving relation (8) is a three steps operation. First, existing conflicts (a set of influences which cannot be unfaulty altogether) are exhibited from the causal system description (CD) of the HCS, each influence stamped with a temporal label and activation condition. They are then turned into diagnosis candidates by a failure-time oriented enhanced version of the hitting set algorithm [14]. Temporal information is drawn from maximizing on each components the delays of the influences downstream the faulty variables in CD.

Second, at the configurations level, the TMS negates the activation conditions of the conflicting influences and fastly iterates through the logical remaining configurations to reinsure the consistency. Finally, every found configuration is checked against the past observations over the temporal window before being approved as in [11] except that candidate generation and consistency checks are interleaved and run from present time back to the beginning of the temporal window. Configuration solutions to the diagnosis problem contain a mode instanciation of every necessary component in the HTS explaining the observations. Note that on figure 7, for all three scenarios, the diagnosis operation is performed in less than 0.1 seconds on a Pentium II 300 Mhz, which is beneath the measures' frequency, so the detection time-step is equal to the diagnosis time-step.

4.3.1 Diagnosis example with a fault mode

When applied to the first scenario, the diagnosis starts as soon as $\dot{x}$ goes out of its bounds for all currently tracked trajectories: iterating through the system nominal CD from figure 3, at timestep 68 the influences in conflict are $\Gamma = \{T.i_3,\ T.i_2,\ R.i_1,\ R.i_3,\ R.i_4\}$. Relatively to the current configuration (7) it is equivalent to add the constraints $\Gamma_C = \{\bigvee_{m^i=D[T.mode]} T.mode = m^i, R.mode = closed, T.mode = off \vee T.mode = stuck\_off, \bigvee_{m^j \in D[R.mode]} R.mode = m^j\}$ which are activation conditions on the influences in conflict. As $R.i_4$ has a delay of 1, the elements of the last conflict are stamped with the current physical time minus 1. Other conflicts elements are stamped with the current physical time.

The TMS then seeks for consistency on both the configurations and the transition model starting from the current configuration by inserting the negation of the elements in Γ_C: $\Gamma_{\neg C} = \{T.mode = unknown, R.mode = open \vee R.mode = unknown, T.mode = on \vee T.mode = stuck\_on \vee T.mode = unknown, R.mode = unknown\}$ and returns the following possible configurations ranked according to the probabilities attached to transitions and to the number of faults leading to them:

1 : $(R.mode = closed) \wedge (T.mode = stuck\_on) \wedge (R.x \geq M)$
$2a$: $(R.mode = closed) \wedge (T.mode = unknown) \wedge (R.x \geq M)$
$2b$: $(R.mode = unknown) \wedge (T.mode = stuck\_on) \wedge (R.x \geq M)$
3 : $(R.mode = unknown) \wedge (T.mode = unknown) \wedge (R.x \geq M)$

Other configurations with the thermostat in modes *on, stuck_off*, or the room in mode *open* are ruled out during the search process because there are no transitions or past observations and commands consistent with these configurations. Diagnosis 1 fits with the fault in the first scenario (thermostat took transition τ^3_{fail}). The state vector is reinitialized according to the mapping function of τ^3_{fail} (l^{id}) before the tracking continues.

4.3.2 Diagnosis example with the unknown mode

Scenarios 2 and 3 primarily lead to diagnosis $2a$ where the thermostat is in the *unknown* mode. This mode is useful at the discrete level because it assures that there is always a solution to the diagnosis problem[5]. At the continuous level however, it has no model, so it is not possible to track a HTS in that mode. Isolating the *unknown* automata so as to continue the prediction of the behavior of others HTS in the model often leads to tracking based on a wrong model: in scenario 2, once the mode of T has been diagnosed to be *unknown*, influences referring to T are inactive which is equivalent to predict R's behavior with $T.h = 0$. Our current solution to that problem is to use a dedicated *blind state-tracking method* that is applicable thanks to the semi-closed loop fault detection strategy described in subsection 4.2. When a component is found to be in its *unknown* mode, the nominal model of the component is used instead. The detection module runs on open-loop prediction mode until the measures fall into the envelopes again. This is guaranteed to occur because the open-loop predicted envelopes widen with time (uncertainty propagation of interval models). Triggered by this event, the detection module then switches to closed-loop prediction mode and is able to track the system until the measures get out of their bounds again, and so on. This is the method applied on scenarios 2 and 3 on figure 7. However in scenario 2, an improved solution could be to use parameter estimation techniques as proposed in [9] because the structure of the model is still valid. But drawbacks are the additional computational cost and the fact this would leave the system untracked for a period of time (proper parameter estimation requires to wait for properly excited data). More research is needed to integrate existing parameter estimation and model fitting techniques into our framework. Also note that such faults generally result from the natural degradation of the monitored physical system and could be taken into account in causal models [12].

5 Summary, Discussion and Related Work

In this paper we extend previous work on diagnosis in the AI community by presenting a formalism that merges concurrent automata with continuous dynamic system models and reasons about its configurations using logical tools. The problem of reasoning about and diagnosing complex physical plants without computing their continuous reachable state-space is addressed. The approach integrates numerous techniques from different fields into a runnable standalone application, which is able to deal with real-world problems such as satellite state-tracking [3]. The modeling, simulation and diagnosis tools are implemented, including the engine that splits the configurations. The program generates a C++ runtime that is intended to be demonstrated on an autonomous spacecraft test bench at CNES.

Other formalisms for building comprehensive and tracktable hybrid systems include [10] and [4]. But none of these approaches provide an intuitive component-based framework allowing engineers to build reusable models of equipments. Moreover the models often include numerous functional modes that are irrelevant to the diagnosis task. For instance [4] introduces additional modes to deal with delayed transitions, and [10] rather focuses on the expression of the approximations able to produce sound hybrid models of complex physical systems. Besides, it examines types of discontinuities that are rarely encountered in controlled systems. In such systems, most

[5] Note that the *unknown* mode is also a dead-end since no nominal transition can lead out of this mode.

of the discontinuities are driven by controller actions and preserve state variables continuity.

Our work takes numerous ideas from the discrete-only work at the basis of Livingstone [17, 11] and adds and links continuous knowledge to it. The difficult problem of the temporal window that required aggregating in a history all past states in every tracked trajectory is now strongly reduced as it is less likely that a wrong trajectory is tracked without detecting anomalies at the continuous level. [9] introduced a diagnosis-dedicated hybrid formalism relying on error bounds for the detection parts, but without concurrence nor transitions triggered autonomously from the continuous level; it uses probabilities, parameter estimation as well as data fitting to refine the diagnosis. [20] unifies traditional continuous state observers with hidden Markov models belief update in order to track hybrid systems with noise but do not include concurrent models nor any mapping function discussion. The approach is interesting because it makes extensive use of probabilities where we chose to rely on bounded uncertainties (intervals) at the continuous level and on probabilities at the discrete level. In fact these are different uncertainties as the uncertainty is uniformly distributed in the case of intervals whereas [20] relies on normal laws. In our point of view using probabilities at the discrete levels allows to prune an otherwise prohibitive search, but intervals offer a more compact representation of uncertainties on continuous variables. However, the point would need more discussion and research. Similar approaches also include [21] that combines a Petri net and signal analysis to estimate the discrete modes and overcome an exponential cost in the number of sensors, but lacks an efficient diagnosis engine; and [7] that uses a dedicated bayesian network as well as a method of smoothing that helps successfully diagnose faults with a very low belief state. Note that the model checking community has recently investigated the use of interval-based numerical models [5].

An advantage of our approach is that any type conditions associated to transitions and influences (e.g. continuous functions as guards) can be modeled and tracked without being directly observed. Finally on-line performances can be enhanced as the formalism allows the logical model to be pre-compiled before use by generating prime-implicants on transition guards [19] and influence conditions. However it still happens that trajectories cannot be discriminated due to too much imprecision on parameters that leads to overlapping envelopes. A solution to this problem has been to merge such envelopes and corresponding trajectories. Another remark concerns the splits that occur and are not linked to any real mode or structure changes in the model: when starting the thermostat and room models with external temperature $x_{ext} < m$, a split occurs when first crossing at $x = m$. These splits however are sound and refine the bounds on continuous variables as they allow the system to reduce temporal uncertainty at the crossing point.

Further work will focus on reconfiguration by reasoning on configurations with the same core algorithms as for diagnosis. This will be done by identifying a set of goal configurations and find under uncertainty a valid plan made of least costly endogeneous commands to reach each goal. We think that additive improvements could also include automatic controller synthesis as in [2] as well as parameter estimation based on the causal structure of the continuous level in order to refine the tracking of the system when in its *unknown* mode. In a near future more results are to come out as our implementation is intended to be tested on spacecraft models and run on-board ground based satellite hardware.

6 Acknowledgements

We are very thankful to Marie-Claire Charmeau and Bernard Polle for providing information about application and valuable comments on this work.

REFERENCES

[1] R. Alur, C. Courcoubetis, N.Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A.Olivero, J.Sifakis, and S.Yovine, 'The algorithmic analysis of hybrid systems', in *Proceedings of the 11th International Conference on Analysis and Optimization of Discrete Event Systems*, pp. 331–351, (1995).

[2] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, 'Effective controller synthesis of switching controllers for linear systems', *Proceedings of the IEEE, Special Issue on Hybrid Systems*, **88**, 1011–1025, (July 2001).

[3] E. Benazera, L. Travé-Massuyès, and P. Dague, 'Hybrid model-based diagnosis for autonomous spacecrafts', in *Proceedings of the first ESA Workshop on On-Board Autonomy, October 2001, Nordwijk, Netherlands*, pp. 279 – 286, (2001).

[4] T. Henzinger, 'The theory of hybrid automata', in *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pp. 278–292, New Brunswick, New Jersey, (1996).

[5] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi, 'Beyond HYTECH: Hybrid systems analysis using interval numerical methods', in *HSCC*, pp. 130–144, (2000).

[6] E. Hyvonen, 'Constraint reasoning based on interval arithmetic: The tolerance propagation approach', *Artificial Intelligence*, **58**(1-3), 71–112, (1992).

[7] Uri Lerner, Ronald Parr, Daphne Koller, and Gautam Biswas, 'Bayesian fault detection and diagnosis in dynamic systems', in *AAAI/IAAI*, pp. 531–537, (2000).

[8] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems - Specification*, Springer-Verlag, 1992.

[9] S. McIlraith, G. Biswas, D. Clancy, and V. Gupta, 'Towards diagnosing hybrid systems', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis DX-99*, (1999).

[10] P. J. Mosterman and G. Biswas, 'A comprehensive methodology for building hybrid models of physical systems', *Artificial Intelligence*, **121**, 171–209, (2000).

[11] P. Nayak and J. Kurien, 'Back to the future for consistency-based trajectory tracking', in *Proceedings of AAAI-2000, Austin, Texas*, (2000).

[12] R. Pons, L. Travé-Massuyès, and M. Porcheron, 'Model-based diagnosis and maintenance of time-varying dynamic systems', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis DX-99*, pp. 211–219, (1999).

[13] L. Travé-Massuyès, T. Escobet, R. Pons, and S. Tornil, 'The ca-en diagnosis system and its automatic modeling method', *Computación i Sistemas Journal*, **5**(2), 128–143, (2001).

[14] L. Travé-Massuyès and J.A. Jimenez, 'Fault detection and isolation in the ca-en system', Technical report, LAAS-CNRS, Toulouse, France, (2001).

[15] L. Travé-Massuyès and R. Milne, 'Tigertm: Gas turbine condition monitoring using qualitative model based diagnosis', *IEEE Expert Intelligent Systems & Applications*, (1997).

[16] L. Travé-Massuyès and R. Pons, 'Causal ordering for multiple modes systems', in *Proceedings of the Eleventh International Workshop on Qualitative Reasoning*, pp. 203 – 214, (1997).

[17] B. C. Williams and P. Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of AAAI-96, Portland, Oregon*, pp. 971–978, (1996).

[18] B. C. Williams and P. Nayak, 'Fast context switching in real-time reasoning', in *Proceedings of AAAI-97, Providence, Rhode Island*, (1997).

[19] B. C. Williams and P. Nayak, 'A reactive planner for a model-based executive', in *Proceedings of IFCAI-97*, (1997).

[20] B.C. Williams, M. Hofbaur, and T. Jones, 'Mode estimation of probabilistic hybrid systems', Technical report, Massachusset Institute of Technology, (2002).

[21] Feng Zhao, Xenofon D. Koutsoukos, Horst W. Haussecker, James Reich, Patrick Cheung, and Claudia Picardi, 'Distributed monitoring of hybrid systems: A model-directed approach', in *IJCAI*, pp. 557–564, (2001).

HCBFS: Combining Structure-Based and TMS-Based Approaches in Model-Based Diagnosis

T. K. Satish Kumar
Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

Abstract

Model-based diagnosis can be formulated as the combinatorial optimization problem of finding an assignment of behavior modes to all the components in a system such that it is not only consistent with the system description and observations, but also maximizes the prior probability associated with it. Because the general case of this problem is exponential in the number of components, we try to leverage the structure of the physical system under consideration. Traditional dynamic programming techniques based on the underlying constraint network (like heuristics derived from maximum cardinality ordering) do not necessarily supplement or do better than algorithms based on using truth maintenance systems (like conflict-directed best first search).

In this paper, we compare the two approaches and examine how we can incorporate the dynamic programming paradigm into TMS-based algorithms to achieve the best of both the worlds. We describe an algorithm called hierarchical conflict-directed best first search (HCBFS) to solve a large diagnosis problem by heuristically decomposing it into smaller sub-problems. We also delve into some of the implications of HCBFS with respect to (1) precompiling the system description to a form that can amortize the cost of a diagnosis call and (2) facilitating other hybrid techniques for diagnosis.

1 Introduction

Diagnosis is an important component of autonomy for any intelligent agent. Often, an intelligent agent plans a set of actions to achieve certain goals. Because some conditions may be unforeseen, it is important for it to be able to reconfigure its plan depending upon the state in which it is. This mode identification problem is essentially a problem of diagnosis. In its simplest form, the problem of diagnosis is to find a suitable assignment of modes in which each component of a system is behaving in, given some observations made on it. It is possible to handle the case of a dynamic system by treating the transition variables as components (in one sense) [Kurien and Nayak, 2000].

Definition *(Diagnosis System)***:** A *diagnosis system* is a triple $(SD, COMPS, OBS)$ such that:
1. SD is a system description expressed in one of several forms — constraint languages like propositional logic, probabilistic models like Bayesian network etc. SD specifies both component behavior information (SD_B) and component structure information (i.e. the topology of the system) (SD_T).
2. $COMPS$ is a finite set of components of the system. A component $comp_i$ ($1 \leq i \leq |COMPS|$) can behave in one of several, but finite set of modes (M_i). If these modes are not specified explicitly, then we assume two modes — failed ($AB(comp_i)$) and normal ($\neg AB(comp_i)$).
3. OBS is a set of observations expressed as variable values.
The task of diagnosis is to "identify" the modes in which individual components are behaving given the system description (SD) and the observations (OBS).

Definition *(Candidate)***:** Given a set of integers $i_1 \cdots i_{|COMPS|}$ (such that for $1 \leq j \leq |COMPS|$, $1 \leq i_j \leq |M_j|$), a *candidate* $Cand(i_1 \cdots i_{|COMPS|})$ is defined as $Cand(i_1 \cdots i_{|COMPS|}) = (\bigcup_{k=1}^{|COMPS|}(comp_k = M_k(i_k)))$.
Here, $M_u(v)$ denotes the v^{th} element in the set M_u (assumed to be indexed in some way).

2 Diagnosis as Combinatorial Optimization

Consider diagnosing a system consisting of three bulbs B_1, B_2 and B_3 connected in parallel to the same voltage source V under the observations $off(B_1)$, $off(B_2)$ and $on(B_3)$. $AB(V) \wedge AB(B_3)$ is a diagnosis under the consistency-based formalization of diagnosis [de Kleer *et al.*, 1992] if we had constraints only of the form $\neg AB(B_3) \wedge \neg AB(V) \rightarrow B_3 = on$. Intuitively however, it does not seem reasonable because B_3 cannot be *on* without V working properly. One way to get around this is to include fault models in the system [Struss and Dressler, 1989]. These are constraints that explicitly describe the behavior of a component when it is not in its nominal mode (most expected mode of behavior of a component). Such a constraint in this example would be $AB(B_3) \rightarrow off(B_3)$. Diagnosis can become indiscriminate without fault models. It is also easy to see that the consistency-based approach can exploit fault models (when they are specified) to produce more intuitive diagnoses

(like only B_1 and B_2 being abnormal).

The technique of using fault models however is associated with the problem of being too restrictive. It may not model the case of some strange source of power making B_3 *on* etc. The way out of this is to allow for many modes of behavior for the components of the system. Each component has a set of modes with associated models — normal modes and fault modes. Each component has the *unknown* fault mode with the empty model. The *unknown* mode tries to capture the *modeling incompleteness* assumption (obscure modes that we cannot model in the system). Also, each mode has an associated probability that is the prior probability of the component behaving in that mode. Diagnosis can now be cast as a combinatorial optimization problem of assigning modes of behavior to each component such that it is not only consistent with $SD \cup OBS$, but also maximizes the product of the prior probabilities associated with those modes [de Kleer and Williams, 1989]. Note that the combinatorial optimization formulation of diagnosis assumes independence of the behavior modes of components.

Definition *(Combinatorial Optimization Characterization)* A candidate $H = Cand(i_1 \cdots i_{|COMPS|})$ is a diagnosis if and only if $SD \cup H \cup OBS$ is satisfiable and $P(H) = (\Pi_{k=1}^{|COMPS|} P(comp_k = M_k(i_k)))$ is maximized.

There are many other characterizations of diagnoses based on the notions of abduction, Bayesian model selection, model counting [Kumar, 2002] etc. These characterizations (including combinatorial optimization) are mostly for choosing the most likely diagnosis and do not incorporate any notion of refinement [Lucas, 1997]. The combinatorial optimization formulation to return the most likely diagnosis is however justified, practical and suited for a variety of real-life applications [Kurien and Nayak, 2000]. It also benefits from the availability of computationally efficient algorithms to solve combinatorial optimization problems [Williams and Nayak, 1996].

3 Computational Methods

Definition *(Combinatorial Optimization Problem)***:** A *combinatorial optimization problem* is a tuple (V, f, c) where (1) V is a set of discrete variables with finite domains. (2) An *assignment* maps each v in V to a value in v's domain. (3) f is a function that decides *feasibility* of assignments. (4) c is a function that returns the *cost* of an assignment. (5) We want to minimize $c(V)$ such that $f(V)$ holds.

In the context of diagnosis, the following correspondences hold: (1) $V = COMPS$. (2) Domains correspond to modes of behavior of components (3) An *assignment* is a candidate. (4) c is a simple cost model assuming independence in behavior modes of components $c(comp_i = M_i(v)) = log\frac{P(comp_i=M_i(v^*))}{P(comp_i=M_i(v))}$. Here, $M_i(v^*)$ is the *nominal* mode of behavior of $comp_i$; $P(comp_i = M_i(v^*)) \geq P(comp_i = M_i(v))$ for any $v \neq v^*$. $c(Cand(i_1 \cdots i_{|COMPS|})) = \sum_{k=1}^{|COMPS|} c(comp_k = M_k(i_k))$. (5) f is the *satisfiability* of $SD \cup Cand(i_1 \cdots i_{|COMPS|}) \cup OBS$.

A brute-force method of solving such a problem is to use a simple best first search (BFS) which is clearly exponential in the number of components. It can however, be potentially improved by leveraging the structure of the system. One popular method of leveraging structure using the paradigm of dynamic programming is to use heuristics derived from a maximum cardinality ordering (m-ordering) [Tarjan and Yannakakis, 1984] over the constraint network relating the variables of the system. Such techniques have been used in a variety of domains — Bayesian network reasoning, constraint satisfaction problems [Dechter, 1992] etc. A constraint network on the variables of the system is defined by having the variables represent nodes and constraints in SD represent hyper-edges. Any kind of optimization or satisfaction defined over the variables can be done in time exponential in the induced width of the graph [Dechter, 1992]. Although the induced width itself cannot be found constructively in polynomial time, heuristics derived from m-ordering perform reasonably well in practice. Throughout the rest of this paper, we will refer to all such heuristics as naive m-ordering (naive because they do not supplement the power of TMS-based algorithms).

These heuristics however, may not be directly beneficial or applicable when the number of components is somewhat lesser than the total number of variables in the system (which is usually the case). The induced width of the constraint network relating all the variables in a physical system can easily be much more than the number of components. A further disadvantage of such approaches is that often the relationships between variables are too complex and consistency checks may involve some kind of a "simulation". Since dynamic programming techniques based on these heuristics maintain and build partial assignments, they are very likely to be costly processes. Furthermore, in many cases, the number of faulty components is usually far lesser than the total number of components and these techniques do not exploit this significantly towards computational gains.

One approach that addresses these problems somewhat indirectly is conflict-directed best first search (CBFS) [Williams and Nayak, 1996]. It is based on the idea of examining hypotheses in decreasing order of their prior probabilities and using a truth maintenance system (TMS) to catch minimal conflicts and focus the search. QCBFS [Kumar, 2001] is an extension of CBFS that leverages qualitative knowledge present in the system. Because hypotheses are examined in order of their probabilities, diagnoses that entail a nominal behavior for all but a few components are caught as soon as possible (unlike in the naive m-ordering case).

A TMS incorporates and uses the following properties: (1) If a partial assignment to the mode behaviors of a subset of the components is inconsistent, then any other assignment that contains this subset unchanged is also inconsistent. (2) Smaller conflicts result in more pruning of the search space and therefore, whenever an assignment A is infeasible, a minimal infeasible subset of A is returned (using dependency tracking). (3) Since the hypotheses that we examine differ only incrementally from one another in the assignments for behavior modes of components, feasibility checks are made more efficient (like in ITMS [Nayak and Williams, 1997]).

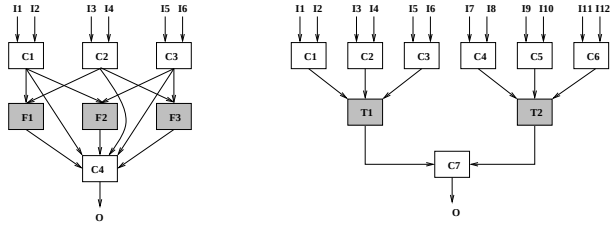


Figure 1: (a) Shows the worst-case scenario for m-ordering. (b) Shows the worst-case scenario for CBFS.

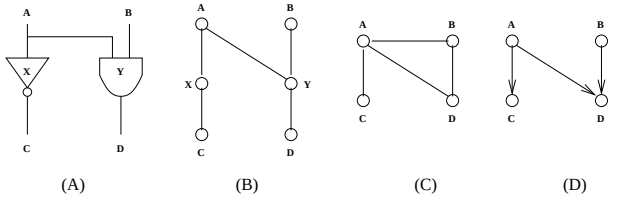


Figure 2: (A) The physical setting. (B) The graph representation. (C) The constraint network. (D) The *T-Graph.*

3.1 Comparison of naive m-ordering and CBFS

While naive m-ordering exploits the structure of the underlying constraint network, it does not exploit the fact that we are interested in an assignment only to the components of the system (and not the intermediate variables). This becomes a liability especially when consistency checks involve "simulation" and are therefore costly. It performs badly when a "small" number of components are "tightly" connected. Figure 1(a) illustrates the bad behavior of m-ordering. There are 4 components that can possibly behave in different modes (C1, C2, C3 and C4). F1, F2 and F3 are not modeled as components but are some complex mappings (involving simulation) from their inputs to outputs. The number of parents of C4 is equal to 6 and the combinatorial optimization problem is exponential in this quantity [Darwiche, 1998]. A TMS-based algorithm however, would require only a search space exponential in the number of components (=4). This can be verified by noting that once a set of modes is assumed for each component (as in a TMS-based algorithm), verifying that the current set of inputs lead to the observations is not exponential but only polynomial in the size of the graph. This is because any component maps its inputs to a unique output and we just need to follow the inputs through all the transformations defined by the components to eventually verify whether there is a match with the observations. In the case of naive m-ordering however, combinatorial optimization requires us to compute and store against all values of communication variables around a family (also called partition), the most likely modes of behavior of the components in it. This makes it exponential in the induced width of the graph. It is also easy to see (as claimed earlier) that when the diagnosis is quite close to the nominal behaviors of components, there is no obvious way of exploiting it with m-ordering.

CBFS on the other hand, exploits the fact that we are interested in an assignment only for the components of the system, but does not exploit the structure of the physical setting efficiently. The only indirect way in which the structure comes into play is in the TMS implementation of *f* to catch minimal conflicts. The problem with CBFS is in large due to the fact that all inconsistencies are traced back to the components. This makes CBFS perform sub-optimally when components are "loosely" connected. Figure 1(b) illustrates the bad behavior of CBFS. An observation of O = 1 when C7 is an XOR gate entails the conflicts $\{T1 = 1, T2 = 1\}$ and $\{T1 = 0, T2 = 0\}$. Note that T1 = 0, T2 =0, T1 = 1 or T2 = 1 are not conflicts by themselves. If all inconsistencies are traced back to the components C1 - C6 however, the search space over component behavior modes is never pruned by a minimal conflict of size lesser than 6. If on the other hand, we split the problem into two (by treating the cases $\{T1 = 1, T2 = 0\}$ and $\{T1 = 0, T2 = 1\}$ separately) the search space can be reduced to being exponential in 4 variables (rather than 6).

4 Hierarchical Conflict-Directed Best First Search (HCBFS)

Before we describe HCBFS as an algorithm that can combine the best of both the above approaches, we define the following notions related to the structure of a physical setting.

Definition *(Structural Parameter Set)***:** The *structural parameter set* S of a physical system is the 4-tuple $S = (COMPS, I, O, T)$. Here, I is the set of external inputs, O is the set of output variables under observation, and T is the set of intermediate variables in the system which are not under observation.

Definition *(Graph Representation)***:** The *graph representation* of a physical system with structural parameter set S and a topology characterized by SD_T is a graph with nodes corresponding to elements in S and undirected edges corresponding to physical connections inferred from SD_T.

Definition *(*-node)***:** A node in the graph representation of a physical system is a *c-node*, *i-node*, *o-node* or a *t-node* when it corresponds respectively to a component, input variable, output variable or an intermediate variable.

Definition *(T-Graph)***:** The *T-Graph* of a physical system with structural parameter set S and topology SD_T is a graph built out of removing the *c-nodes* from its graph representation and directly connecting the inputs to their outputs (in that direction).

Figure 2 illustrates the above definitions for a simple physical setting. Note that the *graph representation* is not the same as the constraint network specified by SD. While the constraint network is built on the variables of the system (excluding components) using SD, the graph representation is built only out of SD_T (and includes the components). The *T-Graph* represents the *causal* relationships among the variables (excluding the components) and it can be observed that the constraint network is equivalent to the *T-Graph* moralized by making a clique out of all the parents of any node [Dechter, 1992].

Notation: Let $M(i)$ be the set of modes in which component $comp_i$ can behave. Let c_i be the cardinality of this set. Let $T(i)$ be the set of values an intermediate variable $t\text{-}node_i$ can

take. Let t_i be the cardinality of this set.

Definition *(c-size)***:** The *c-size* of a sub-graph G is the product of the number of modes in which each component it contains can behave, = $\Pi_{i \in COMPS(G)} c_i$.

Definition *(t-partition)***:** A *t-partition* of a graph representation is any collection of vertex induced sub-graphs $S_1 \cdots S_k$ such that for all i, j with $1 \leq i, j \leq k$, $S_i \cap S_k \subseteq T$.

Definition *(t-size)***:** The *t-size* of a sub-graph in a *t-partition* of the original graph is the product of the number of different values each of the *t-nodes* it shares with other sub-graphs, can take. In other words, suppose $S_1 \cdots S_k$ form a *t-partition* of the original graph. Denote the *t-nodes* in each of these sub-graphs by $ST_1 \cdots ST_k$. The *t-size* of S_i is given by $\Pi_{j \in ST_i}(t_j | \exists h, 1 \leq h \leq k, h \neq i, j \in ST_h)$.

Definition *(ct-size)***:** The *ct-size* of a graph is the product of its *c-size* and *t-size*.

Given the *graph representation* of a physical system, its *c-size* characterizes the size of the search space for CBFS. The general idea behind HCBFS is to reduce the effective search space of CBFS using dynamic programming. Suppose we were able to divide the system into two subsystems that had components $comp_{i_1} \cdots comp_{i_{n_1}}$ and $comp_{j_1} \cdots comp_{j_{n_2}}$ such that $n_1 + n_2 = |COMPS|$. Now, the search space for each of these two individual partitions (for CBFS) becomes their respective *c-sizes*. Calling them C_1 and C_2 respectively, we have $C_1.C_2 = C$ (C is the *c-size* of the original graph). Of course, the search cannot simply be done in each of them independently because of the common variables they share. However, we can apply the idea of dynamic programming to solve each of these partitions for *all* values of the variables they share and then "join" the two results. If we allow for the common variables to be only among the *t-nodes*, then the size of the search space becomes $C_1T + C_2T + T^2$ (T is the *t-size* of the common *t-nodes*). $C_1T + C_2T$ accounts for solving the sub-problems for all values of the communication variables, and T^2 accounts for "joining" them. It should be noted however, that if consistency checks involve "simulation", then the T^2 term tends to be negligible (because search over the join-space does not involve simulation). Generalizing the above idea of dynamic programming, it is also possible to characterize n-way splits which partition the original graph into n partitions each of which share communication variables with a subset of the others.

Definition *(Splitting Condition)***:** The *splitting condition* holds for a *t-partition* in a graph G if the sum of the *ct-sizes* of the partitions and the join-size is strictly lesser than the *c-size* of G.

To obtain maximum computational benefits, we have to find a *t-partition* that minimizes the sum of the *ct-sizes* of the resulting partitions and the join-size. This general n-way split is NP-hard to find (easy to prove from the fact that finding the induced width is NP-hard). However, HCBFS employs a heuristic to decompose a large diagnosis problem into optimal sub-problems based on the topological structure of the system. It runs in polynomial time and is always assured of yielding computational benefits (albeit in sub-optimal amounts). The idea is to examine only a polynomial number of 2-way splits and choose the greediest one

ALGORITHM HCBFS (Graph $G = (V, E)$)
  T = *T-Graph* of G
  T' = Partition-Tree formed by m-ordering on moralized T
  E = Edges of T'
  GREEDYSPLIT (G, E)
END HCBFS

ALGORITHM GREEDYSPLIT (Graph G, Candidate-Splits B)
  b_k = BEST-SPLIT (G, B)
  IF (SPLITTING-CONDITION (G, b_k)) **THEN**
    (G_1, G_2) = PARTITION (G, b_k)
    $B_1 = \{b_i | b_i$ is on the same side of b_k as $G_1\}$
    $B_2 = \{b_i | b_i$ is on the same side of b_k as $G_2\}$
    GREEDYSPLIT (G_1, B_1)
    GREEDYSPLIT (G_2, B_2)
  END IF
END GREEDYSPLIT

Figure 3: Hierarchical Conflict-Directed Best First Search

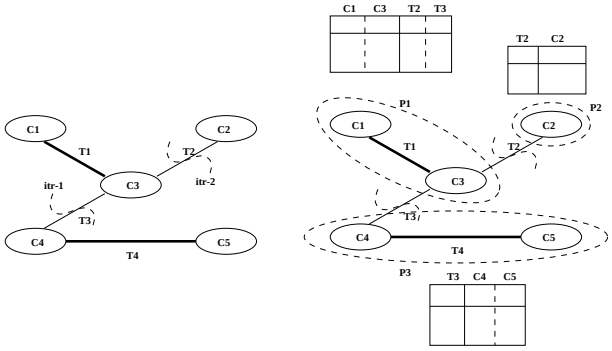


Figure 4: Illustrates the working of HCBFS to produce sub-problems. Thicker edges denote greater communication (*t-size*). P1, P2, P3 are the final partitions. The tables indicate the solutions to diagnosis sub-problems for all values of the surrounding communication variables.

if it satisfies the splitting condition. Such a splitting process is performed recursively until there is no more apparent scope for computational benefits. Interestingly enough, the candidate *t-partitions* that are examined are themselves derived using the m-ordering heuristics. Figure 3 presents the working of HCBFS; and Figure 4 illustrates its working on a small example. The following properties hold true for the HCBFS algorithm.

Property 1: The edges of T' maintain the *running intersection property* [Dechter, 1992] and hence the *t-nodes* constituting the communication variables on any edge form a valid *t-partition*.

Property 2: Let the *c-sizes* of the final partitions be $C_1 \cdots C_k$. The *c-size* of the original graph is therefore $\Pi_{i=1}^{i=k} C_i$. The first time we partition G, it must have been the case that (because of the splitting having to be satisfied) $\Pi_{i=1}^{i=k} C_i > S \times T + T \times R$ (T is the size of the communication; S and R are the *c-sizes* of the two resulting partitions with $S \times R = \Pi_{i=1}^{i=k} C_i$). In later iterations, the effective S and R are only made to decrease recursively and this essentially means that HCBFS is always *safe* in producing computational benefits.

Property 3: The total number of splits considered is clearly linear since they correspond to the edges of T'. Although there are two recursive calls to GREEDYSPLIT, the candidate set of edges that enter them are disjoint and hence GREEDYSPLIT is called only a linear number of times. This proves that the running time of HCBFS is polynomial.

Property 4: Choosing certain edges in a tree as splits results in a set of partitions that themselves form a tree with respect to the split edges (as illustrated in Figure 4). Since we know that optimization in a tree structured network is exponential in the *ct-size* of the largest partition, the complexity of diagnosis using HCBFS is exponential in this parameter.

4.1 Analysis and Implications of HCBFS

We briefly delve into the computational implications of HCBFS. HCBFS facilitates search in two ways. First, it reduces the effective search space by using the dynamic programming paradigm. Second, it propagates "easiness" in constraint checking. Constraint checking in general may not be computationally straightforward — it may often involve system "simulation" of some kind over an extended period of time. It can be noticed however, that constraint checking over the join space is a mere verification that two selected rows of the partition tables have similar values for their communication variables. By using HCBFS, the simulation-based constraint checks are "pushed" to smaller parts of the system (the partitions). Even for consistency checks that do not involve "simulation", implementing a TMS for each small partition is more effective (in terms of the complexity of data structures to be maintained) than one large TMS for the system as a whole.

HCBFS not only reduces the effective *un-amortized* search complexity for a diagnosis call, but also reduces the *amortized* complexity. The solutions to sub-problems occurring for diagnosis calls made in the past can be stored and used for future diagnosis calls when they need to solve the same sub-problems. Eventually, when all sub-problems for all values of communication variables have been solved at least once, a diagnosis call can be answered by doing a search only over the join-space of the partitions. This too (as argued before) is computationally easier than "simulation".

The dynamic programming idea of HCBFS can further be used to pre-process or compile the system description to facilitate diagnosis. Consider a partition of the graph representation of a physical setting. The idea is to solve the diagnosis problem for this partition for all values of the surrounding intermediate variables (*t-nodes*) and store the results. We can then treat this partition as a single physical component that can take any value (mode) corresponding to a combination of the values for each of its surrounding *t-nodes*. The associated probabilities would be derived from the results for the corresponding diagnosis sub-problems. This kind of pre-compilation of the system to treat partitions as components provides computational benefits only if their *t-size* is lesser than their *c-size* (which is often the case).

The space complexity associated with HCBFS has two components. One is the size of the tables associated with the sub-problems. This is referred to as the *table-space* complexity. It is easy to observe that the table space complexity is equal to the sum of the *t-sizes* over all partitions. Another component of the space requirement is the actual space required for the diagnosis algorithms to build the tables and compose them to answer a diagnosis call. This space requirement is identical to the running time complexities associated with solving and composing sub-problems. It is worth noting that the cost of implementing dynamic programming in HCBFS is reflected only in its table-space complexity.

HCBFS also leads to what are called hybrid approaches. These are techniques that combine conflict-based and coverage-based approaches [Kumar, to appear] to solve sub-problems and combine their solutions. Coverage-based algorithms are those that record conflicts and cast the diagnosis problem as a minimum weight hitting set problem [Kurien and Nayak, 2000]. Conflict-based approaches refer to the standard TMS-based algorithms like CBFS and QCBFS. In general, hybrid approaches do the following: (1) Employ the hierarchical partitioning algorithm to reduce the effective search space. (2) Employ one of coverage-based or conflict-based approaches for the sub-problems and the join space.

5 Comparison with Related Work

Related work on trying to leverage structure into the task of diagnosis can be found in [Darwiche, 1998], [Autio and Reiter, 1998], [Provan, 2001] etc. In [Darwiche, 1998], negation normal forms (NNF) are used to represent the consequence of $SD \cup OBS$. Subsequently, minimal cardinality diagnoses are extracted from them using a simple cost propagation and pruning algorithm. For such a procedure to be effective, it is important to ensure the decomposability of the NNF. Decomposability is achieved by partitioning SD to perform a case analysis on the shared atoms that do not appear among the observations. The partitioning choices are inspired by trying to produce a join-tree of the topological structure of the system much like the m-ordering heuristics. The complexity of the algorithm is exponential in the size of the hyper-nodes of

the join tree and linear in the number of such hyper-nodes.

There are at least three important ways in which this approach differs from ours. Firstly, this approach does not reason about probabilities but rather looks for minimal diagnoses (minimizes the number of faulty components). Secondly (and more importantly), it tries to produce diagnoses (minimal) by maintaining at each stage, a representation for all the consistent candidates. The optimization phase (of producing minimal candidates) occurs as a separate phase. Usually, we are not interested in all consistent diagnoses and trying to represent them at any stage when there could potentially be an exponentially large number of them can be a bottleneck. In our approach, the optimization and satisfaction phases are interleaved. This allows us to produce candidates as and when we want them, in decreasing order of their optimization values, and to prune the search space using both optimality- and satisfiability-reasoning. Thirdly, if the number of intermediate variables is too many, achieving decomposability in the NNF is exponential in the induced width of the moralized *T-Graph*; but since we are interested only in the behavior modes of components and not that of intermediate variables, the search space may be significantly reduced using our approach when the components are "tightly" coupled.

In [Provan, 2001] the idea of hierarchical diagnosis has a different meaning. It is based on the use of abstraction operators to define an abstraction hierarchy of the model (a lattice induced by a set of partitions of the system variables). A group of components and intermediate variables at a particular abstraction level are "merged" to form "abstract" components at a higher level with appropriately defined inputs, outputs and constraints relating them. A structural abstraction sc of subcomponents $c_1 \cdots c_k$ defines two modes of behavior for sc — $AB(sc)$ and $\neg AB(sc)$ with the constraint that $\neg AB(c_1) \cdots \neg AB(c_k) \rightarrow \neg AB(sc)$. Such an abstraction mechanism is useful only for isolating a group of components all of which cannot be behaving in their nominal modes (abstract models isolate diagnoses only at the abstract level, but more efficiently). At each level of abstraction we only define the nominal mode of behavior for the abstract component. The only other implicit mode is the faulty mode. This limits the scope of diagnosis even at the abstract levels. Under a combinatorial optimization formulation of the diagnosis problem, abstraction of $c_1 \cdots c_k$ to sc only defines what happens when all components $c_1 \cdots c_k$ are behaving in their most probable modes (nominal mode for sc). It does not say anything about what probabilities are associated or what happens with any of the other remaining exponentially large number of non-nominal modes. This makes diagnosis not only infeasible at more detailed levels, but also information-lossy at abstract levels.

6 Conclusions

In this paper, we employed the combinatorial optimization characterization of the diagnosis problem. We compared two different approaches that exploit different features of the problem: (1) naive m-ordering exploits the structure of the system by leveraging the causal dependencies among the variables (*T-Graph*) (2) CBFS exploits the fact that the output is uniquely determined for given inputs to a component behaving in a known mode, and that we are interested only in an assignment to the component modes of the system. We observed that naive m-ordering performs poorly when there is high interconnectedness among components and that CBFS performs poorly when there is low coupling. We proposed a computationally feasible algorithm called HCBFS (extending on CBFS) to achieve the best of both the worlds. HCBFS uses CBFS in tightly coupled parts of the system and m-ordering to identify them. We showed that HCBFS has many important implications on the complexity of diagnosis — reduces the un-amortized complexity of a diagnosis call, reduces amortized complexity of a diagnosis call by reusing computation done for sub-problems arising in past diagnosis calls, allows pre-compilation of the system description to facilitate diagnosis, and enhances hybrid algorithms. Finally, we compared and contrasted our work with somewhat related approaches.

References

[Autio and Reiter, 1998] Autio K. and Reiter R. Structural Abstractions in Model-Based Diagnosis. *In Proceedings of ECAI'98. Pages: 269-273.*

[Darwiche, 1997] Darwiche A. New Advances in Structure-Based Diagnosis: A Method for Compiling Devices. *In Proceedings of the Eighth International Workshop on Principles of Diagnosis (DX'1997). Pages: 35-42.*

[Darwiche, 1998] Darwiche A. Model-Based Diagnosis Using Structured System Descriptions. *Journal of Artificial Intelligence Research 8: 165-222.*

[Darwiche and Provan, 1997] Darwiche A. and Provan G. The Effect of Observations on the Complexity of Model-Based Diagnosis. *In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97). Pages: 99-104.*

[Dechter, 1992] Dechter R. Constraint Networks. *Encyclopedia of Artificial Intelligence, second edition, Wiley and Sons, Pages: 276-285, 1992.*

[de Kleer *et al.*, 1992] de Kleer J., Mackworth A. K., and Reiter R. Characterizing Diagnoses and Systems. *Artificial Intelligence 56 (1992) 197-222.*

[de Kleer and Williams, 1989] de Kleer J. and Williams B. C. Diagnosis with Behavioral Modes. *In Proceedings of IJCAI'89. Pages: 104-109.*

[Forbus and de Kleer, 1992] Forbus K. D. and de Kleer J. Building Problem Solvers. *MIT Press, Cambridge, MA, 1992.*

[Hamscher *et al.*, 1992] Hamscher W., Console L., and de Kleer J. Readings in Model-Based Diagnosis. *Morgan Kaufmann, 1992.*

[Kumar, 2001] Kumar T. K. S. QCBFS: Leveraging Qualitative Knowledge in Simulation-Based Diagnosis. *Proceedings of the Fifteenth International Workshop on Qualitative Reasoning (QR'01).*

[Kumar, 2002] Kumar T. K. S. A Model Counting Characterization of Diagnoses. *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis (DX'02).*

[Kurien and Nayak, 2000] Kurien J. and Nayak P. P. Back to the Future for Consistency-Based Trajectory Tracking. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00).*

[Lucas, 1997] Lucas P. A Theory of Diagnosis as Hypothesis Refinement. *Proceedings of NAIC'97.*

[Nayak and Williams, 1997] Nayak P. P. and Williams B. C. Fast Context Switching in Real-time Propositional Reasoning. *In Proceedings of AAAI-97.*

[Provan, 2001] Provan G. Hierarchical Model-Based Diagnosis. *In Proceedings of the Twelfth International Workshop on Diagnosis (DX'01).*

[Struss, 1988] Struss P. Extensions to ATMS-based Diagnosis. *In J. S. Gero (ed.), Artificial Intelligence in Engineering: Diagnosis and Learning, Southampton, 1988.*

[Struss and Dressler, 1989] Struss P. and Dressler O. "Physical Negation" - Integrating Fault Models into the General Diagnosis Engine. *In Proceedings of IJCAI-89. Pages: 1318-1323.*

[Tarjan and Yannakakis, 1984] Tarjan R. E. and Yannakakis M. Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput. 13, pages 566-576, 1984.*

[Williams and Nayak, 1996] Williams B. C. and Nayak P. P. A Model-Based Approach to Reactive Self-Configuring Systems. *In Proceedings of AAAI-96. Pages: 971-978.*

Possible conflicts, ARRs, and conflicts

Belarmino Pulido Junquera and **Carlos Alonso González**
Departamento de Informática. Universidad de Valladolid
Edificio Tecnologías de la Información y las Telecomunicaciones
E-47011. Valladolid (Spain) ***e-mail*****:** {**belar,calonso**}**@infor.uva.es**
Phone: +34 983 42 36 70 Fax: +34 983 42 36 71

Abstract. Consistency-based diagnosis is the most widely used approach to model-based diagnosis within the Artificial Intelligence community. It is usually carried out through an iterative cycle of behavior prediction, conflict detection, and candidate generation and refinement. Many approaches to consistency-based diagnosis have relied on some kind of on-line dependency-recording mechanism for conflict calculation. These techniques have had different problems, specially when applied to dynamic systems. Recently, off-line compilation of dependencies has been established as a suitable alternative approach. In this work we compare one compilation technique, based on the *possible conflict* concept, with results obtained with the classical on-line dependency recording engine as in GDE. Moreover, we compare possible conflicts with another compilation technique coming from the FDI community, which is based on analytical redundancy relations. Finally, we study the relationship between possible conflicts, analytical redundancy relations, and conflicts.

1 Introduction

For more than thirty years different techniques have been applied to diagnose systems in multiple domains. Diagnosis has been carried out through knowledge-based systems, case-based reasoning, model-based reasoning, and so on. This work is focused in the model-based approach to diagnosis. Moreover, we will only talk about diagnosis of physical devices [18].

More specifically, consistency-based diagnosis is the most widely used approach to model-based diagnosis within the Artificial Intelligence community (usually known as DX). It is a research field that has reported successful results in recent years [39, 7]. This approach has proven its maturity, both in theory, and in practice. On the one hand, the diagnosis process and the diagnosis results have been completely characterized from a logical point of view [32, 12], thus facilitating further comparison. On the other hand, consistency-based diagnosis has been successfully applied to a wide variety of domains such as automotive industry [3, 38], bio-medicine [20], nuclear plants [24], or ecology [37].

In such a framework, GDE [13] is the most well known implementation, and *de facto* paradigm. GDE organizes the diagnosis process as an iterative cycle of behavior prediction, conflict detection, and candidate generation and refinement. But conflict computation is a non-trivial step, which has deserved a lot of attention from the consistency-based diagnosis community. In GDE, the set of minimal conflicts is computed by means of an ATMS [11], which records on-line the set of correctness assumptions, or dependencies, used by the inference engine. It should be noticed that dependency-recording can be done forward (whenever new input data are introduced), or backward (when a discrepancy is found, such as in CAEN [2, 21], DYNAMIS [6], or TRANSCEND [25]). Another important feature of the GDE framework is that it calculates labels propagating values through constraints in every possible direction.

However, one problem related to on-line dependency-recording is that the set of labels needs to be computed each time a new different value is introduced. Another problem was found in the combined use of on-line dependency-recording together with qualitative models for diagnosing dynamic systems [17, 14]. Mainly for these reasons several research groups have looked for alternative methods to such a kind of on-line dependency-recording. On the one hand *state-based diagnosis* [36] has emerged as an alternative to *simulation-based diagnosis*, just for qualitative models. On the other hand, *topological methods* propose to explicitly use the structural description of the system to be diagnosed. This information is implicitly stated in the system description. Within this last approach, we make difference of two major trends: those methods that use other on-line dependency-recording than ATMS (by exploring causal-graphs [2, 24], signed directed graphs [26], or other topological and functional structures [5]), and those methods that perform off-line dependency-recording.

Last techniques are also known as compilation methods within the DX community. The main idea supporting this approach is that redundancy within the models can be found off-line. A similar idea was used in the Control Engineering community (or FDI), where Staroswiecki and Declerk proposed to use Analytical Redundancy Relations (ARRs for short), for fault detection and localization [34]. Given such a similarity, there is an ongoing interest from the DX and the FDI communities in comparing their approaches.

Between the FDI and AI proposals, Lunze and Schiller [23] were able to perform diagnosis using causal graphs associated with over-constrained systems. These systems were obtained from the logical formula in the models of the system.

Within the DX community we have found the following compilation techniques:

- Darwiche and Provan [10] characterized the set of diagnoses using the consequence concept [9], instead of using the conflict concept. Analyzing the system structure, those sub-systems which could lead to a diagnosis can be found off-line.
- Similar information is used by Steele and Leitch [35] to refine the set of candidates, in an adaptive approach to diagnosis [4].
- In DOGS, Loiez and Taillibert [22] proposed to localize, off-line, over-constrained sets of equations. They were looking for those sub-systems capable to become conflicts. The work done is con-

ceptually equivalent to that in [34], as it has been stated in [8].

- Fröhlich and Nejdl [15] used structural information two-fold: they analyzed the whole set of logical formula in the model to find sub-sets of formula capable to generate diagnosis, and they benefit from these sub-sets in order to refine the whole set of diagnosis candidates.
- Pulido and Alonso [27, 28] proposed to organize consistency-based diagnosis around the *possible conflict* concept. A possible conflict is a sub-system in system description which is capable to become a conflict, within the GDE framework.

In this work we revisit the compilation technique based on the *possible conflict* concept [27, 28]. Initially we summarize the characterization of that concept, in order to compare possible conflicts against real conflicts. Later on, we establish the relationship between possible conflicts and ARRs. Finally, we revisit the work by Cordier et al. [8] in order to compare conflicts and ARRs from a computational point of view.

Due to space limitations we do not compare possible conflicts and other compilation techniques from the DX community. Such a comparison can be found in [28, 30].

2 The possible conflict concept

Main assumptions in this work are that there is no structural fault, and it is possible to know beforehand the number and placement of available observations (sensors). An additional assumption is that the model of the system can be expressed as a set of constraints: quantitative or qualitative, linear or not, algebraic or not.

In Reiter's framework for model-based diagnosis [32] a minimal conflict identifies a set of constraints containing enough redundancy to perform diagnosis. In the most simple case, when constraints are made up of equations, a minimal conflict would denote a strictly over-determined system[1].

As it was mentioned in the previous Section, shared basis in compilation techniques is: the set of analytically redundant sub-systems, which can be used for diagnosis purposes, can be computed off-line. Moreover, it has been proven that GDE provides all the existing minimal conflicts. Since the set of possible conflicts tries to be a computational alternative to on-line dependency recording for conflict computation, we have imposed an additional requirement: over-constrained sub-systems should be the same as the set of minimal conflicts computed by GDE [2].

Finding analytical redundancy is a necessary but not a sufficient condition for a system to be suitable for consistency-based diagnosis purposes. The system must also be solved using local propagation alone[3]. To fulfill both requirements we have split the search process into two phases. First, we look for over-determined systems. Second, we check whether these systems can be solved using local propagation alone. To do so, we just need abstractions of model-description. For the sake of readability, below we include a summary of definitions the reader can find in [27, 28].

[1] In an over-determined system the number of equations, e, is greater than the number of unknowns, u: $e \geq u+1$. In a strictly over-determined system, $e = u+1$.

[2] For this reason, we always assume that we have the same model (system description or *SD* in Reiter's terminology) as GDE has.

[3] Current consistency-based diagnosis systems do not impose that constraint [19]. In [30] we extended the possible conflict concept to deal with such (cyclical) configurations.

2.1 Searching for over-determined systems

We have represented the model in SD as a hyper-graph: $H_{SD} = \{V, R\}$ which is made up of:

- $V = \{v_1, v_2, \ldots, v_n\}$, the set of variables in the model. It is made up of observed OBS, and not observed or unknown variables, $NOBS$: $V = OBS \bigcup NOBS$.
- $R = \{r_1, r_2, \ldots, r_m\}$ is a family of sub-sets in V, where each r_k represents a constraint in the model, and it contains some model variables, observed and not observed ones.

We have called *Evaluation Chains* the over-constrained sub-systems in H_{SD} (in Appendix A the reader can find definitions for terminology in graphs and hyper-graphs c.f. [16, 1]):

Evaluation chain: $H_{ec} \subseteq H_{SD}$ is a partial sub-hypergraph in H_{SD}: $H_{ec} = \{V_{ec}, R_{ec}\}$, where $V_{ec} \subseteq V$, $R_{ec} \subseteq R$, and $X_{ec} = V_{ec} \cap NOBS$ is the set of unknowns in V_{ec}, and H_{ec} verifies:

1. H_{ec} is a connected hypergraph,
2. $V_{ec} \cap OBS \neq \emptyset$,
3. $\forall v_{no} \in X_{ec} \Rightarrow d_{H_{ec}}(v_{no}) \geq 2$,
4. let $G(H_{ec})$ be a bipartite graph made up of two kinds of nodes: $x \in X_{ec}$, and $r_{i_{ec}} \in R_{ec}$, such that two nodes are linked in $G(H_{ec})$ if and only if $x \in r_{i_{ec}}$. Then, $G(H_{ec})$ has a *matching* with maximal cardinality $m' = |X_{ec}|$ and $|R_{ec}| \geq m' + 1$.

Figure 1 shows a classical example in consistency-based diagnosis. In order to make difference of components and constraints, we will use capital letters for components, and small letters for constraints in their models. m_i and a_j denote the models of multipliers and adders, respectively. Each model is made up of just one constraint; for instance, $m_1 = \{A, C, X\}$. Whenever a model has more than one constraint, indices are used to distinguish them. The related hyper-graph is

$$H_{polybox} = \{\{A, B, C, D, E, F, G, X, Y, Z\}, \{m_1, m_2, m_3, a_1, a_2\}\}$$

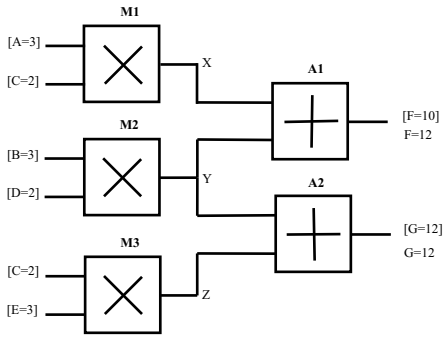


Figure 1. Classical polybox example in the consistency-based diagnosis. Observed values are in brackets. $\{X, Y, Z\}$ are non-observed values.

Since we are interested in minimal conflicts, only minimal evaluation chains, MEC for short, are useful.

Minimal Evaluation Chain : H_{ec} is a minimal evaluation chain if there is no evaluation chain $H'_{ec} \subset H_{ec}$.

The set of minimal Evaluation chains, SMEC, is built based on the algorithms: *build-every-mec()*, *build-mec()*, and *justify()* which perform depth-first search in H_{SD} using backtracking. All these algorithms can be found in Appendix B. In the polybox example, these

algorithms have found three MECs:

$$H_{ec_1} = \{\{A, B, C, D, F, X, Y\}, \{m_1, m_2, a_1\}\}$$
$$H_{ec_2} = \{\{B, C, D, E, G, Y, Z\}, \{m_2, m_3, a_2\}\}$$
$$H_{ec_3} = \{\{A, C, E, F, G, X, Y, Z\}, \{m_1, a_1, a_2, m_3\}\}$$

2.2 Can an evaluation chain be solved?

A minimal conflict is a strictly over-determined system that we want to solve using local propagation alone. However, the hyper-graph has not enough information about how each constraint can be solved. To tackle this problem, we create an AND-OR graph for each minimal evaluation chain. In such a graph, there is one or more AND-OR arcs for each hyper-arc in the MEC. Each AND-OR arc represents one way the hyper-arc could be solved. In fact, to solve a MEC, we should select one AND-OR arc from each constraint. As a consequence, choosing different AND-OR arcs from the AND-OR graph generates different ways of solving the MEC. Moreover, the over-determined system can only be solved using local propagation criteria. Each one of the different ways of solving a MEC is called a Minimal Evaluation Model, or MEM.

For instance, each constraint (m_i or a_i) used to model the polybox system provides three different interpretations to the AND-OR graph:

$$m_i(v_{out}, v_{in_1}, v_{in_2}) \Rightarrow \begin{cases} m_{i_1} \equiv v_{out} = v_{in_1} \times v_{in_2} \\ m_{i_2} \equiv v_{in_1} = v_{out}/v_{in_2},\ if\ v_{in_2} \neq 0 \\ m_{i_3} \equiv v_{in_2} = v_{out}/v_{in_1},\ if\ v_{in_1} \neq 0 \end{cases}$$

Interpretations for a constraint are usually obtained when applying the invertibility criterion. Nevertheless, there are additional criteria. Appendix D shows constraints used to model a physical system made up of tanks, pumps and valves. Constraints $tr1_3, t2_3, tr2_5$ are used to compute the mass in a tank. In such kind of constraint, just one interpretation is allowed, since we have taken an integration approach:

$m_T(t) = \int m'_T(t-1)dt + m_T(t-1)$

This interpretation can not be reversed. Hence, additional concepts are necessary to define a Minimal Evaluation Model.

Given the relation between $r_{i_{ec}} \in R_{ec}$, and the set of AND-OR arcs $r_{i_{k_{em}}}$, derived from $r_{i_{ec}}$, we can state the following proposition.

Proposition 1 *Let $AOG(H_{ec}) = \{V_{em}, R_{em}\}$ be the AND-OR graph obtained from $H_{ec} = \{V_{ec}, R_{ec}\}$ applying the local resolution criterion, where:*

- $V_{em} = V_{ec}$,
- $\forall r_{i_{ec}} \in R_{ec} \Rightarrow \exists r_{i_{k_{em}}} \in R_{em}, k \geq 1$

Then, $r_{i_{ec}} \in R_{ec}$ induces a partition in R_{em}.

Proof: Each $r_{i_{ec}} \in R_{ec}$ induces an equivalence class in R_{em}. By definition, it induces a partition too.

Leaf node: v_i is a leaf node in graph H iff $\hat{\Gamma}^{-1}_{v_i} = 0$.
Discrepancy node: v_i is a discrepancy node in graph H iff

- $(d^-_H(v_i) \geq 2 \ \wedge \ v_i \in NOBS)$, or
- $(d^-_H(v_i) \geq 1 \ \wedge \ v_i \in OBS)$

That is, a leaf node has no predecessors, and a discrepancy node can be found in two different ways: estimating an observed variable, or doing a double estimation for an unknown variable.

Minimal Evaluation Model: A partial AND-OR graph, $H_{mem} \subseteq AOG(H_{ec})$, where $H_{mem} = \{V_{mem}, R_{mem}\}$, is a minimal Evaluation model iff:

1. R_{mem} is a minimal hitting-set for the partition induced by $r_{i_{ec}} \in R_{ec}$ in R_{em},
2. $(\forall v_i \mid v_i \in V_{mem}$ and v_i is a leaf node$) \Rightarrow v_i \in OBS$,
3. $\exists_1 x_j \in V_{mem} \mid x_j$ is a discrepancy node,
4. if x_j is a discrepancy node, then there exists a directed and acyclic path in $H_{mem} : \{x_i, x_{i+1}, \ldots, x_{i+k}, x_j\}$ from each node x_i to x_j.

Algorithms used to calculate every MEM for each MEC: *build-every-mem()*, and *build-mem()*, are given in Appendix C. These algorithms are exhaustive too, since they perform depth-first search using backtracking. For instance, MEC H_{ec_1} has a related AND-OR graph:

$$AOG(H_{ec_1}) = \{\{A, B, C, D, F, X, Y\},$$
$$\{m_{1_1}, m_{1_2}, m_{1_3}, m_{2_1}, m_{2_2}, m_{2_3}, a_{1_1}, a_{1_2}, a_{1_3}\}\}$$

Given H_{ec_1} and the set of available interpretations in $AOG(H_{ec_1})$, algorithm *build-mem()* is able to find seven different MEMs[4]:

MEMs	**Equivalent to evaluate the expression**
$\{m_{1_1}, m_{2_1}, a_{1_1}\}$	$F_{obs} \equiv F_{pred} = A \times C + B \times D$
$\{m_{1_1}, m_{2_1}, a_{1_2}\}$	$X_{pred_1} = A \times C \equiv X_{pred_2} = F - B \times D$
$\{m_{1_2}, m_{2_1}, a_{1_2}\}$	$A_{obs} \equiv A_{pred} = (F - B \times D)/C,\ if\ C \neq 0$
$\{m_{1_3}, m_{2_1}, a_{1_2}\}$	$C_{obs} \equiv C_{pred} = (F - B \times D)/A,\ if\ A \neq 0$
$\{m_{1_1}, m_{2_1}, a_{1_3}\}$	$Y_{pred_1} = F - (A \times C) \equiv Y_{pred_2} = B \times D$
$\{m_{1_1}, m_{2_2}, a_{1_3}\}$	$B_{obs} \equiv B_{pred} = (F - A \times C)/D,\ if\ D \neq 0$
$\{m_{1_1}, m_{2_3}, a_{1_3}\}$	$D_{obs} \equiv D_{pred} = (F - A \times C)/B,\ if\ B \neq 0$

It should be noticed that a MEC would provide no MEM if the over-determined system can not be solved using available interpretations and local propagation. In [31] the reader can find additional information on how temporal information has been included in this framework and one example of a MEC which can not provide any MEM.

Once summarized the possible conflict concept, next section studies the relationship between MECs, and MEMs, which are computed off-line, and real conflicts computed on-line.

3 Conflicts and possible conflicts

If evaluated, a MEM could lead to discrepancy, i.e., it could lead to a conflict. However, the set of MEM is computed off-line, without any model evaluation. And conflicts would appear only when observations are introduced and the evaluation model is computed. So, we have introduced the following concept:

Possible conflict: The set of constraints in a Minimal Evaluation Chain giving rise to, at least, one Minimal Evaluation Model.

For example, in the polybox system in Figure 1, there are three possible conflicts: $\{\{m_1, m_2, a_1\}, \{m_1, a_1,\ a_2, m_3\}, \{m_2, m_3, a_2\}\}$, because every MEC has, at least, one MEM.

In such a case, where component models are made up of only one relation, the set of possible conflicts is equivalent to the set of minimal conflicts in Reiter's terminology computed on-line by GDE, whatever the faults and whatever the set of available observations.

At this point it is necessary to answer the following question: is the set of possible conflicts equivalent to the set of minimal conflicts computed on-line by GDE? In order to answer, we need additional definitions:

$P(S)$**:** is the set of subsets in S;

[4] Since the MEM will have the same set of variables as MEC, we just include the set of interpretations.

$model : COMPS \rightarrow P(R_{SD})$**:** $model(C)$ identifies the family of relations modelling C behavior;

$comp : R_{SD} \rightarrow COMPS$**:** $r_i \rightarrow comp(r_i) = \{C \mid r_i \in model(C)\}$:
$comp(r_i)$ indicates the component containing relation r_i in its model.

Proposition 2 *Let co be a minimal conflict found by GDE, and co is related to a discrepancy in $v \in V_{SD}$: there is a minimal evaluation chain, $H_{ec} = \{V_{ec}, R_{ec}\}$, such that:*
$v \in V_{ec}$ *and* $co = \bigcup_{r_i \in R_{ec}} comp(r_i)$

Proof: GDE solves a minimal over-determined system to find a minimal conflict related to v [19]. Since *build-every-mec()* performs exhaustive search, it is able to find every minimal over-determined system in H_{SD}. Hence, it will find that over-determined system too.

Hence, once GDE finds a minimal conflict, *build-every-mec()* will find a MEC containing the same set of constraints which were used to find a conflict. Those constraints belong to the same set of components.

Proposition 3 *Let co be a minimal conflict found by GDE, and co is related to a discrepancy in $v \in V_{SD}$: there is a minimal evaluation model, $H_{me} = \{V_{em}, R_{em}\}$, that can obtain a discrepancy in v, and $v \in V_{em}$ and $co = \bigcup_{r_i \in R_{em}} comp(r_i)$*

Proof: By proposition 2, there is a MEC related to co, such that:

$$co = \bigcup_{r_i \in R_{ec}} comp(r_i)$$

Moreover *build-every-mem()* performs an exhaustive search too. Therefore, it will find every MEM related to such MEC, i.e., every possible way the MEC can be solved. Hence, it will find the over-determined system used to obtain the minimal conflict. Also, each $r_{i_k} \in R_{em}$ is an interpretation for some $r_i \in R_{ec}$. Hence:

$$co = \bigcup_{r_{i_k} \in R_{em}} comp(r_i)$$

At least one of the MEM related to the CEM will find a discrepancy in v, in the same way the GDE does.

Unfortunately, the number of MEMs for each MEC is exponential in the average number of interpretations for each hyper-arc in the MEC. Due to practical reasons we just select one MEM related to a MEC. Based on that MEM, we build an executable model which is used for fault detection. In [31] the reader can find a detailed description of how possible conflicts can be used to perform consistency-based diagnosis for both static and dynamic systems.

Nevertheless, it is still possible to claim that the set of possible conflicts is theoretically equivalent to the set of conflicts found online by means of GDE. We will show this fact in next two propositions.

Proposition 4 *If H_{ec} is a MEC, H_{em} is one of its MEMs and the evaluation of the executable model associated to H_{em} generates a discrepancy in $v \in V_{em}$, then GDE will find a discrepancy in v.*

Proof: There is a discrepancy in v related to the evaluation of a MEM. The MEM is an strictly over-determined system. Moreover, GDE finds any discrepancy related to any minimal over-determined system. Hence, it will find the discrepancy in v too.

This proposition always holds. Unfortunately, the converse does not hold universally, because we can not guarantee for an arbitrary set of non-linear constraints that every MEM for a MEC will provide the same solution for a given set of observations [40]. This assumption should be stated in the following way:

Equivalence assumption : Every MEM in a MEC provides the same set of solutions for any given set of input observations.

Now, it is possible to define the following proposition:

Proposition 5 *If GDE finds a minimal conflict, co, related to a discrepancy in v, and* ***the equivalence assumption holds*** *for a H_{ec} containing v, then the possible conflict related to H_{ec} will be confirmed as a minimal conflict.*

Proof: The proof is straightforward based on propositions 2, and 3.

4 Comparing possible conflicts, conflicts, and ARRs

As previously mentioned, there is an on-going research interest from the DX and FDI communities in comparing their approaches. Recently, Cordier et al. [8] proposed a common framework to compare conflicts and ARRs [34, 33]. In that trend, we compare ARRs and possible conflicts considering the way they are computed. Afterwards, we discuss results in [8] and extract some conclusions.

4.1 Possible conflicts and ARRs

The set of ARRs is obtained from the unique canonical decomposition of the structural description of the system into under-determined, just-determined, and over-determined sets of constraints. The canonical decomposition is based on finding a complete matching, w.r.t. unknown variables, in the bipartite graph associated to the structural description of the system. Combination of just-determined systems together with redundant relations is the basis for an *Analytical Redundancy Relation*[34].

Each complete matching can be considered as a causality assignment, but it is necessary to obtain a causal matching for the over-determined system, from the set of causal matchings satisfying the invertibility condition [33]. Each ARR can be solved and used for diagnosis purposes once observed values are introduced.

It should be noticed that all the steps, except the solving one, could be done off-line. Hence, computing ARRs is a compilation technique in FDI. And, it seems obvious that strong similarities do exist between the way ARRs and possible conflicts are computed.

- Both methods search for over-determined sub-systems. Direct or deduced ARRs can be used to estimate a value for an observed variable in the system. Moreover, algorithms used for computing MEC, can be used to obtain the whole set of over-determined subsystems[5]. Hence, the algorithms will find an evaluation chain with the same set of constraints as of the ARR.
- An ARR need a causal matching, because not every causality assignment can be done in the complete matching. In the same way, AND-OR arcs are introduced to limit the ways an hyper-arc can be solved. It seems obvious that one of the evaluation models for an evaluation chain will be equivalent to the causal matching in the ARR.

[5] It is straightforward to modify algorithm *Justify()* to search for any over-determined system.

- The set of evaluation models for an evaluation chain are built based on local propagation criterion, i.e., the evaluation model does not contain any cycle. This condition has been imposed in the ARR approach too. For this reason, the ARR is obtained once graph reduction, by means of loop elimination, has been done in the causal graph [33]. This step is equivalent to loop elimination in the possible conflict approach [29].

  However, there are some differences:

- Staroswiecki et al. [33] assume that in an over-determined system the set of unknowns can be computed in different ways, using constraints and known values, and "deduced redundancy relations are obtained writing that all these results have to be the same". This assumption is the same as the equivalence assumption in the previous section.
  As mentioned above, that assumption is never done in GDE while computing minimal conflicts, because the assumption does not hold universally for physical systems made up of general non-linear constraints [40]. Therefore, based on propositions 4 and 5, it can not be claimed that model-based diagnosis relying upon ARRs and consistency-based diagnosis using conflicts will provide always the same set of results. Results obtained using ARRs would be the same as of those obtained using just one MEM for each MEC. These results can be sub-optimal, w.r.t. the number of detected conflicts, unless the equivalence assumption holds.
- Moreover, *build-every-mec()* provides the whole set of minimal evaluation chains, because we look for minimal conflicts. This is not guaranteed in the original ARR approach, which should be revised to find just minimal ARRs.

4.2 Discussion

Cordier et al. [8] defined the *support* for an ARR as "the set of components involved in the ARR". This term was also called "potential R-conflict", because of their Proposition 4.1:

> "Let OBS be a set of observations for a system modeled by SM (resp. SD). There is an identity between the set of minimal R-conflicts for OBS and the set of minimal potential R-conflicts associated to the ARRs which are not satisfied by OBS."

As stated in the previous section, we think it is necessary to make three explicit assumptions to guarantee that such a conclusion holds universally:

- the equivalence assumption holds,
- the set of ARRs is built based on minimality criteria, and
- we have a component-oriented behavior description of the system, but minimality is considered w.r.t. sets of constraints.

Regarding first two conditions, it seems obvious that proposition 5 in Section 3 is equivalent to proposition 4.1. in [8] when both assumptions hold. Third assumption must be taken into account when behavioral models are made up of more than one constraint. Minimality w.r.t. sets of constraints is needed because not every possible conflict is equivalent to a minimal conflict in Reiter's framework. We will illustrate this using the system in Figure 2. The system is made up of common components in process industry such as tanks, pumps, valves, and so on.

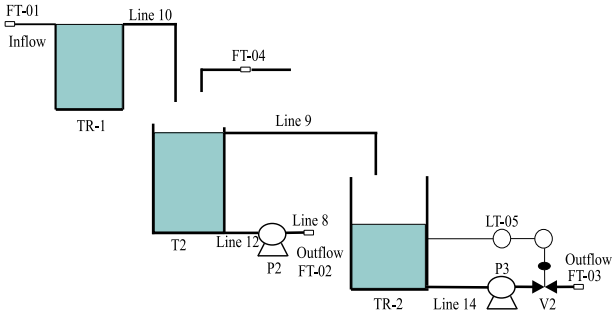


Figure 2. Scheme of the system to be diagnosed. Measured variables are flows $FT01 = f_1^*$, $FT02 = f_8^*$, $FT03 = f_7^*$, and $FT04 = f_{11}^*$; level of tank $LT05 = h_{TR2}^*$, and the value of the control action on valve $V_2 = u_2$ at the output of tank $TR2$.

Its related hyper-graph can be described as:

$\mathbf{H}_{SD} = \{V_{SD}, R_{SD}\}$;
$\mathbf{V}_{SD} = \{OBS \cup NOBS\}$;
$\mathbf{O}BS = \{f_1^*, f_7^*, f_8^*, f_{11}^*, h_{TR_2}^*\}$;
$\mathbf{N}OBS = \{f_9, f_{10}, f_{12}, f_{14}, m'_{TR_1}, m_{TR_1}, h_{TR_1}, m'_{T_2}, m_{T_2}, h_{T_2}, m'_{TR_2}, m_{TR_2}, h_{TR_2}, \Delta P_{P_2}, \Delta P_{P_3}, P_{1_{TR_1}}, P_{2_{TR_1}}, P_{1_{T_2}}, P_{2_{T_2}}, P_{1_{TR_2}}, P_{2_{TR_2}}, u_{cont_2}\}$
$\mathbf{R}_{SD} = \{tr1_1, tr1_2, tr1_3, tr1_4, t2_1, t2_2, t2_3, t2_4, t2_5, p2_1, p2_2, p2_3, p3_1, p3_2, p3_3, v2_1, v2_2, tr2_1, tr2_2, tr2_3, tr2_4, tr2_5, tr2_6\}$

The meaning for each equation above can be found in Appendix D. We have used common equations for computing mass balances, overflows, and so on. Analyzing the system we have found three possible conflicts. The reader should notice that PC_3 is minimal w.r.t. constraints, but not minimal w.r.t. components.

$\mathbf{PC}_i$	**Components**
$\{tr1_1, tr1_2, tr1_3, tr1_4, t2_1, t2_2, t2_3, t2_4, t2_5, p2_1, p2_2, p2_3\}$	$\{TR_1, T_2, P_2\}$
$\{tr2_1, tr2_3, p3_1, p3_2, p3_3, v2_1, v2_2\}$	$\{TR_2, P_3, V_2\}$
$\{tr1_1, tr1_2, tr1_3, tr1_4, t2_1, t2_2, t2_3, t2_5, p2_3, tr2_4, tr2_5, tr2_6, p3_3, v2_1\}$	$\{TR_1, T_2, P_2, TR_2, P_3, V_2\}$

5 Conclusions

In this paper we have shown that compilation of dependencies by means of the possible conflict approach is theoretically equivalent to on-line dependency recording in GDE. However, it is not possible to claim that, in practice, consistency-based diagnosis using possible conflicts provides the same results as GDE does, unless the equivalence assumption holds.

We have found out that the model of an ARR is equivalent to some evaluation model for an evaluation chain. Since we select just one MEM for each MEC for practical reasons, we conclude that both approaches can obtain equivalent results (assuming ARRs are computed based on minimality criteria).

Finally, we have concluded that Proposition 4.1 in [8] need to be revised taking into account results in propositions 4 and 5, and considering minimality criteria w.r.t. constraints.

Acknowledgements

Authors wants to thank Louise Travé-Massuyès for critical reading of the original manuscript, and three anonymous referees for their valuable comments.

REFERENCES

[1] C. Berge, *Hypergraphs. Combinatorics of Finite Sets*, North-Holland, Amsterdam, 1989.

[2] K. Bousson, J.-P. Steyer, L. Travé-Massuyès, and B. Dahhou, 'From a heuristic-based to a model-based approach for monitoring and diagnosis of biological processes', *Engineering Applications of Artificial Intelligence*, **11**, 447–493, (1998).

[3] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupre, 'Generating on-board diagnostics of dynamic automotive systems based on qualitative models', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis (DX-99)*, pp. 27–35, Loch Awe, Scotland (UK), (1999).

[4] M.J. Chantler, G.M. Coghill, Q. Shen, and R.R. Leitch, 'Selecting tools and techniques for model-based diagnosis', *Artificial Intelligence in Engineering*, **12**, 81–98, (1998).

[5] L. Chittaro, G. Guida, C. Tasso, and E. Toppano, 'Functional and teleological knowledge in the multimodeling approach for reasoning about physical systems: A case study in diagnosis', *IEEE Transactions on Systems, Man and Cybernetics*, **23**, 1718–1751, (1993).

[6] L. Chittaro, R. Ranon, and J. Lopez Cortes, 'Ship over troubled waters: Functional reasoning with influences applied to the diagnosis of a marine technical system', in *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, pp. 69–78, Val Morin, Quebec, Canada, (1996).

[7] L. Console and O. Dressler, 'Model-based diagnosis in the real world: lessons learned and challenges remaining', in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 1393–1400, Stockholm, Sweeden, (1999).

[8] M.O. Cordier, P. Dague, M. Dumas, F. Levy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès, 'A comparative analysis of ai and control theory approaches to model-based diagnosis', in *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI 2000)*, pp. 136–140, Berlin, Germany, (2000).

[9] A. Darwiche, 'Model-based diagnosis using structured system descriptions', Technical report 97-07, Department of Mathematics, American University of Beirut, Beirut, Lebanon, (1997).

[10] A. Darwiche and G. Provan, 'Exploiting system structure in model-based diagnosis of discrete-event systems', in *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, pp. 93–105, Val Morin, Quebec, Canada, (1996).

[11] J. de Kleer, 'Problem solving with the ATMS', *Artificial Intelligence*, **28**, 197–224, (1986).

[12] J. de Kleer, A.K. Mackworth, and R. Reiter, 'Characterizing diagnosis and systems', in *Readings in Model Based Diagnosis*, 54–65, Morgan-Kauffman Pub., San Mateo, (1992).

[13] J. de Kleer and B.C. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**, 97–130, (1987).

[14] O. Dressler and P. Struss, 'The consistency-based approach to automated diagnosis of devices', in *Principles of Knowledge Representation*, ed., Gerhard Brewka, 269–314, CSLI Publications, Standford, (1996).

[15] P. Froelich and W. Nejdl, 'A static model-based engine for model-based reasoning', in *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence (IJCAI-97)*, pp. 446–471, Nagoya, Japan, (1997).

[16] M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley and Sons, New York, 1984.

[17] T. Guckenbiehl and G Schaefer-Richter, 'SIDIA: Extending prediction based diagnosis to dynamic models', in *Proceedings of the First International Workshop on Principles of Diagnosis (DX-90)*, pp. 74–82, Stanford, California, USA, (1990).

[18] W.C. Hamscher, L. Console, and J. de Kleer (Eds.), *Readings in Model based Diagnosis*, Morgan-Kaufmann Pub., San Mateo, 1992.

[19] G. Katsillis and M.J. Chantler, 'Can dependency-based diagnosis cope with simultaneous equations?', in *Proceedings of the Eigth International Workshop on Principles of Diagnosis (DX-97)*, pp. 51–59, Le Mont Saint Michel, France, (1997).

[20] M. Lafon, J. Pastor, L. Travé-Massuyès, B. Doyon, J.F. Demonet, and P. Celsis, 'Qualitative modeling of cerebral information propagation mechanisms', in *Proceedings of the Third International Conference on Computational and Neurosciences*, volume 2, pp. 21–23, Research Triangle Park, USA, (1998).

[21] A. Ligeza and B. Gorny, 'Systematic conflict generation in model-based diagnosis', in *Proceedings of the*, pp. 1103–1108, (2000).

[22] E. Loiez and P. Taillibert, 'Polynomial temporal band sequences for analog diagnosis', in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 474–479, Nagoya, Japan, (1997).

[23] J. Lunze and F. Schiller, 'Logic-based process diagnosis utilising the causal structure of dynamical systems', in *Proceedings of the Artificial Intelligence in Real-Time Control (IFAC/IFIP/IMACS)*, pp. 649–654, Delft, Holland, (1992).

[24] P.J. Mosterman, *Hybrid dynamic systems: a hybrid bond graph modeling paradigm and its applications in diagnosis*, Phd thesis, Vanderbilt University, Nashville, Tennessee, USA, May 1997.

[25] P.J. Mosterman and G. Biswas, 'Monitoring, prediction and fault isolation in dynamic physical systems', in *Proceedings of the Fourteenth AAAI National Conference on arificial Intelligence (AAAI-97)*, pp. 100–105, Providence, Rhode Island, USA, (1997).

[26] O.O. Oyeleye, *Qualitative modeling of continuous chemical processes and applications to fault diagnosis*, Phd thesis, Department of Chemical Engineering, M.I.T., Cambridge, Massachusetts, USA, 1990.

[27] B. Pulido and C. Alonso, 'Possible conflicts instead of conflicts to diagnose continuous dynamic systems', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis (DX-99)*, pp. 234–241, Loch Awe, Scotland (UK), (1999).

[28] B. Pulido and C. Alonso, 'An alternative approach to dependency-recording engines in consistency-based diagnosis', in *Artificial Intelligence: Methodology, Systems, and Applications. Ninth International Conference (AIMSA-00)*, volume 1904 of *Lecture Notes in Artificial Intelligence*, 111–120, Springer Verlag, Berlin, Germany, (2000).

[29] B. Pulido and C. Alonso, 'Dealing with cyclical configurations in mordred', in *IX Conferencia Nacional de la Asociación Española de Inteligencia Artificial, (CAEPIA-01)*, pp. 983–992, Gijon, Spain, (2001).

[30] B. Pulido and C. Alonso, 'Revisión del concepto de posible conflicto como técnica de pre-compilación', *Revista Iberoamericana de Inteligencia Artificial*, 41–53, (2001).

[31] B. Pulido, C. Alonso, and F. Acebes, 'Lessons learned from diagnosing dynamic systems using possible conflicts and quantitative models', in *Engineering of Intelligent Systems. Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2001)*, volume 2070. of *Lecture Notes in Artificial Intelligence*, pp. 135–144, Budapest, Hungary, (2001).

[32] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**, 57–95, (1987).

[33] M. Staroswiecki, S. Attouche, and M.L. Assas, 'A graphic approach for reconfigurability analysis', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis (DX-99)*, pp. 250–256, Loch Awe, Scotland (UK), (1999).

[34] M. Staroswiecki and P. Declerk, 'Analytical redundancy in non linear interconnected systems by means of structural analysis', in *Proceedings of the IFAC Advanced Information Processing in Automatic Control (AIPAC-89)*, pp. 51–55, Nancy, France, (1989).

[35] A.D. Steele and R.R. Leitch, 'A strategy for time-constraint qualitative parameter identification', in *Proceedings of the Computational Engineering in Systems Applications (CESA IMACS-IEE/SMC Multiconferece)*, Lille, France, (1996).

[36] P. Struss, 'Fundamentals of model-based diagnosis of dynamic systems', in *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence (IJCAI-97)*, pp. 480–485, Nagoya, Japan, (1997).

[37] P. Struss and U. Heller, 'Model-based support for water treatment', in *Qualitative and Model Based Reasoning for Complex Systems and their Control, Workshop KRR-4 at the Sixteenth International Joint Conference on Artifical Intelligence (IJCAI-99)*, pp. 84–90, Stockholm, Sweeden, (1999).

[38] P. Struss, M. Sachenbacher, and C.M. Carlén, 'Insights from building a prototype for model-based on-board diagnosis of automotive systems', in *Proceddings of the Eleventh International Workshop on Principles of Diagnosis (DX-00)*, pp. 201–212, Morelia, Mexico, (2000).

[39] L. Travé-Massuyès and R. Milne, 'Gas-turbine condition monitoring using qualitative model-based diagnosis', *IEEE Expert*, **12**, 22–31, (1997).

[40] B.C. Williams and B. Millar, 'Automated decomposition of model-based learning problems', in *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, pp. 258–266, Val Morin,

Quebec, Canada, (1996).

A Graph and hyper-graph notation

$H = [V, E]$	Hyper-graph H, made up V: nodes, and E: a family of sub-sets in V
Γ_i	Successors for node i
Γ_i^{-1}	Predecessors for node i
$d_H(i)$	Degree for node i in H
$d_H^+(i),\ d_H^-(i)$	Output and input demi-degree for node i in H

Bipartite graph: $G = [V, E]$ is a bipartite graph if there are two disjoints parts in $V = S \cup T$, and edges in E are always directed from S to T.

Matching: A matching M in $G = [V, E]$ is a subset of E such that no two arcs in M share a common vertex incident to them.

B Algorithms for computing the set of minimal evaluation chains

Algorithm build-every-mec (SMEC) **is**
SMEC: set of MEC; { Each MEC is a set of constraints}
available, to-be-justified, justified, chain: set of constraints;
R, R2: constraint;
Begin
available := Constraints-in(H_{SD});
while available $\neq \emptyset$ *do*
R := Select-constraint(available);
chain := $\emptyset$;
available := available $\setminus$ {R};
build-mec (SMEC, chain, R, available);
end while
End

Algorithm build-mec (SCEM, chain, R, available) **is**
Begin
Insert R in chain;
to-be-justified := R.nobs;
justified := $\emptyset$;
Justify (SMEC, chain, to-be-justified, justified, available);
End

Algorithm Justify (SMEC, chain, to-be-justified, justified, available) **is**
v: unknown variable;
related: set of constraints;
Begin
if to-be-justified = $\emptyset$ *then*
if there is no subset of chain in SMEC *then*
Erase chain supersets from SMEC;
Insert chain in SMEC;
end if { Only minimal chains are included in SMEC.}
else
v := select-variable (to-be-justified);
related := R | R $\in$ available and v $\in$ R.nobs;
while related $\neq \emptyset$ *do*
R1 := select-r (related);
related := related $\setminus$ {R1};
chain2 := chain $\cup$ {$R1$};
Justified2 := Justified $\cup\{v\}$;
to-be-justified2 := (to-be-justified $\setminus$ v) $\cup$ (R1.nobs $\setminus$ justified2 };
available2 := available $\setminus$ R1;
Justify (SMEC, chain2, to-be-justified2, justified2, available2);
end while
end if
End

C Algorithms for computing the set of minimal evaluation models

Algorithm build-every-mem (SMEC, SMEM) **is**
Begin
for chain = each MEC in SMEC *do*
for R = each constraint in chain *do*
for I = each interpretation for R *do*
model := {I};
to-be-justified:= I.nobs;
justified := $\emptyset$;
chain := chain $\setminus$ {R};
build-mem (model, chain, to-be-justified, justified, SMEM);
end for
end for
end for
End

Algorithm build-mem (model, available, to-be-justified, justified, SMEM) **is**
Begin
if to-be-justified = $\emptyset$ and available = $\emptyset$ and $\exists_1$ discrepancy node in model *then*
Insert model in SMEM;
end if
else
for S = each constraint in available *do*
if S.nobs $\cap$ to-be-justified = $\emptyset$ *then*
for I2 = each interpretation for S *do*
if head(I2) $\cap$ to-be-justified $\neq \emptyset$ *then*
Insert {$I2$} in model;
available := available $\setminus$ {S};
to-be-justified := (to-be-justified $\setminus$ head(I2)) $\cup$ tail(I2).nobs;
Insert head(I2) in justified;
Build-mem (model, available, to-be-justified, justified, SMEM);
end if
end for
end if
end for
end if
End

D Constraints used to model the hydraulic system

Constraints	Represent
$tr1_1, t2_1, tr2_4$	Mass balance in T: $m'_T = \sum f_{in} - \sum f_{out}$
$tr1_2, t2_2$	Overflow in T: $f_{out} = \sqrt{k \cdot (h_T - h_{ext})}$
$tr1_3, t2_3, tr2_5$	Mass: $m_T(t) = \int m'_T(t-1)dt + m_T(t-1)$
$tr1_4, t2_5, tr2_6$	Height in T: : $h_T = k_1 \cdot \frac{m_T}{A_T}$
$t2_4, tr2_2$	Pressure at bottom: $P_{T_1} = k_2 \cdot h_T + P_{atm}$
$p2_1, p3_2$	Pump load curve in P: $\Delta P_P = tablePQ(f_{out})$
$p2_2, p3_1$	Outflow in T: $f_{out} = \sqrt{k_3 \cdot \frac{(P_{T_1} + \Delta P_P - P_2)}{k_4}}$
$p2_3, p3_3, v2_1$	Flow out of tank: $f_{in} = f_{out}$
$tr2_1$	Control: $u = PID(h_T)$
$v2_2$	Flow through a valve: $f_{out} = \sqrt{k_5 \cdot \frac{(P_{T_2} - P_{atm})}{k_6 + (\frac{100}{u})^2}}$

Model-Based Reliability and Diagnostic: A Common Framework for Reliability and Diagnostics *

Bernhard Anrig and Jürg Kohlas
Department of Informatics, University of Fribourg,
CH–1700 Fribourg, Switzerland
{Bernhard.Anrig,Juerg.Kohlas}@unifr.ch

Abstract. Technical systems are in general not guaranteed to work correctly. They are more or less reliable. One main problem for technical systems is the computation of the reliability of a system. A second main problem is the problem of diagnostic. In fact, these problems are in some sense dual to each other.

In this paper, we will use the concept of probabilistic argumentation systems PAS for modeling the system description as well as observation and specifications of behaviour in one common framework. We show that PAS are a framework which allows to formulate both main problems easily and all concepts for these two problems can clearly be defined therein. Using PAS, reliability and diagnostic can be considered as dual problems. PAS allows to consider one common strategy for computing answers to the questions in the different situations.

1 Introduction and Overview

One main problem for technical systems is the computation of the reliability of a system. This is studied in reliability theory (see for example [7, 8]). The reliability depends on various factors like the quality and the age of components, complexity of the system, etc. The reliability of a system conveys some information about the behavior of the system in the future, based on information about the components, for example probabilistic information about the reliability over time.

A second main problem for technical systems is the problem of diagnostic. Here, the problem is to explain the behavior of the system, usually based on measurements and observations of some parts of the system, together with the system description in some framework. The actual observations and the description of the system are the only ingredients for the computation of the diagnoses. Additionally, if probabilistic knowledge is available about the different operating modes of the components, then the likelihood of the system states can be defined and prior as well as posterior probabilities can be computed for the set of possible system states.

* Research supported by grant No. 2000-061454.00 of the Swiss National Foundation for Research.

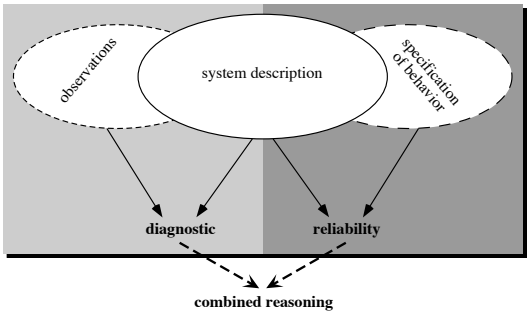


Figure 1. Reliability versus Diagnostic.

The two main problems depend both on a formalization of the system in some framework together with either observations, measurements, or requirements (Fig. 1). Here, we will use the concept of probabilistic argumentation systems PAS for modeling the system description as well as observation and specifications of behaviour in one common framework. The goal of a PAS is to derive arguments in favor and against the hypothesis of interest. An argument is a defeasible proof built on uncertain assumptions, i.e. a chain of deductions based on assumptions that makes the hypothesis true. If probabilistic information is available, a quantitative judgement of the situation is obtained by considering the probabilities that the arguments are valid. The resulting degrees of support and possibility correspond to belief and plausibility, respectively, in the Dempster-Shafer theory of evidence [24, 20]. In fact, PAS combines the strengths of logic and probability in one framework. In this paper we show that probabilistic argumentation systems are a framework which allows to formulate both main problems, i.e. reliability and diagnostic, easily and all concepts therefore can clearly be defined therein. The framework will especially allow to consider one common strategy for computing answers to the questions in the different situations. Some work in this direction but without using PAS has been done by Provan [22].

The main information for both problems is the description of the system in some formalism; we will focus here on a for-

malization using logic. In the case of reliability, we may have a specification which describes the goals which have to be fulfilled by the system. This information will be used to compute the structure function from the system description. Different specifications may lead to different structure functions. Even in the absence of an explicit specification of a reliability requirement, we may deduce a structure function by assuming that the system should be functioning at least if all components are working.

On the other hand, in the case of diagnostic, some observations of the system may indicate that the system is not working as it is supposed to be. This information — together with the system description — allows then to compute the diagnoses of the system, i.e. minimal sets of components whose malfunctioning "explains" the wrong behaviour of the whole system.

2 Reliability

2.1 Combinatorial Reliability

In binary combinatorial reliability, a system is assumed to be composed of a number of different components. Each component is either intact or it is down, and so is the whole system itself, depending on the states of its components. In order to formulate this, binary variables x_i are associated to components $i = 1, 2, \ldots, n$ of the system, where $x_i = \top$ if the component number i works and $x_i = \bot$ otherwise. Let $\mathbf{x}$ be the vector $(x_1, x_2, \ldots, x_n)$ of the component states. This state-vector has 2^n possible values. These values can be decomposed into two disjoint subsets, the set $S_\top$ of working states, for which the system as a whole is assumed to be functioning, and the set $S_\bot$ of down-states, for which the system is supposed to not work properly. The corresponding system state is denoted by x. Its dependence on the state-vector $\mathbf{x}$ is described by a Boolean function ϕ, defined as

$$x \quad = \quad \phi(\mathbf{x}) \quad = \quad \begin{cases} \top & \text{if } \mathbf{x} \in S_\top, \\ \bot & \text{if } \mathbf{x} \in S_\bot. \end{cases} \tag{1}$$

The Boolean function ϕ is called the *structure function* of the system. In combinatorial reliability it is assumed to be given and it forms the base for reliability analysis.

The structure function ϕ is usually assumed to be *monotone*. That is, if $\mathbf{x}_1 \leq \mathbf{x}_2$, then $\phi(\mathbf{x}_1) \leq \phi(\mathbf{x}_2)$. For a monotone structure function, a subset $P \subseteq \{1, 2, \ldots, n\}$ of components is called a *path*, if $\phi(\mathbf{x}) = \top$ for all state-vectors $\mathbf{x}$ for which the components of the set P are working, $x_i = \top$ for all $i \in P$. That is, the elements of a path are sufficient to guarantee the functioning of the system, regardless of the state of the components outside the path. We assume that the set $\{1, 2, \ldots, n\}$ of all components is a path (otherwise the system would never be functioning). A path P is called *minimal*, if no proper subset of P is still a path. Since the paths are upwards closed it is sufficient to know all minimal paths. Let $\mathcal{P}$ denote the set of minimal paths. This set determines the structure function,

$$\phi(\mathbf{x}) = \bigvee_{P \in \mathcal{P}} \bigwedge_{i \in P} x_i. \tag{2}$$

This logical formula expresses the fact, the system is working, if all components of at least one minimal path are working.

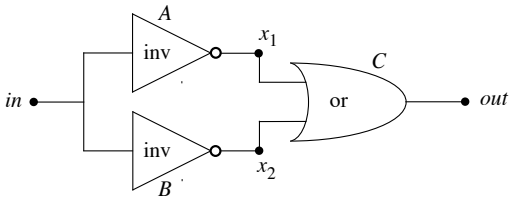


Figure 2. A simple device

Dually, the notion of a *cut* is defined and $\mathcal{C}$ denotes the set of all minimal cuts.

If for every component $i = 1, 2, \ldots, n$ its respective probability p_i of functioning correctly is defined, then the probability that the system is functioning can be computed (assuming the components to be stochastically independent). In fact, $\phi(\mathbf{x})$ is a random variable, and the probability p that the system is functioning is

$$p = E(\phi(\mathbf{x})) = h(\mathbf{p}). \tag{3}$$

Here, $\mathbf{p}$ denotes the vector $(p_1, p_2, \ldots, p_n)$ of probabilities. $h(\mathbf{p})$ is called the reliability function and its computation is a nontrivial task [1, 16, 5].

2.2 Model-Based Reliability

The structure function describes the conditions under which a system is functioning, depending on the states of its components. It is already a compilation of knowledge about the system and its structure. In this section we shall illustrate another approach, where a more physical description of a system is given. Additionally, a specification of the desired behavior of the system is given. These two elements will then allow the deduction of a structure function and its associated reliability function. The discussion in this section will be informal.

Example 1: Detector of Power Failure
(Example adapted from [22])
Consider a simple device which watches a Boolean value *in* and reports an output *out* equal to $\top$, if the value vanishes (becomes $\bot$). A simple version of such a device is depicted in Figure 2. The functionality of this device can be described with propositional logic. Let *in* and *out* be the variables which denote the state of the in- and output respectively. Both variables are binary, i.e. represent the boolean values true or false respectively. Further, there are two internal variables x_1 and x_2, also binary. For every component A, B or C, there is a respective binary variable ok_A, ok_B, and ok_C which describes the working mode of the component.

Consider the inverter A: if it works correctly (ok_A is true), then its input is the negation of its output, *out* is true if and only *in* is false. We express this by the formula $in \leftrightarrow \neg x_1$. So the entire information is modeled as the logical implication $ok_A \rightarrow (in \leftrightarrow \neg x_1)$. Note that so far nothing is said about the behavior of the component, if it is down (ok_A is false). There are several possibilities. One is that in this case the output of the component is always false, i.e. $\neg ok_A \rightarrow \neg x_1$.

For the component B, the same specification can be applied. For the or-gate, if it works correctly, then the output is

true if at least one of its inputs is. So the whole information about the device is modeled by a set of six implications:

$$\Sigma = \left\{ \begin{array}{ll} ok_A \rightarrow (in \leftrightarrow \neg x_1), & \neg ok_A \rightarrow \neg x_1, \\ ok_B \rightarrow (in \leftrightarrow \neg x_2), & \neg ok_B \rightarrow \neg x_2, \\ ok_C \rightarrow (out \leftrightarrow x_1 \vee x_2), & \neg ok_C \rightarrow \neg out \end{array} \right\} \quad (4)$$

This is the *system description*. We add now a specification of what we expect from the system to this physical description of the system. We expect, that negative (false) input is detected, i.e. the output is true. This could be expressed by $\neg in \rightarrow out$. However, this is a weak requirement. It does not exclude that *out* becomes true, even if *in* is true. More stringent would be the specification $\neg in \leftrightarrow out$. This asks that there is an alarm (*out*) if, and only if, *in* is false.

We may now ask under which states, described by the variables ok_A, ok_B, and ok_C, each one of these specifications is fulfilled. This defines the *structure function* of the system associated with the corresponding specification of desired system behavior. We shall see in the next section, that it is a well-defined problem of propositional logic to deduce these structure functions from the system description and the specifications of desired behavior. $\ominus$

This example shows how the physical behavior of systems and the required behavior can be described in the language of propositional logic. We shall examine this structure in the following section in a general context.

3 Probabilistic Argumentation Systems

Probabilistic argumentation systems have been developed as general formalisms for expressing uncertain and partial knowledge and information in artificial intelligence. They combine in an original way logic and probability. Logic is used to derive arguments and probability serves to compute the reliability or likelihood of these arguments. These systems can be used for model-based diagnostics as has been demonstrated in [2, 19]. Here we shall show how they relate to reliability theory.

In this section we give a short introduction into propositional probabilistic argumentation systems. For a more detailed presentation of the subject we refer to [15]. We remark also that such systems have been implemented in a system called ABEL which is available on the internet (cf. [14] for further information).

3.1 Propositional Logic

Propositional logic deals with declarative statements, called called *propositions*, that can be either true or false. Let $P = \{p_1, \ldots, p_n\}$ be a finite set of propositions. The symbols $p_i \in P$ together with $\top$ (tautology) and $\bot$ (falsity), are called *atoms*. Compound formulas are built by the following syntactic rules:

- atoms;
- if γ is a formula, then $\neg\gamma$ is a formula;
- if γ and δ are formulas, then $(\gamma \wedge \delta)$, $(\gamma \vee \delta)$, $(\gamma \rightarrow \delta)$, and $(\gamma \leftrightarrow \delta)$ are formulas.

By assigning priority in decreasing ordering to $\neg$, $\wedge$, $\vee$, $\rightarrow$, some parentheses can be eliminated. The set $\mathcal{L}_P$ of all formulas generated by the above recursive rules is called *propositional language* over P.

A literal is either an atom p_i or the negation of an atom $\neg p_i$. A *term* is either $\top$ or a conjunction of literals where every atom occurs at most once (but none of $\bot$ and $\top$), and a *clause* is either $\bot$ or a disjunction of literals where every atom occurs at most once (but none of $\bot$ and $\top$). $C_P \subseteq \mathcal{L}_P$ denotes the set of all terms, and D_P the set of all clauses.

$N_P = \{0,1\}^n$ denotes the set of all 2^n different interpretations for P. If $\gamma \in \mathcal{L}_P$ evaluates to 1 under $\mathbf{x} \in N_P$, then $\mathbf{x}$ is called a *model* of γ. The set of all models of γ is denoted by $N_P(\gamma) \subseteq N_P$.

A propositional sentence γ *entails* another sentence δ (denoted by $\gamma \models \delta$) if and only if $N_P(\gamma) \subseteq N_P(\delta)$. Sometimes, it is convenient to write $\mathbf{x} \models \gamma$ instead of $\mathbf{x} \in N_P(\gamma)$. Also we write $\gamma \models \bot$ if γ is not satisfiable. Furthermore, two sentences γ and δ are *logically equivalent* (denoted by $\gamma \equiv \delta$), if and only if $N_P(\gamma) = N_P(\delta)$.

3.2 Basic Concepts of Argumentation Systems

Consider two finite sets $P = \{p_1, \ldots, p_m\}$ and $A = \{a_1, \ldots, a_n\}$ of propositional variables with $A \cap P = \emptyset$, the elements of P are called propositions, the elements of A assumptions. We consider a fixed set of formulas $\Sigma \subseteq \mathcal{L}_{A \cup P}$ called the *knowledge base*, which models the information available; sets of formulas are interpreted conjunctively, i.e. $\Sigma = \bigwedge\{\xi \in \Sigma\}$. We assume that this knowledge base is satisfiable. A triple (Σ, A, P) is called a *propositional argumentation system PAS*.

The elements of N_A are called *scenarios* (or *system states*). A scenario represents a specification of all values of the assumptions in A. Define now:

Inconsistent Scenarios: $CS_A(\Sigma) := \{\mathbf{s} \in N_A : \mathbf{s}, \Sigma \models \bot\}$,

Quasi-Supporting Scenarios of $h \in \mathcal{L}_N$**:**
$$QS_A(h, \Sigma) := \{\mathbf{s} \in N_A : \mathbf{s}, \Sigma \models h\},$$

Supporting Scenarios of $h \in \mathcal{L}_N$**:**
$$SP_A(h, \Sigma) := QS_A(h, \Sigma) - CS_A(\Sigma),$$

Possible Scenarios for $h \in \mathcal{L}_N$**:**
$$PL_A(h, \Sigma) := SP_A{}^c(\neg h, \Sigma).$$

Inconsistent scenarios are in contradiction with the knowledge base and therefore to be considered as excluded by the knowledge. Supporting scenarios for a formula h are scenarios, which, together with the knowledge base imply h and are consistent with the knowledge. So, under supporting scenarios, the hypothesis h is true. Possible scenarios for h are scenarios, which do not imply $\neg h$ and thereby do not refute h. Quasi-supporting scenarios for h are the union of supporting scenarios and inconsistent scenarios.

Scenarios are the basic concepts of assumption-based reasoning. However, sets of inconsistent, quasi-supporting, supporting and possible scenarios may become very large. Therefore, more economical, logical representations of these sets are needed. For this purpose, the following concepts are defined:

Set of Supporting Argument for h**:**
$$SP(h, \Sigma) = \{\alpha \in C_A : N_A(\alpha) \subseteq SP_A(h, \Sigma)\},$$

The sets of quasi-supporting and of possible arguments are defined analogously. Remark that supporting arguments are similar to paths for structure functions in reliability theory. This similarity will be exploited later. These sets are

all upward closed. Hence the sets of arguments are already determined by their *minimal* elements. We denote by $\mu QS(h,\Sigma)$, $\mu SP(h,\Sigma)$ and $\mu PL(h,\Sigma)$ the sets of minimal quasi-supporting, supporting and possible arguments. Further,

Conflict: $conf(\Sigma) := \bigvee_{\alpha\in\mu QS(\bot,\Sigma)} \alpha,$

Support of h**:** $sp(h,\Sigma) := \bigvee_{\alpha\in\mu SP(h,\Sigma)} \alpha,$

Quasi-support $qs(h,\Sigma)$ and possibility $pl(h,\Sigma)$ are defined analogously. Clearly, any formula which is logically equivalent to logical representations above can be used as a representation.

Example 2: (Cont. of Example 1)
The information of Example 1 is modeled in an argumentation system as follows: $A = \{ok_A, ok_B, ok_C\}$, $P = \{in, x_1, x_2, out\}$ and Σ as in (4). There are no inconsistent scenarios and for $h = \neg in \rightarrow out$ we have $QS_A(h,\Sigma) = \{(0,1,1),(1,0,1),(1,1,1)\}$ and $PL_A(h,\Sigma) = N_A$. As $CS_A(\Sigma) = \emptyset$, we have $QS_A = SP_A$ in this situation and there are some arguments in favor of the hypothesis, but none against it. Hence, $qs(h,\Sigma) = (ok_A \wedge ok_C) \vee (ok_B \wedge ok_C)$ and $pl(h,\Sigma) = \top$. $\ominus$

3.3 Probabilistic Information

On top of the structure of a propositional argumentation systems, we may easily add a probability structure. Assume that there is a probability $p(a_i) = p_i$ for every assumption $a_i \in A$ given. Assuming stochastic independence between assumptions, a scenario $\mathbf{s} = (s_1, \ldots, s_n)$ gets the probability

$$p(\mathbf{s}) \quad = \quad \prod_{i=1}^{n} p_i^{s_i}(1-p_i)^{1-s_i}. \tag{5}$$

This induces a probability measure p on $\mathcal{L}_A$,

$$p(f) = \sum_{\mathbf{s}\in N_A(f)} p(\mathbf{s})$$

for $f \in \mathcal{L}_A$. A quadruple (Σ, A, P, Π) with $\Pi = (p_1, \ldots, p_n)$ is then called a *probabilistic (propositional) argumentation system* PAS.

The problem of computing the probability $p(f)$ is similar to the problem of computing the reliability of a structure function, except, that monotonicity cannot be assumed in general; for algorithms for efficiently computing the probability $p(f)$ see [20, 9, 13].

Once we have such a probability structure on top of a propositional argumentation system, we can exploit it to compute likelihoods (or in fact, reliabilities) of supporting and possible arguments for hypothese h. First, we note, that Σ imposes that we eliminate the inconsistent scenarios and condition the probability on the consistent ones. In other words, Σ is an event that restricts the possible scenarios to the set $N_A - CS_A(\Sigma)$, hence their probability has to be conditioned on the event Σ. This conditional probability is defined by

$$p'(\mathbf{s}) = \frac{p(\mathbf{s})}{1 - p(qs(\bot,\Sigma))}.$$

for consistent scenarios $\mathbf{s}$. $p(qs(h,\Sigma)) = dqs(h)$ is the so-called degree of quasi-support for h. Now, the degree of support dsp for hypotheses h is defined by

$$dsp(h) = p'(sp(h,\Sigma)) = \frac{dqs(h,\Sigma) - dqs(\bot,\Sigma)}{1 - dqs(\bot,\Sigma)}.$$

This result explains the importance of quasi-support. It is sufficient to compute degrees of quasi-supports. Further, we obtain the degree of plausibility of h,

$$dpl(h) = p'(pl(h,\Sigma)) = \frac{1 - dqs(\neg h,\Sigma)}{1 - dqs(\bot,\Sigma)} = 1 - dsp(\neg h).$$

Degree of quasi-support $dqs(h)$ and of support $dsp(h)$ correspond in fact to unnormalized and normalized belief in the Dempster-Shafer theory of evidence [24, 20, 15].

3.4 Computational Theory

Computing quasi-supports is the basic operation in PAS. It can be based on resolution and variable elimination (forgetting) [15, 12, 13]. In the sequel, we will sketch some of the main concepts for computation.

First, note that the computation of $qs(h)$ can be reduced to the computation of the conflicts with respect to an updated knowledge base: $qs(h,\Sigma) = qs(\bot, \Sigma \cup \{\neg h\})$. So for any hypothesis h, the quasi-supporting arguments $qs(h,\Sigma)$ can be determined by computing the conflicts with respect to the knowledge base $\Sigma \cup \{\neg h\}$. Hence in the sequel, we focus on the computation of the conflicts with respect to a general knowledge base.

The ideas presented in the sequel are based on representations of knowledge in conjunctive normal form (CNF), i.e. a conjunction of clauses. The main step is based on the principle of resolution. Let $x \in A \cup P$. A disjoint decomposition of Σ is then defined as follows:

$$\begin{aligned}
\Sigma_x^+ &= \{\xi \in \Sigma : x \in Lit(\xi)\} \\
\Sigma_x^- &= \{\xi \in \Sigma : \neg x \in Lit(\xi)\} \\
\Sigma_x^0 &= \{\xi \in \Sigma : x \notin Lit(\xi) \text{ and } \neg x \notin Lit(\xi)\}
\end{aligned}$$

$Lit(\Sigma)$ denotes the set of all literals occurring in Σ. A literal is either a (positive) atom or a negated atom.

Consider two clauses $\xi^+ = x \vee \delta^+$ and $\xi^- = \neg x \vee \delta^-$ in Σ_x^+ and Σ_x^- respectively. The clause $\rho(\xi^+, \xi^-) = \delta^+ \vee \delta^-$ is called the resolvent; note that we simplify implicitly the resolvent so that $\rho(\xi^+, \xi^-)$ is again a clause, i.e. double occurrences of atoms etc. are simplified.

Eliminating a variable $x \in P \cup A$ from Σ means now to compute

$$Elim_x(\Sigma) = \mu(\Sigma_x^0 \cup \{\rho(\xi^+, \xi^-) : \xi^+ \in \Sigma_x^+,\ \xi^- \in \Sigma_x^-\})$$

Consider a set $Q \subseteq P \cup A$. We define now, for $Q = \{q_1, \ldots, q_r\}$,

$$Elim_Q(\Sigma) = Elim_{q_r}(\ldots(Elim_{q_2}(Elim_{q_1}(\Sigma)))\ldots)$$

The result does not depend on the very order of the elimination of atoms; yet note that the computations depend *critically* on a "good" ordering, see [15] for a discussion as well as relations to the theory of local computation (in the sense of Shenoy & Shafer [25]).

This allows then to compute the quasi-supporting arguments of a knowledge base Σ as follows:

Theorem 1 ([15])

$$QS_A(h,\Sigma) = N_A^c(Elim_P(\Sigma \cup \{\neg h\}))$$

In other words, this theorem asserts that

$$qs(h,\Sigma) \equiv \neg \bigwedge Elim_P(\Sigma \cup \{\neg h\}).$$

The concept of elimination allows to compute quasi-supporting and therefore also supporting as well as possible arguments for hypotheses. This notation connects the concepts presented here to the more general theory of valuation algebras, a general theory for representing, combining and focusing pieces of information [18, 21].

4 Reliability Analysis Using Probabilistic Argumentation Systems

4.1 Reliability based on Requirement Specification

We discuss now how probabilistic argumentation systems can be used to formulate and solve reliability problem. The basic idea is simple: The system behavior is described in terms of the states of its components. In addition the desired or required behavior of the system is specified. The system description forms a probabilistic argumentation system. The question is then: how likely (probable) is it, that the specified requirement is satisfied? In order to answer this question, the specification of required behavior is taken as a hypothesis. The *support* of this specification determines then essentially the structure function of this reliability problem, and the degree of support of the specified requirement is the reliability of the system with respect to the required behavior. Note that — depending on different goals a system should attain, or services it should provide — different requirements may be formulated. So the corresponding reliability analysis has to be differentiated, but can be carried out within the same framework of probabilistic argumentation systems.

Example 3: (Cont. of Example 1)
We have already formulated Σ and two different specifications $\delta_1 = \neg in \rightarrow out$ and $\delta_2 = \neg in \leftrightarrow out$. We can compute the supports of these two specifications. It turns out, that both are the same,

$$sp(\delta_1,\Sigma) = sp(\delta_2,\Sigma) = (ok_A \wedge ok_C) \vee (ok_B \wedge ok_C).$$

Note that this is just the path representation of the expected structure function. In fact this structure function could be reformulated as $(ok_A \vee ok_B) \wedge ok_C$, which shows that it is a series system composed of component C and a parallel module of the components A and B. The remarkable fact is, that this structure function has been automatically deduced from the system description and the specification of requirements.

The system description is an essential element for this analysis. If it is changed, then this may influence the results of the analysis. Suppose that, in contrast to the model above, we do not know how the faulty components behave. The knowledge base becomes now

$$\Sigma' = \left\{ \begin{array}{l} ok_A \rightarrow (in \leftrightarrow \neg x_1),\ ok_B \rightarrow (in \leftrightarrow \neg x_2), \\ ok_C \rightarrow (out \leftrightarrow x_1 \vee x_2). \end{array} \right\}$$

With this less complete model, the structure function of the two specifications above become different,

$$\begin{array}{rcl} sp(\delta_1,\Sigma') & = & (ok_A \wedge ok_C) \vee (ok_B \wedge ok_C), \\ sp(\delta_2,\Sigma') & = & ok_A \wedge ok_B \wedge ok_C. \end{array}$$

Now, the stronger requirement δ_2 can only be guaranteed if all three components work correctly (a series system), whereas the weaker one still has the same redundancy as before. $\ominus$

In the general case, we have a PAS (Σ, A, P), where the assumable symbols in A correspond to the components of the system. Positive assumptions correspond to working components. Accordingly in the context of reliability analysis, we shall call the scenarios of this argumentation system *system states*. The propositional symbols in P are needed to describe the system behavior. We assume that the system description Σ excludes no system states, that is there are no conflicts, $QS_A(\bot,\Sigma) = \emptyset$. A knowledge base Σ which satisfies this is called A-consistent.

The required behavior is specified by a formula δ. Usually δ will not contain assumptions, but there is no reason to exclude this in general. δ formulates a reliability goal. There may be several such goals.

The set of system states $SP_A(\delta,\Sigma)$ supporting δ contains all states guaranteeing the required specification from the system description. Its complement $SP_A{}^c(\delta,\Sigma) = PL_A(\neg\delta,\Sigma)$ contains the system states where this guarantee is no more assured. These are the unreliable states. So $SP_A(\delta,\Sigma)$ defines the *structure function* associated with the specification δ

$$s \;=\; \phi_{\delta,\Sigma}(\mathbf{s}) \;=\; \left\{ \begin{array}{ll} \top & \text{if } \mathbf{s} \in SP_A(\delta,\Sigma), \\ \bot & \text{if } \mathbf{s} \notin SP_A(\delta,\Sigma). \end{array} \right. \tag{6}$$

The index Σ in $\phi_{\delta,\Sigma}$ will be omitted if Σ is clear from the context. Here, s denotes the "system state", which is $\top$, when the reliability specification is assured and $\bot$ otherwise. Since the set $SP_A(\delta,\Sigma)$ has a logical representation based on minimal arguments, the same holds for the structure function ϕ_δ,

$$\phi_\delta = \bigvee_{\alpha \in \mu SP(\delta,\Sigma)} \alpha \quad = sp(\delta,\Sigma) \tag{7}$$

In the same way, based on minimal possible arguments $PL(\neg\delta,\Sigma)$, we obtain

$$\neg\phi_\delta = \bigvee_{\beta \in \mu PL(\neg\delta,\Sigma)} \beta \quad = pl(\neg\delta,\Sigma).$$

By de Morgan laws this transforms into

$$\phi_\delta = \bigwedge_{\beta \in \mu PL(\neg\delta,\Sigma)} \neg\beta. \tag{8}$$

Note that $\neg\beta$, the negation of a term, is a clause. This is a second logical representation of ϕ_δ.

A comparison with the minimal path and minimal cut representation of monotone structure functions (2) shows that minimal supporting arguments α for δ and minimal possible arguments β for $\neg\delta$ play a role similar to minimal paths and minimal cuts.

According to our assumption of A-consistency, we have $QS_A(\bot,\Sigma) = \emptyset$. Thus

$$SP_A(\delta,\Sigma) = QS_A(\bot, \Sigma \cup \{\neg\delta\}). \tag{9}$$

On the other hand, we have also

$$PL_A(\neg\delta, \Sigma) = QS_A{}^c(\bot, \Sigma \cup \{\neg\delta\}). \quad (10)$$

This shows, that a reliability analysis of a system Σ relative to a requirement specification δ, requires essentially the computation of the conflict states $QS_A(\bot, \Sigma \cup \{\neg\delta\})$. We shall see below, that this is exactly also what is required for diagnosis. This is a first hint to the duality between the problems of reliability and diagnosis.

Once probabilities for the assumptions, i.e. component availabilities or reliabilities are defined, system reliability relative to a specification δ is simply the degree of support of δ, (since $QS_A(\bot, \Sigma) = \emptyset$), i.e.

$$p_{\delta,\Sigma} = dsp(\delta, \Sigma) = dqs(\delta, \Sigma) = p(QS_A(\bot, \Sigma \cup \{\neg\delta\})).$$

4.2 Implicitly Defined Reliability

A specification δ is called *consistent* with the system description Σ, if the system state $\mathbf{1}$ belongs to $SP_A(\delta, \Sigma)$. In this section we only consider specifications consistent with the system description.

A system description Σ often contains, besides assumptions, another set O of special propositional atoms, namely those which are *observable*. Then specifications δ can be assumed to be formulated with observables only, $\delta \in \mathcal{L}_O$. Observables are typically input and output variables of some system.

Assume now, that in a system description (Σ, P, A) a set of observable variables O is singled out. Usually, $O \subseteq P$, i.e. component states can not be observed directly. But it does no harm to assume more generally $O \subseteq P \cup A$. Then we define an implicit specification

$$\hat{\delta} = Elim_{(A\cup P)-O}(\Sigma \cup \{a_1 \wedge a_2 \wedge \cdots \wedge a_n\}).$$

That is, $\hat{\delta}$ represents all the functionality of the system in terms of observables which can be obtained from a system with all components working. We call this the *implicit* reliability specification with respect to O. Now, the system may be — with respect to this specification — as good as "new" also for some states including faulty components. Therefore we define the implicit structure function by the set of up-states relative to $\hat{\delta}$, i.e. $SP_A(\hat{\delta}, \Sigma)$. Hence, we obtain

$$\phi_{\hat{\delta}} = \bigvee_{\alpha\in\mu SP(\hat{\delta},\Sigma)} \alpha, \quad \text{or} \quad \phi_{\hat{\delta}} = \bigwedge_{\beta\in\mu PL(\neg\hat{\delta},\Sigma)} \neg\beta.$$

Accordingly, the implicit reliability of such a system can be obtained as the degree of support $dsp(\hat{\delta}, \Sigma)$. This approach helps to decide whether a system has some implicit redundancy, namely, whether $\phi_{\hat{\delta}}$ represents simply a series system, i.e. $\mu SP(\hat{\delta}, \Sigma)$ has only the set of all assumptions as minimal supporting argument for $\hat{\delta}$.

Lemma 2 *If $\delta \in \mathcal{L}_O$ is a consistent specification with respect to Σ, then $\hat{\delta} \models \delta$.*[1]

This shows that $\hat{\delta}$ is the most stringent, consistent specification over observables O. For all specifications over O the implicit specification has least reliability:

[1] For proofs see [6].

Lemma 3 *If $\delta \in \mathcal{L}_O$ is a consistent specification with respect to Σ, then $SP_A(\hat{\delta}, \Sigma) \subseteq SP_A(\delta, \Sigma)$.*

Corollary 4 *If $\delta \in \mathcal{L}_O$ is a consistent specification with respect to Σ, then $p_{\hat{\delta}} \leq p_\delta$.*

5 Model-Based Diagnostic

5.1 Duality Between Reliability and Diagnostics

A problem of diagnostics arises if an observation indicates that a requirement specification δ is violated. Then the question is: how can the required functionality be recovered? That is, one would like to find out those components whose failure caused the system failure and which have to be fixed or replaced. This analysis will be based on the system description Σ and on the specification δ which is violated.

In fact, we ask, which system states are compatible or consistent with the system description Σ and the violation of the specification δ, expressed by $\neg\delta$. Well, these are of course all states which are consistent with $\Sigma \cup \{\neg\delta\}$, that is the set

$$QS_A{}^c(\bot, \Sigma \cup \{\neg\delta\}) = PL_A(\neg\delta, \Sigma). \quad (11)$$

Remark that this is exactly the set of down states relative to the specification δ (see (10)). Here we have the basic *duality* between *reliability analysis* relative to a requirement specification δ and the *diagnostic problem* relative to the same specification. The conflict set $QS_A(\bot, \Sigma \cup \{\neg\delta\})$ is the computational key to both reliability analysis and diagnostics. It gives the up-states which define reliability and its complement gives the possible states explaining the violation of the reliability specification, i.e. possible diagnostics. It is well known in model-based diagnostics that such conflict sets play a key role [23, 10, 19]. The duality implies that they play an equally important role for model-based reliability.

If the structure function $\phi_{\delta,\Sigma}$ is *monotone*, then to the minimal possible arguments $\beta \in \mu PL(\neg\delta, \Sigma)$ correspond the minimal cuts $\neg\beta$. They represent minimal sets of failed components, which explain the violation of the specification δ, independently on the state of the other components.

Minimal cuts correspond to kernel diagnoses in model-based diagnostics [23]. Usually model-based diagnostics goes not beyond such concepts of diagnostics. It neglects the important role of probability.[2] The observation of the violation of the specification $\neg\delta$ in fact defines the event $QS_A{}^c(\bot, \Sigma \cup \{\neg\delta\})$ in the sample space N_A. That is, the prior probabilities $p(\mathbf{s})$ defined on the states have now to be conditioned on this event. This gives us the *posterior probabilities*

$$p(\mathbf{s}|\neg\delta) = \frac{p(\mathbf{s})}{1 - p(QS_A(\bot, \Sigma \cup \{\neg\delta\}))} = \frac{p(\mathbf{s})}{dpl(\neg\delta, \Sigma)}, \quad (12)$$

for diagnostic states $\mathbf{s} \in QS_A(\bot, \Sigma \cup \{\neg\delta\})$. This underlines once more the key role of the conflict set $QS_A(\bot, \Sigma \cup \{\neg\delta\})$. Its prior probability is sufficient to compute the posterior probabilities of the possible diagnostic states explaining the violation of δ.

[2] See however [19, 3] for a discussion of this subject, and especially [19] for the problems of the approach of De Kleer & Williams [11]. Other approches focus for example on minimal entropy [26] or on restricting the device to have a Bayesian network model [17].

These posterior probabilities represent important additional diagnostic information. For example we may look for diagnostic states with maximal posterior probability. $\tilde{\mathbf{s}}$ is called a *maximal likelihood state*, if

$$p(\tilde{\mathbf{s}}|\neg\delta) = \max_{\mathbf{s}\in {QS_A}^c(\bot,\Sigma\cup\{\neg\delta\})} p(\mathbf{s}|\neg\delta). \quad (13)$$

There may be several such states. They represent most likely states explaining the violation of δ.

Reiter [23] proposed to look especially at possible diagnostic states with a minimal number of faulty components. Intuitively this makes sense: The failure should be explained with a minimal number of down components. If $\mathbf{s}$ is a state, we define s^- to be the set of its negative (down) components. Then we define a partial order between states: $\mathbf{s}' \leq \mathbf{s}$ if $s'^- \subseteq s^-$. *Reiter diagnoses* are now those diagnostic states $\mathbf{s} \in {QS_A}^c(\bot, \Sigma \cup \{\neg\delta\})$, which are minimal with respect to this partial order. We make the reasonable assumption that for every component i we have $p_i > 0.5$ such that $p_i > 1 - p_i$. I.e. it is more probable that a component works than that it is down. Then $\mathbf{s}' < \mathbf{s}$ implies that $p(\mathbf{s}'|\neg\delta) > p(\mathbf{s}|\neg\delta)$. So maximum likelihood states are Reiter diagnoses. The inverse of course does not hold necessarily. Also, if the structure function ϕ_δ is monotone, the s^- of Reiter diagnoses correspond to minimal cuts relative to the specification δ.

The posterior fault probabilities of the components, $p(\neg a_i|\neg\delta)$, are also of interest. The larger this probability, the more critical is component i for the requirement specification δ. So this is a possible *importance measure* for component i relative to the specification (for other importance measures see [4]).

Example 4: (Cont. of Example 1)
Suppose we observe that, although $\neg in$, we have also $\neg out$, i.e. a power system failure is not detected. Note that $\neg in \wedge \neg out \equiv \neg\delta_1$ (cf. Example 3). So we consult the minimal cuts relative to the specification $\neg\delta_1$. There are two minimal cuts: $\{\neg ok_C\}$ and $\{\neg ok_A, \neg ok_B\}$. To any minimal cut corresponds a Reiter diagnosis, namely, $\{ok_A, ok_B, \neg ok_C\}$ to the first cut, and $\{\neg ok_A, \neg ok_B, ok_C\}$ to the second one. One of these two diagnoses must be the maximum likelihood state. The first one has prior probability $0.99 \times 0.99 \times 0.05 = 0.049$, the second one $0.01 \times 0.01 \times 0.95 = 0.000095$. So clearly, the first one is by far the most likely state. The posterior probability is obtained by dividing the prior probability by the unreliability 0.05 relative to δ_1. We obtain for the maximum likelihood state a posterior probability of 0.98. $\ominus$

5.2 Diagnostics Based on Observations of System Behavior

The actual observation is not necessarily the negation of a system requirement, but may be something stronger, which implies the violation of a specification. Indeed, as we saw in Example 4 we observed $\neg in \wedge \neg out \equiv \neg\delta_1$, but $\neg in \wedge \neg out \models \neg\delta_2$. So, we should reconsider the duality between reliability and diagnostics. In fact, assume that we make some observation of the system behavior, expressed in a formula ω over observables. Then we may test whether $\omega \models \neg\hat{\delta}_\Sigma$. If this is the case, then we have a diagnostic problem, in the sense that at least one component must be down.

The solution of this diagnostic problem is found along similar lines as in the previous section. Possible states are those, which are consistent with the system description and the observation. Or, in other words, the states in the conflict set $QS_A(\bot, \Sigma \cup \{\omega\})$ are those which are excluded by the observation. So, the possible diagnostic states are those of the set $PL_A(\omega, \Sigma) = {QS_A}^c(\bot, \Sigma \cup \{\omega\})$. We see that this diagnostic problem is dual to a (fictitious) reliability problem with a "requirement" specification $\neg\omega$. Note that the specification $\neg\omega$ is always consistent with Σ, since $\hat{\delta}_\Sigma$ is consistent and $\omega \models \neg\hat{\delta}_\Sigma$.

Of course, we get a much sharper diagnostic with an observation $\omega \models \neg\hat{\delta}$, than with the information of $\neg\hat{\delta}$ only. This follows, because according to Lemma 3, we have $PL_A(\omega, \Sigma) \subseteq PL_A(\hat{\delta}, \Sigma)$. So, the more precise the observation, the more states are eliminated. A mere statement that a given reliability specification is violated is less informative than a precise observation implying a violation of a requirement specification.

6 Combining Diagnostic and Reliability

We conclude this discussion of duality between reliability and diagnostics by remarking that we may have an observation of the system behavior which does neither entail a specification δ nor its violation $\neg\delta$. But still this observation is information and we can use it to improve reliability analysis and also to perform a preventive diagnostic analysis (see [6]). For reliability as well as for diagnostic, additional measurements — or more generally any additional information — can be taken into account in the framework presented above and helps to focus the reasoning.

7 Conclusions

In this paper we have shown how closely reliability and model-based diagnostic are connected. The framework of probabilistic argumentation system appears to be a framework which covers both approaches. Therefore the generic structure of PAS can be used for solving problems in both domains. The approaches can even be combined and the information specified can be used in the common framework. Further, from the system description of an argumentation system, we can derive the appropriate structure function and — if desirable — take into consideration a reliability requirement. PAS allows to use local computation architectures and approximation techniques [25, 15]. This complements the computational theory of reliability theory.

REFERENCES

[1] J. A. Abraham, 'An improved algorithm for network reliability', *IEEE Transactions on Reliability*, **28**, 58–61, (1979).

[2] B. Anrig, 'Probabilistic argumentation systems and model-based diagnostics', in *DX'00, Eleventh Intl. Workshop on Principles of Diagnosis, Morelia, Mexico*, eds., A. Darwiche and G. M. Provan, pp. 1–8, (2000).

[3] B. Anrig, *Probabilistic Model-Based Diagnostics*, Ph.D. dissertation, University of Fribourg, Institute of Informatics, 2000.

[4] B. Anrig, 'Importance measures from reliability theory for probabilistic assumption-based reasoning', in *European Conf. ECSQARU'01, Toulouse*, eds., S. Benferhat and P. Besnard, pp. 692–703. Lecture Notes in Artif. Intell., Springer, (2001).

[5] B. Anrig and F. Beichelt, 'Disjoint sum forms in reliability theory', *ORiON J. OR Society South Africa*, **16**(1), 75–86, (2001).
[6] B. Anrig and J. Kohlas, 'Model-based reliability and diagnostic: A common framework for reliability and diagnostics', Technical Report 02-01, Department of Informatics, University of Fribourg, (2002).
[7] R. E. Barlow and R. Proschan, *Statistical Theory of Reliability and Life Testing*, New York, 1975. IAUTOM 3.9.4-10.
[8] F. Beichelt, *Zuverlässigkeits- und Instandhaltungstheorie*, Teubner, Stuttgart, 1993.
[9] R. Bertschy and P.-A. Monney, 'A generalization of the algorithm of Heidtmann to non-monotone formulas', *J. of Computational and Applied Mathematics*, **76**, 55–76, (1996).
[10] R. Davis, 'Diagnostic reasoning based on structure and behaviour', *Artif. Intell.*, **24**, 347–410, (1984).
[11] J. De Kleer and B. C. Williams, 'Diagnosing multiple faults', *Artif. Intell.*, **32**, 97–130, (1987).
[12] R. Haenni, 'Cost-bounded argumentation', *Int. J. of Approximate Reasoning*, **26**(2), 101–127, (2001).
[13] R. Haenni, 'A query-driven anytime algorithm for assumption-based reasoning', Technical Report 01-26, University of Fribourg, Department of Informatics, (2001).
[14] R. Haenni, B. Anrig, R. Bissig, and N. Lehmann. ABEL homepage. `http://diuf.unifr.ch/tcs/abel`, 2000.
[15] R. Haenni, J. Kohlas, and N. Lehmann, 'Probabilistic argumentation systems', in *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, eds., J. Kohlas and S. Moral, volume 5: Algorithms for Uncertainty and Defeasible Reasoning, Kluwer, Dordrecht, (2000).
[16] K. D. Heidtmann, 'Smaller sums of disjoint products by subproduct inversion', *IEEE Transactions on Reliability*, **38**(3), 305–311, (August 1989).
[17] P. H. Ibargüengoytia, L. E. Sucar, and E. Morales, 'A probabilistic model approach for fault diagnosis', in *DX'00, Eleventh Intl. Workshop on Principles of Diagnosis, Morelia, Mexico*, eds., A. Darwiche and G. M. Provan, pp. 79–86, (2000).
[18] J. Kohlas. Valuation algebras: Generic architecture for reasoning. draft, 2002.
[19] J. Kohlas, B. Anrig, R. Haenni, and P.-A. Monney, 'Model-based diagnostics and probabilistic assumption-based reasoning', *Artif. Intell.*, **104**, 71–106, (1998).
[20] J. Kohlas and P.-A. Monney, *A Mathematical Theory of Hints. An Approach to the Dempster-Shafer Theory of Evidence*, volume 425 of *Lecture Notes in Economics and Mathematical Systems*, Springer, 1995.
[21] J. Kohlas and R. F. Stärk, 'Information algebras and information systems', Technical Report 96–14, University of Fribourg, Institute of Informatics, (1996).
[22] G. M. Provan, 'An integration of model-based diagnosis and reliability theory', in *DX'00, Eleventh Intl. Workshop on Principles of Diagnosis, Morelia, Mexico*, eds., A. Darwiche and G. M. Provan, pp. 193–200, (2000).
[23] R. Reiter, 'A theory of diagnosis from first principles', *Artif. Intell.*, **32**, 57–95, (1987).
[24] G. Shafer, *The Mathematical Theory of Evidence*, Princeton University Press, 1976.
[25] P. P. Shenoy and G. Shafer, 'Axioms for probability and belief functions propagation', in *Uncertainty in Artif. Intell. 4*, eds., R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer. North Holland, (1990).
[26] P. Struss, 'Testing for discrimination of diagnoses', in *DX'94, Fifth Intl. Workshop on Principles of Diagnosis, New Paltz, USA*, (1994).

Far-sighted Diagnosis of Active Systems

Roberto Garatti and **Gianfranco Lamperti** and **Marina Zanella** [1]

Abstract. Active systems are a class of discrete-event systems modeled as networks of nondeterministic automata communicating through either synchronous or asynchronous connection links. The model-based diagnosis of an active system is carried out by first reconstructing its behavior based on the observation, from which faults are later derived. The complexity of behavior reconstruction is exacerbated by the possibility of queuing events within links, thereby making essential the simulation of the order in which events are buffered within links. Unfortunately some sequences of events may lead to blind alleys in the search space. This is especially critical if events exchanged among components are assumed to be uncertain, as the number of alternative sequences of queued events is still larger. Therefore, behavior reconstruction without any prospection in the search space is generally bound to detrimental backtracking. To make diagnosis of active systems more efficient, we present an off-line technique for processing the models inherent to the system at hand so as to automatically generate prospection knowledge relevant to the mode in which events are produced and consumed over links. Such a knowledge is then exploited on-line, when the diagnostic engine is running, to guide the search process, thus reducing both time and space.

1 INTRODUCTION

Diagnosis of discrete-event systems (DESs) is a complex and challenging task that has been receiving an increasing interest from both the model-based diagnosis community [9], within the AI area, and the fault detection and isolation (FDI) community [16, 8, 10], within the automatic control area. The current shared prospect is that, in the general case, the specific faults cannot be inferred without first finding out what has happened to the system to be diagnosed. Once the system evolution is available, the sets of candidate faults can be derived from it.

In this respect, in spite of slightly different terminologies, such as histories [2], situation histories or narratives [4], paths [5], and trajectories [11, 6], all the distinct approaches describe the evolution of a DES as a sequence interleaving states and transitions, as the favorite behavioral models of DESs in the literature are automata.

Based on the method for tracking the evolutions of the system that explain a given observation, two broad categories of approaches to diagnosis of DESs can be basically singled out:

- Those that first generate (a concise/partial model of) all possible evolutions and then retrieve only the evolutions that explain the observation;
- Those that generate in one shot the evolutions explaining the observation.

The first category includes some relevant works from both the automatic control area [19, 20, 7, 15] and the AI area [12, 6]. Embodied in the second category are some approaches of the AI area [2, 11, 17].

Since finding out the system evolutions is a computationally expensive and, therefore, inefficient process (see, for instance, [18] about the computational difficulties of the diagnoser approach [19, 20], or the worst case computational complexity analysis in [2], or the discussion in [11]), most of the approaches exploit a trade-off between off-line and on-line computation.

Focusing on the second category outlined above, the decentralized diagnoser approach [17] draws off-line a local diagnoser for each component. Such a diagnoser is an automaton whose states and (observable) transitions are labeled with compiled knowledge about unobservable paths and interacting components, respectively. Each local diagnoser is employed on-line for both a more efficient reconstruction of all the possible evolutions of the relevant component that comply with the observation and a more efficient merging of the histories of distinct components into global system histories.

This paper applies knowledge compilation to the active system approach [2, 3], to which purpose it isolates a kind of knowledge, implicit in the models of the structure and behavior of the system at hand, that can be compiled off-line in order to speed up on-line execution. The framework is that of active systems, a class of DESs modeled as networks of nondeterministic automata communicating through directed links. If an active system includes one or more asynchronous buffered links, its reaction to an event coming from the external world is assumed to continue until there is no event left in the links. The component that sends events on a link is the event *producer* and that extracting them from the link is the *consumer*. The knowledge we compile is actually that inherent to the producer-consumer relationships between components. In particular, we present, by means of an example:

- An extension of both the modeling primitives and the on-line 'short-sighted' evolution reconstruction method so as to cope with uncertain events;
- A method for generating off-line, under the form of a deterministic automaton, called a *prospection graph*, the model of the way events are exchanged over one or more links;
- A 'far-sighted' method for exploiting prospection graphs on-line while reconstructing the evolutions of (sub)systems.

Finally, the computational advantages of far-sighted diagnosis are discussed and some conclusions are hinted.

2 ACTIVE SYSTEMS WITH UNCERTAIN EVENTS

Topologically, an active system Σ is a network of *components* which are connected to one another through *links*. Each component is completely modeled by an automaton which reacts to events either coming from the external world or from neighboring components through links. Formally, the automaton is a 6-tuple

$$(\mathbf{S}, \mathbf{E}_{\mathrm{in}}, \mathbf{I}, \mathbf{E}_{\mathrm{out}}, \mathbf{O}, \mathbf{T})$$

[1] Dipartimento di Elettronica per l'Automazione, Università di Brescia, via Branze 38, 25123 Brescia, Italy, email: garatrob@tin.it, lamperti@ing.unibs.it, zanella@ing.unibs.it

where $\mathbf{S}$ is the set of *states*, $\mathbf{E}_{\rm in}$ the set of *input events*, $\mathbf{I}$ the set of *input terminals*, $\mathbf{E}_{\rm out}$ the set of *output events*, $\mathbf{O}$ the set of *output terminals*, and $\mathbf{T}$ the (nondeterministic) *transition function*:

$$\mathbf{T} : \mathbf{S} \times \mathbf{E}_{\rm in} \times \mathbf{I} \times 2^{\mathbf{E}_{\rm out} \times \mathbf{O}} \mapsto 2^{\mathbf{S}}.$$

A transition from state S to state S', which is triggered by the input event $\alpha = (E, I)$, $E \in \mathbf{E}_{\rm in}$, $I \in \mathbf{I}$, and generates the set $\beta = \{(E_1, O_1), \ldots, (E_n, O_n)\}$ of output events, $E_k \in \mathbf{E}_{\rm out}$, $O_k \in \mathbf{O}$, $k \in [1\,..\,n]$, is denoted by

$$S \xrightarrow{\alpha \,|\, \beta} S'.$$

Components are implicitly equipped with three *virtual terminals*, the *standard input* ($In \in \mathbf{I}$) for events coming from the external world, the *standard output* ($Out \in \mathbf{O}$) for events directed toward the external world (messages), and the *fault terminal* ($Flt \in \mathbf{O}$) for modeling faulty transitions.

An event (E, Flt) is a *fault event*. The approach assumes that both nominal and faulty behavior of each component are specified in the automaton. A fault event is not exchanged among components. Rather, it is a formal artifice to describe the faulty behavior of components uniformly. The name of fault events are supposed to be informative as to the specific fault affecting the component when the relevant transition is performed[2].

An event may be *uncertain* in nature, that is, represented by a disjunction of possible values. Links are the means of storing the events exchanged between components.

Each link L is characterized by a 4-tuple

$$(I, O, \chi, P)$$

where I is the *input terminal* (connected with a component output terminal), O the *output terminal* (connected with a component input terminal), χ the *capacity*, that is, the maximum number of queued events, and P the *saturation policy*, which dictates the effect of the triggering of a transition T attempting to insert a new event E into L when L is *saturated*, that is, when the length of the queue equals χ. Three cases are possible:

- *LOSE*: E is lost;
- *OVERRIDE*: E replaces the last event in the queue of L;
- *WAIT*: T cannot be triggered until L becomes unsaturated, that is, until at least one event in L is consumed.

The *queue domain* $\mathbf{Q}$ of L is the set of possible sequences (queues) of events in L. The length of the queue Q of events incorporated in L is denoted by $|Q|$.

The polymorphic $Link$ function is defined as follows. Let

$$\alpha = (E, \theta)$$

represent an event relevant to a terminal θ. Then,

$$Link(\alpha) \stackrel{\rm def}{=} L \mid L \text{ is the link connected with } \theta.$$

No more than one link can be connected with a component terminal. If θ is a virtual terminal, then $Link(\alpha) \stackrel{\rm def}{=} null$. Let

$$\beta = \{(E_1, \theta_1), \ldots, (E_n, \theta_n)\}$$

[2] For example, consider a breaker which is in the state *open* and is expected to change state to *close* when it receives a command (nominal behavior). The possible misbehavior of the breaker can be defined by inserting a faulty transition, from state *open* to *open*, that generates the fault event $(stuckToOpen, Flt)$.

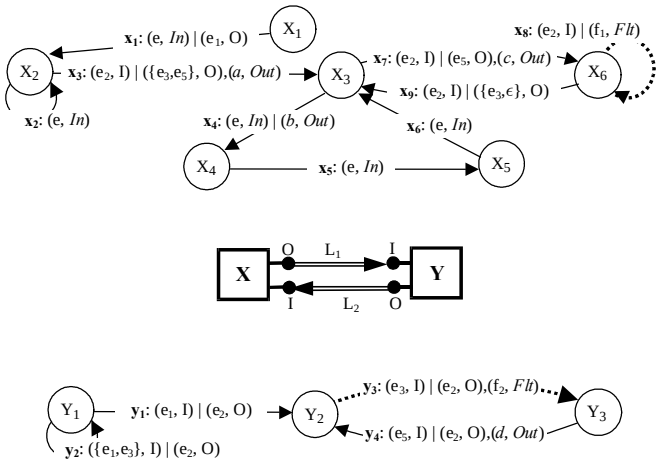


Figure 1. System Ψ and models of components X (top) and Y (bottom).

be a set of events relevant to terminals θ_i, $i \in [1\,..\,n]$, respectively. Then,

$$Link(\beta) \stackrel{\rm def}{=} \{L_\beta \mid L_\beta = Link(\mathcal{E}), \mathcal{E} \in \beta\}.$$

Initially, Σ is in a *quiescent* state Σ_0, wherein all links are empty. At the arrival of an event from the external world, Σ becomes *reacting*, thereby making a series of transitions until a final quiescent state is reached, wherein all links are empty anew. This *reaction* yields a sequence of observable events, the *messages*, which make up a system *observation* $OBS(\Sigma)$.

Let Σ_0 denote the initial state of system Σ. Based on a *diagnostic problem*

$$\wp(\Sigma) = (OBS(\Sigma), \Sigma_0)$$

a *reconstruction* of the system reaction is carried out, which yields an *active space*, that is, a graph representing the whole set of *candidate histories*, each history being a path from Σ_0 to a final state, in other terms, a sequence of component transitions which *explains* $OBS(\Sigma)$.

Candidate *diagnoses* are eventually distilled from the active space, each diagnosis being a set of faulty components, that is, those components which made at least one *faulty transition* during a candidate system history.

Example 1. Displayed in the center of Figure 1 is a system Ψ, where X and Y are components, while L_1 and L_2 are links. Both components are endowed with an input terminal I and an output terminal O. For both links we assume $\chi = 1$ and $P = WAIT$. The behavioral models of X and Y are displayed on the top and on the bottom, respectively. Accordingly, Y involves three states ($Y_1 \cdots Y_3$) and four transitions ($y_1 \cdots y_4$), one of which is faulty (y_3) (states and transitions are denoted by capital and small letters, respectively). For instance, transition y_4 is triggered by the input event (e_5, I) and generates the set of output events $\{(e_2, O), (d, Out)\}$, where the former is directed toward X on link L_2, while the latter is a message labeled d (y_4 is said to be *observable*). Transition y_2 involves the input event $(\{e_1, e_3\}, I)$, meaning that y_2 may either be triggered by e_1 or e_3. Considering the model of X, note that, when triggered, transition x_3 generates the uncertain event $(\{e_3, e_5\}, O)$, meaning that either e_3 or e_5 is randomly generated (no assumption is made about the likelihood of event generation). Likewise, x_9 generates the uncertain event $(\{e_3, \epsilon\}, O)$, meaning that either e_3 or nothing is generated (ϵ denotes the *null* event). □

3 SHORT-SIGHTED DIAGNOSIS

The main task relevant to the resolution of a diagnostic problem $\wp(\Sigma) = (OBS(\Sigma), \Sigma_0)$ is the reconstruction of the system reaction to make up the relevant active space $Act(\wp(\Sigma))$. A node N in the search space is identified by three fields, $N = (\sigma, \Im, \mathcal{Q})$, where:

- $\sigma = (S_1, \ldots, S_n)$ is the record of states of the system components, each S_i, $i \in [1\,..\,n]$, being a state relevant to a component C_i in Σ (n is the number of components in Σ);
- $\Im$ is the *index* of $OBS(\Sigma)$, that is, an integer ranging from 0 to the number of messages (length) of $OBS(\Sigma)$, which implicitly denotes the prefix of the observation composed of the first $\Im$ messages;
- $\mathcal{Q} = (Q_1, \ldots, Q_\ell)$ is the record of queues of the ℓ links in Σ.

Node N is said to be *final* when $\Im$ equals the length of $OBS(\Sigma)$ and all links are empty. The search for the nodes of the active space is started at the initial node $N_0 = (\Sigma_0, 0, (\langle\rangle, \ldots, \langle\rangle))$, where all link queues are empty. Each successor node of a given node is obtained by applying a component transition that is consistent with both the system topology and the observation. An applied transition is an edge of the search space. When the reconstruction process is carried out in one step (*monolithically*) without any prospection knowledge (*short-sightedly*), it can be described by Algorithm 1, where nodes and edges generated during the search are stored in variables $\aleph$ and $\mathcal{E}$, respectively.

Algorithm 1. (*Short-sighted Reconstruction*)

1. $\aleph = \{N_0\}$; $\mathcal{E} = \emptyset$; (N_0 is unmarked)
2. *Repeat Steps 3 through 5 until all nodes in* $\aleph$ *are marked;*
3. *Get an unmarked node* $N = (\sigma, \Im, \mathcal{Q})$ *in* $\aleph$;
4. *For each* i *in* $[1\,..\,n]$, *for each transition* T *within the model of component* C_i, *if* T *is triggerable, that is, if its triggering event is available within the link and* T *is consistent with both* $OBS(\Sigma)$ *and the link policy (when* T *generates output events on non-virtual terminals), do the following steps:*
   (a) *Create a node* $(N' = (\sigma', \Im', \mathcal{Q}')) := N$; ($N'$ is created as a copy of N)
   (b) $\sigma'[i] :=$ *the state reached by* T;
   (c) *If* T *is observable, then* $\Im := \Im + 1$; (a message is generated)
   (d) *If the triggering event* E *of* T *is relevant to an internal link* L_j, *then remove* E *from* $\mathcal{Q}'[j]$;
   (e) *Insert the internal output events of* T *into the relevant queues in* $\mathcal{Q}'$;
   (f) *If* $N' \notin \aleph$ *then insert* N' *into* $\aleph$; (N' is unmarked)
   (g) *Insert edge* $N \xrightarrow{T} N'$ *into* $\mathcal{E}$;
5. *Mark* N;
6. *Remove from* $\aleph$ *all the nodes and from* $\mathcal{E}$ *all the edges that are not on a path from the initial state* N_0 *to a final state in* $\aleph$.

The algorithm aims to make up all the nodes which are reachable from the initial node under the given observation. To this end, it considers, one at a time, all the nodes which have been reached already (those in $\aleph$) and have not yet been processed (the unmarked ones). For each of them, it attempts to find a transition that is triggerable by a component in the corresponding state. If so, it generates the target node N' with the appropriate values σ', $\Im'$, and $\mathcal{Q}'$. In the new node was not created already, it is inserted into $\aleph$ (note that two nodes which differ in the $\Im$ field only have to be considered different, as the mode in which messages have been generated differ). The corresponding edge $N \xrightarrow{T} N'$ is inserted into $\mathcal{E}$ too. Finally,

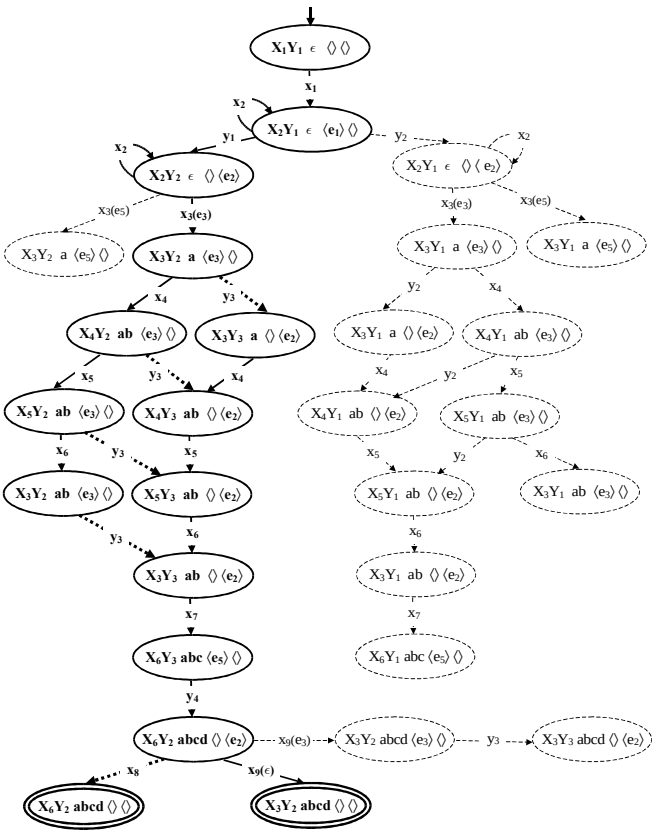


Figure 2. Short-sighted reconstruction space (see Example 2).

when there are no more nodes to be processed (all nodes in $\aleph$ are marked), the search space is pruned by eliminating the inconsistent nodes, that is, those that are on a blind alley.

It is worthwhile highlighting that the search process does not terminate at a final node. In fact, the system might continue to react and loop on unobservable paths. In other words, when a final node is met in the search, it is inserted into $\aleph$ as an unmarked node like all other nodes, since in principle, unobservable paths might happen to leave it.

When uncertain output events are involved, several new nodes N' are to be generated for the same transition T, specifically, one for each combination of possible values within each disjunction. For example, since transition x_3 in Figure 1 involves the uncertain output event $(\{e_3, e_5\}, O)$, two target nodes will be generated, one for e_3 and one for e_5. If the set of output events included several uncertain events, all possible combinations would be required to be enumerated.

Example 2. Shown in Figure 2 is the reconstruction space generated short-sightedly for the diagnostic problem $\wp(\Psi) = (OBS(\Psi), \Psi_0)$, where Ψ is the system outlined in Figure 1, $OBS(\Psi) = \langle a, b, c, d\rangle$, and $\Psi_0 = (X_1, Y_1)$. Each node is depicted by an ellipse, wherein

- $\sigma = (X_i, Y_j)$ is the pair of component states;
- $\Im$ is the prefix of the observation generated so far;
- $\mathcal{Q} = (Q_1, Q_2)$ is the pair of link queues.

Edges are marked by the corresponding component transitions, possibly qualified by the relevant chosen label when the involved output event is uncertain. Dotted edges denote faulty transitions. Final nodes are depicted as double ellipses. The dashed part of the graph

corresponds to inconsistent states, which are almost half the search space. Owing to cycles in the graph (edges marked by x_2), the active space includes an unbound number of candidate histories. However, only two candidate diagnoses are possible, namely $\{Y\}$ and $\{X, Y\}$. Note that, although not relevant to our example, the replication of the same faulty transition in a cycle does not change the diagnosis. A finer-grained diagnosis can be defined, as in [2], called *deep diagnosis*. The latter is a set of pairs (C, f), where C is a component and f a fault event. This way, even if not relevant to our example where each component model includes a single faulty transition, it is possible to know all the faulty transitions performed by each misbehaving component. □

4 FAR-SIGHTED DIAGNOSIS

The essential problem with short-sighted diagnosis lies in the lack of any prospection in the search space as to the consistency of the link queues. In other words, the inability to understand that a given configuration of $\mathcal{Q}$ is bound to a 'blind alley' forces the reconstruction algorithm to uselessly explore possibly large parts of the search space. In order to overcome this limitation, prospection knowledge can be automatically generated off-line based on the system model. Considering Figure 2, such a knowledge will allow the reconstruction process to avoid entering the inconsistent sub-space through y_2.

The basic idea is to view a link L as a buffer in which a *producer* component C^{p} generates events that are consumed by a *consumer* component C^{c}. That is, L connects an output terminal of C^{p} to an input terminal of C^{c}. The way events are produced and consumed in L is both constrained by the characteristics of the link (capacity and saturation policy) and the models of C^{p} and C^{c}.

4.1 Prospection graphs

Let $L = (I, O, \chi, P)$ be a link from output terminal O^{p} of component C^{p} to input terminal I^{c} of component C^{c}, with queue domain $\mathbf{Q}$. Let $M^{\mathrm{p}} = (\mathbf{S}^{\mathrm{p}}, \mathbf{E}^{\mathrm{p}}_{\mathrm{in}}, \mathbf{I}^{\mathrm{p}}, \mathbf{E}^{\mathrm{p}}_{\mathrm{out}}, \mathbf{O}^{\mathrm{p}}, \mathbf{T}^{\mathrm{p}})$ and $M^{\mathrm{c}} = (\mathbf{S}^{\mathrm{c}}, \mathbf{E}^{\mathrm{c}}_{\mathrm{in}}, \mathbf{I}^{\mathrm{c}}, \mathbf{E}^{\mathrm{c}}_{\mathrm{out}}, \mathbf{O}^{\mathrm{c}}, \mathbf{T}^{\mathrm{c}})$ be the models of C^{p} and C^{c}, respectively. Let

$$\hat{M}^{\mathrm{p}^{\mathrm{n}}} = (\hat{\mathbf{S}}^{\mathrm{p}^{\mathrm{n}}}, \hat{\mathbf{E}}^{\mathrm{p}^{\mathrm{n}}}, \hat{\mathbf{T}}^{\mathrm{p}^{\mathrm{n}}})$$

be the nondeterministic automaton obtained from M^{p} in such a way that

- $\hat{\mathbf{S}}^{\mathrm{p}^{\mathrm{n}}} = \mathbf{S}^{\mathrm{p}}$ is the set of states;
- $\hat{\mathbf{E}}^{\mathrm{p}^{\mathrm{n}}} \subseteq \mathbf{T}^{\mathrm{p}} \cup \{\epsilon\}$ is the set of events;
- $\hat{\mathbf{T}}^{\mathrm{p}^{\mathrm{n}}} : \hat{\mathbf{S}}^{\mathrm{p}^{\mathrm{n}}} \times \hat{\mathbf{E}}^{\mathrm{p}^{\mathrm{n}}} \mapsto 2^{\hat{\mathbf{S}}^{\mathrm{p}^{\mathrm{n}}}}$ is the transition function.

The transition function $\hat{\mathbf{T}}^{\mathrm{p}^{\mathrm{n}}}$ is obtained from $\mathbf{T}^{\mathrm{p}}$ as follows:

$$\forall T = S \xrightarrow{\alpha|\beta} S' \in \mathbf{T}^{\mathrm{p}} \begin{cases} S \xrightarrow{\epsilon} S' \in \hat{\mathbf{T}}^{\mathrm{p}^{\mathrm{n}}} & \text{if } L \notin Link(\beta) \\ S \xrightarrow{T} S' \in \hat{\mathbf{T}}^{\mathrm{p}^{\mathrm{n}}} & \text{otherwise.} \end{cases}$$

Similarly, let

$$\hat{M}^{\mathrm{c}^{\mathrm{n}}} = (\hat{\mathbf{S}}^{\mathrm{c}^{\mathrm{n}}}, \hat{\mathbf{E}}^{\mathrm{c}^{\mathrm{n}}}, \hat{\mathbf{T}}^{\mathrm{c}^{\mathrm{n}}})$$

be the nondeterministic automaton obtained from M^{c} in such a way that

- $\hat{\mathbf{S}}^{\mathrm{c}^{\mathrm{n}}} = \mathbf{S}^{\mathrm{c}}$ is the set of states;
- $\hat{\mathbf{E}}^{\mathrm{c}^{\mathrm{n}}} \subseteq \mathbf{T}^{\mathrm{c}} \cup \{\epsilon\}$ is the set of events;
- $\hat{\mathbf{T}}^{\mathrm{c}^{\mathrm{n}}} : \hat{\mathbf{S}}^{\mathrm{c}^{\mathrm{n}}} \times \hat{\mathbf{E}}^{\mathrm{c}^{\mathrm{n}}} \mapsto 2^{\hat{\mathbf{S}}^{\mathrm{c}^{\mathrm{n}}}}$ is the transition function.

The transition function $\hat{\mathbf{T}}^{\mathrm{c}^{\mathrm{n}}}$ is obtained from $\mathbf{T}^{\mathrm{c}}$ as follows:

$$\forall T = S \xrightarrow{\alpha|\beta} S' \in \mathbf{T}^{\mathrm{c}} \begin{cases} S \xrightarrow{\epsilon} S' \in \hat{\mathbf{T}}^{\mathrm{c}^{\mathrm{n}}} & \text{if } L \neq Link(\alpha) \\ S \xrightarrow{T} S' \in \hat{\mathbf{T}}^{\mathrm{c}^{\mathrm{n}}} & \text{otherwise.} \end{cases}$$

Let $\hat{M}^{\mathrm{p}} = (\hat{\mathbf{S}}^{\mathrm{p}}, \hat{\mathbf{E}}^{\mathrm{p}}, \hat{\mathbf{T}}^{\mathrm{p}})$ and $\hat{M}^{\mathrm{c}} = (\hat{\mathbf{S}}^{\mathrm{c}}, \hat{\mathbf{E}}^{\mathrm{c}}, \hat{\mathbf{T}}^{\mathrm{c}})$ be the deterministic automata equivalent to $\hat{M}^{\mathrm{p}^{\mathrm{n}}}$ and $\hat{M}^{\mathrm{c}^{\mathrm{n}}}$, respectively. A *prospection state* $\mathcal{L}$ of L is a triple

$$\mathcal{L} = (\hat{S}^{\mathrm{p}}, \hat{S}^{\mathrm{c}}, Q) \in \hat{\mathbf{S}}^{\mathrm{p}} \times \hat{\mathbf{S}}^{\mathrm{c}} \times \mathbf{Q}.$$

Let $\mathcal{L}$ be a prospection state and $\hat{S} \xrightarrow{T} \hat{S}' \in (\hat{\mathbf{T}}^{\mathrm{p}} \cup \hat{\mathbf{T}}^{\mathrm{c}})$, $\hat{S} \in \{\hat{S}^{\mathrm{p}}, \hat{S}^{\mathrm{c}}\}$, $T = S \xrightarrow{\alpha|\beta} S' \in (\mathbf{T}^{\mathrm{p}} \cup \mathbf{T}^{\mathrm{c}})$. Let Q be a queue of events in L and

- $Head(Q)$ denote the first consumable event in Q;
- $Tail(Q)$ denote the sequence of events in Q following the first event;
- $App(Q, e)$ denote the queue obtained by appending e to Q;
- $Repl(Q, e)$ denote the queue obtained by replacing the last event in Q with e.

The $Next$ function yields the set of next prospection states as follows:

$$Next(\mathcal{L}, T) \stackrel{\mathrm{def}}{=} \{\mathcal{L}' \mid \mathcal{L}' \in Next^{\mathrm{p}}(\mathcal{L}, T), T \in \mathbf{T}^{\mathrm{p}}\} \cup \{\mathcal{L}' \mid \mathcal{L}' \in Next^{\mathrm{c}}(\mathcal{L}, T), T \in \mathbf{T}^{\mathrm{c}}\}$$

where

$$Next^{\mathrm{p}}(\mathcal{L}, T) \stackrel{\mathrm{def}}{=} \{\mathcal{L}' \mid \mathcal{L}' = (\hat{S}', \hat{S}^{\mathrm{c}}, Q'), B = (E, O^{\mathrm{p}}) \in \beta, e \in E, Q' = Ins(Q, e), (|Q| < \chi \text{ or } (|Q| = \chi, (e = \epsilon \text{ or } P \in \{LOSE, OVERRIDE\})))\},$$

$$Ins(Q, e) \stackrel{\mathrm{def}}{=} \begin{cases} App(Q, e) & \text{if } |Q| < \chi \\ Q & \text{if } |Q| = \chi, (e = \epsilon \text{ or } P = LOSE) \\ Repl(Q, e) & \text{if } |Q| = \chi, P = OVERRIDE \end{cases}$$

and

$$Next^{\mathrm{c}}(\mathcal{L}, T) \stackrel{\mathrm{def}}{=} \{\mathcal{L}' \mid \mathcal{L}' = (\hat{S}^{\mathrm{p}}, \hat{S}', Q'), \alpha = (E, I^{\mathrm{c}}), e \in E, Head(Q) = e, Q' = Tail(Q)\}.$$

Let $\mathcal{C}_0 = (S^{\mathrm{p}}_0, S^{\mathrm{c}}_0)$ be the pair of initial states for C^{p} and C^{c}, respectively. The *spurious prospection graph* of L and $\mathcal{C}_0$ is the nondeterministic automaton

$$\widetilde{\Gamma}^{\mathrm{n}}(L, \mathcal{C}_0) = (\widetilde{\mathbf{S}}^{\mathrm{n}}, \mathbf{E}^{\mathrm{n}}, \widetilde{\mathbf{T}}^{\mathrm{n}}, S^{\mathrm{n}}_0, \mathbf{S}^{\mathrm{n}}_{\mathrm{f}})$$

where

$\widetilde{\mathbf{S}}^{\mathrm{n}} = \{\mathcal{L} \mid \mathcal{L} \text{ is a prospection state of } L\}$ is the set of states,
$\mathbf{E}^{\mathrm{n}} \subseteq \hat{\mathbf{E}}^{\mathrm{p}} \cup \hat{\mathbf{E}}^{\mathrm{c}} \subseteq \mathbf{T}^{\mathrm{p}} \cup \mathbf{T}^{\mathrm{c}}$ is the set of events,
$S^{\mathrm{n}}_0 = (S^{\mathrm{p}}_0, S^{\mathrm{c}}_0, \langle\rangle)$ is the initial state,
$\mathbf{S}^{\mathrm{n}}_{\mathrm{f}} = \{\mathcal{L} \mid \mathcal{L} \in \mathbf{S}^{\mathrm{n}}, \mathcal{L} = (S^{\mathrm{p}}, S^{\mathrm{c}}, \langle\rangle)\}$ is the set of final states,
$\widetilde{\mathbf{T}}^{\mathrm{n}} : \widetilde{\mathbf{S}}^{\mathrm{n}} \times \mathbf{E}^{\mathrm{n}} \mapsto 2^{\widetilde{\mathbf{S}}^{\mathrm{n}}}$ is the transition function defined as follows:

$$\mathcal{L} \xrightarrow{T} \mathcal{L}' \in \widetilde{\mathbf{T}}^{\mathrm{n}} \text{ iff } \mathcal{L}' \in Next(\mathcal{L}, T).$$

A state of a spurious prospection graph which is not within a path from the initial state to a final state is an *inconsistent state*. Similarly, a transition entering or leaving an inconsistent state of a spurious prospection graph is an *inconsistent transition*.

The *nondeterministic prospection graph* is the nondeterministic automaton

$$\Gamma^{\mathrm{n}}(L, \mathcal{C}_0) = (\mathbf{S}^{\mathrm{n}}, \mathbf{E}^{\mathrm{n}}, \mathbf{T}^{\mathrm{n}}, S^{\mathrm{n}}_0, \mathbf{S}^{\mathrm{n}}_{\mathrm{f}})$$

obtained from $\widetilde{\Gamma}^{\mathrm{n}}(L, \mathcal{C}_0)$ by removing inconsistent states and inconsistent transitions.

The *prospection graph*

$$\Gamma(L, \mathcal{C}_0) = (\mathbf{S}, \mathbf{E}, \mathbf{T}, S_0, \mathbf{S}_{\mathrm{f}})$$

is the deterministic automaton equivalent to the nondeterministuic prospection graph $\Gamma^{\mathrm{n}}(L, \mathcal{C}_0)$.

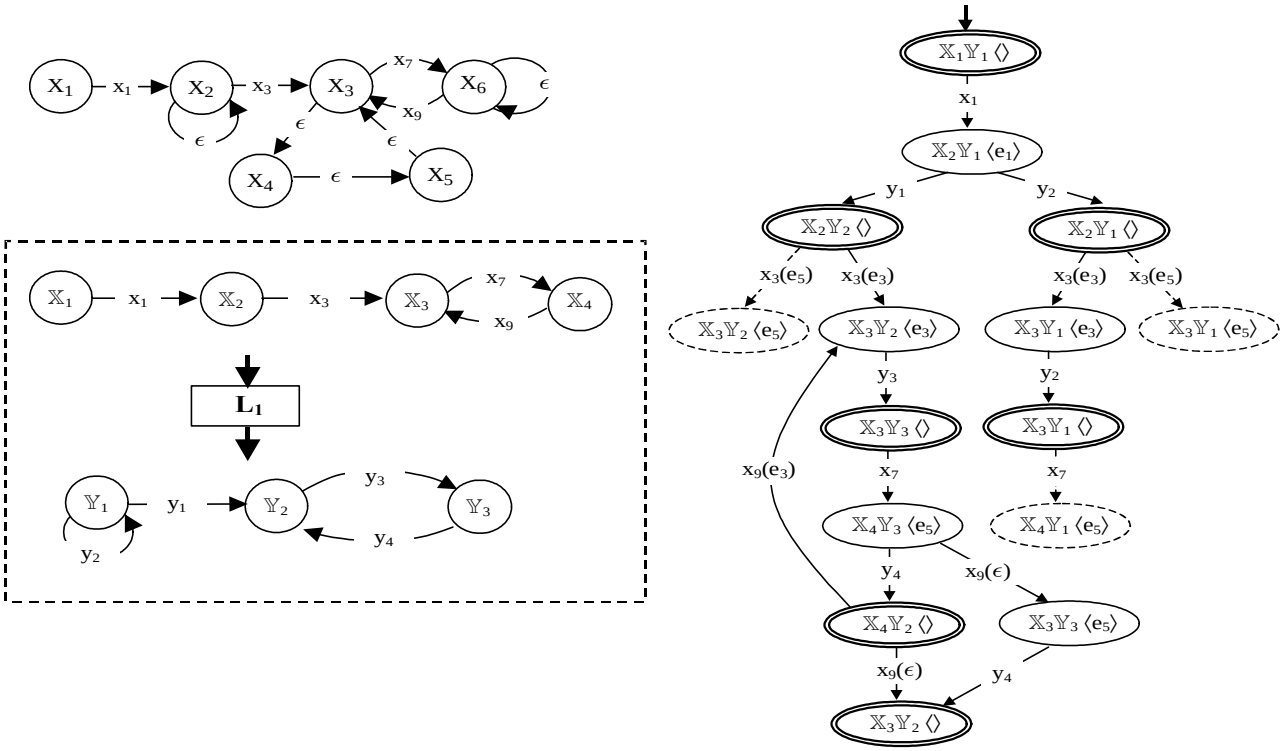


Figure 3. Generation of $\Gamma^{\mathrm{n}}(L_1, (X_1, Y_1))$ (see Example 3).

Example 3. Shown in the dashed box of Figure 3 are the prospection models $\hat{M}^{\mathrm{p}}(X)$ (top) and $\hat{M}^{\mathrm{c}}(Y)$ (bottom), inherent to link L_1, which are relevant to the components X and Y displayed in Figure 1. Depicted on the top of the box is the nondeterministic automaton $\hat{M}^{\mathrm{p}^{\mathrm{n}}}(X)$ equivalent to $\hat{M}^{\mathrm{p}}(X)$. The generation of the nondeterministic prospection graph $\Gamma^{\mathrm{n}}(L_1, (X_1, Y_1))$ is outlined on the right of Figure 4, where double ellipses denote final states, while dashed nodes and edges represent inconsistent states and transitions, respectively. Note that the latter includes a circular path involving four states. This situation is similar to that of active systems, where cycles may stem from (possibly) final states. Within the context of prospection graphs, cycles represent repetitive patterns of link state changes (in our example, events e_3 and e_5 are repeatedly produced and consumed, that is, inserted into and removed from link L_1). □

Note that, essentially, the generation of a prospection graph is analogous to the generation of an active space, where

- Component models are substituted by prospection models;
- Only one link is considered;
- No observation index is considered.

4.1.1 Generalized prospection graphs

The notion of the prospection graph of a single link can be naturally extended to that of a set of links. Let $\mathbb{L} = \{L_1, \ldots, L_m\}$ be a set of links (with queue domains $\mathbf{Q}_1, \ldots, \mathbf{Q}_m$, respectively) connecting a set $\mathbb{C} = \{C_1, \ldots, C_t\}$ of components, where each component C_i, $i \in [1\,..\,t]$, is characterized by model

$$M_i = (\mathbf{S}_i, \mathbf{E}_{\mathrm{in}_i}, \mathbf{I}_i, \mathbf{E}_{\mathrm{out}_i}, \mathbf{O}_i, \mathbf{T}_i).$$

Let $\hat{M}_i^{\mathrm{n}} = (\hat{\mathbf{S}}_i^{\mathrm{n}}, \hat{\mathbf{E}}_i^{\mathrm{n}}, \hat{\mathbf{T}}_i^{\mathrm{n}})$ be the nondeterministic automaton obtained from M_i in such a way that

- $\hat{\mathbf{S}}_i^{\mathrm{n}} = \mathbf{S}_i$ is the set of states;
- $\hat{\mathbf{E}}_i^{\mathrm{n}} \subseteq \mathbf{T}_i \cup \{\epsilon\}$ is the set of events;
- $\hat{\mathbf{T}}_i^{\mathrm{n}} : \hat{\mathbf{S}}_i^{\mathrm{n}} \times \hat{\mathbf{E}}_i^{\mathrm{n}} \mapsto 2^{\hat{\mathbf{S}}_i^{\mathrm{n}}}$ is the transition function.

The transition function $\hat{\mathbf{T}}_i^{\mathrm{n}}$ is obtained from $\mathbf{T}_i$ as follows:

$$\forall T = S \xrightarrow{\alpha|\beta} S' \in \mathbf{T}_i \begin{cases} S \xrightarrow{T} S' \in \hat{\mathbf{T}}_i^{\mathrm{n}} & \text{if } \mathit{Relevant}(\alpha, \beta, \mathbb{L}) \\ S \xrightarrow{\epsilon} S' \in \hat{\mathbf{T}}_i^{\mathrm{n}} & \text{otherwise} \end{cases}$$

where

$$\mathit{Relevant}(\alpha, \beta, \mathbb{L}) \stackrel{\mathrm{def}}{=} (\{\mathit{Link}(\alpha)\} \cup \mathit{Link}(\beta)) \cap \mathbb{L} \neq \emptyset.$$

Let $\hat{M}_i = (\hat{\mathbf{S}}_i, \hat{\mathbf{E}}_i, \hat{\mathbf{T}}_i)$ be the deterministic automaton equivalent to $\hat{M}_i^{\mathrm{n}}$. A *generalized prospection state* $\mathfrak{L}$ of $\mathbb{L}$ is a pair

$$\mathfrak{L} = (\mathbb{S}, \mathbb{Q})$$

where

$$\mathbb{S} = (\hat{S}_1, \ldots, \hat{S}_t) \in (\hat{\mathbf{S}}_1 \times \cdots \times \hat{\mathbf{S}}_t),$$
$$\mathbb{Q} = (Q_1, \ldots, Q_m) \in (\mathbf{Q}_1 \times \cdots \times \mathbf{Q}_m).$$

Let $\mathfrak{L} = (\mathbb{S}, \mathbb{Q})$ be a generalized prospection state and $\hat{S}_i \xrightarrow{T} \hat{S}' \in \hat{\mathbf{T}}_i$, $i \in [1\,..\,t]$, $T = S \xrightarrow{\alpha|\beta} S'$.

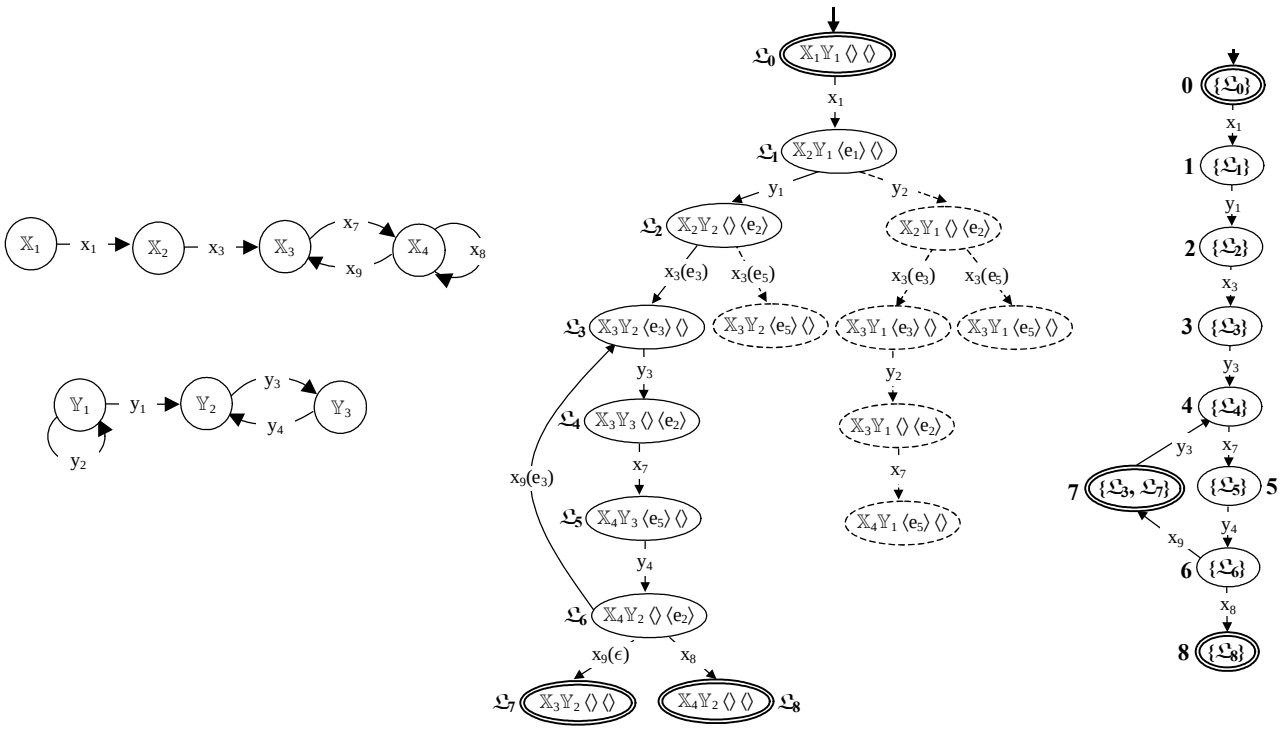


Figure 4. Generation of the generalized prospection graph $\Gamma(\mathbb{L}, \Psi_0)$ (see Example 4).

The *generalized Next* function yields the set of next generalized prospection states as follows:

$$
\begin{aligned}
&Next(\mathfrak{L}, T) \stackrel{\text{def}}{=} \{\mathfrak{L}' \mid \mathfrak{L}' = (\mathbb{S}', \mathbb{Q}'), \mathbb{S}' = (\hat{S}'_1, \ldots, \hat{S}'_t),\\
&\quad \mathbb{Q}' = (Q'_1, \ldots, Q'_m), \alpha = (E_\alpha, I_\alpha),\\
&\quad ((Link(I_\alpha) \notin \mathbb{L}) \text{ or}\\
&\quad (Link(I_\alpha) = L_j, L_j \in \mathbb{L}, e \in E_\alpha, Head(Q_j) = e,\\
&\qquad Q'_j = Tail(Q_j))),\\
&\quad \mathbb{L}_\beta = \{L_\beta \mid L_\beta = Link(O_\beta), (E_\beta, O_\beta) \in \beta, L_\beta \in \mathbb{L}\},\\
&\quad \forall L_h \in \mathbb{L}_\beta (e \in E_\beta, (E_\beta, O_\beta) \in \beta, L_h = Link(O_\beta),\\
&\qquad Q'_h = Ins(Q_h, e),\\
&\qquad (|Q_h| < \chi_h \text{ or}\\
&\qquad (|Q_h| = \chi_h, (e = \epsilon \text{ or } P_h \in \{LOSE, OVERRIDE\}))),\\
&\quad \forall L_k \in (\mathbb{L} - (\mathbb{L}_\beta \cup \{Link(I_\alpha)\}))\ (Q'_k = Q_k),\\
&\quad \hat{S}'_i = \hat{S}', \forall x \in [1\,..\,t], x \neq i\ (\hat{S}'_x = \hat{S}_x)\}.
\end{aligned}
$$

Let $\mathbb{C}_0 = (S_{0_1}, \ldots, S_{0_t})$ be the record of initial states for components in $\mathbb{C}$. The *generalized spurious prospection graph* of $\mathbb{L}$ and $\mathbb{C}_0$ is the nondeterministic automaton

$$\widetilde{\Gamma}^{\text{n}}(\mathbb{L}, \mathbb{C}_0) = (\widetilde{\mathbf{S}}^{\text{n}}, \mathbf{E}^{\text{n}}, \widetilde{\mathbf{T}}^{\text{n}}, S_0^{\text{n}}, \mathbf{S}_{\text{f}}^{\text{n}})$$

where

$\widetilde{\mathbf{S}}^{\text{n}} = \{\mathfrak{L} \mid \mathfrak{L}$ is a prospection state of $\mathbb{L}\}$ is the set of states,
$\mathbf{E}^{\text{n}} \subseteq \bigcup_{i=1}^{t} \hat{\mathbf{E}}_i \subseteq \bigcup_{i=1}^{t} \mathbf{T}_i$ is the set of events,
$S_0^{\text{n}} = (\mathbb{C}_0, (\langle\rangle \cdots \langle\rangle))$ is the initial state,
$\mathbf{S}_{\text{f}}^{\text{n}} = \{\mathfrak{L} \mid \mathfrak{L} \in \mathbf{S}^{\text{n}}, \mathfrak{L} = (\mathbb{S}, (\langle\rangle \cdots \langle\rangle))\}$ is the set of final states,
$\widetilde{\mathbf{T}}^{\text{n}} : \widetilde{\mathbf{S}}^{\text{n}} \times \mathbf{E}^{\text{n}} \mapsto 2^{\widetilde{\mathbf{S}}^{\text{n}}}$ is the transition function defined as follows:

$$\mathfrak{L} \xrightarrow{T} \mathfrak{L}' \in \widetilde{\mathbf{T}}^{\text{n}} \text{ iff } \mathfrak{L}' \in Next(\mathfrak{L}, T).$$

The *generalized nondeterministic prospection graph* is the nondeterministic automaton

$$\Gamma^{\text{n}}(\mathbb{L}, \mathbb{C}_0) = (\mathbf{S}^{\text{n}}, \mathbf{E}^{\text{n}}, \mathbf{T}^{\text{n}}, S_0^{\text{n}}, \mathbf{S}_{\text{f}}^{\text{n}})$$

obtained from $\widetilde{\Gamma}^{\text{n}}(\mathbb{L}, \mathbb{C}_0)$ by removing inconsistent states and inconsistent transitions.

The *generalized prospection graph*

$$\Gamma(\mathbb{L}, \mathbb{C}_0) = (\mathbf{S}, \mathbf{E}, \mathbf{T}, S_0, \mathbf{S}_{\text{f}})$$

is the deterministic automaton equivalent to the $\Gamma^{\text{n}}(\mathbb{L}, \mathbb{C}_0)$.

Example 4. Shown in Figure 4 is the generation of the generalized prospection graph $\Gamma(\mathbb{L}, \Psi_0)$ relevant to the links in system Ψ (see Figure 1), where $\mathbb{L} = \{L_1, L_2\}$ and $\Psi_0 = (X_1, Y_1)$. Specifically, outlined on the left are the prospection models of components X and Y, namely $\hat{M}(X)$ and $\hat{M}(Y)$. Shown on the center is the generation of the generalized nondeterministic prospection graph $\Gamma^{\text{n}}(\mathbb{L}, \Psi_0)$ (the dash part of the graph denotes the inconsistent search space), where consistent nodes are identified by labels $\mathfrak{L}_0 \cdots \mathfrak{L}_6$. Finally, displayed on the right is the corresponding deterministic prospection graph $\Gamma(\mathbb{L}, \Psi_0)$. The latter is determined based on the *subset construction algorithm* presented in [1], which identifies each node of the deterministic automaton by means of a subset of nodes of the nondeterministic one, specifically, those nodes that are reachable through the same marking transition. For example, since there are two edges, leaving the same state $\mathfrak{L}_6$ in the nondeterministic automaton, that are marked by the same label x_9, the deterministic automaton will include the node identified by the subset $\{\mathfrak{L}_3, \mathfrak{L}_7\}$, which is reached from $\{\mathfrak{L}_6\}$ by means of the (unique) edge marked by x_9. According to the algorithm, each node in the deterministic automaton that includes a final state of the nondeterministic one is final itself. Nodes of the deterministic automaton are identified by labels $0 \cdots 8$. □

Given a system Σ, in order to exploit the prospection knowledge in the reconstruction process, we need to create a set of g prospection graphs

$$\mathbf{\Gamma}(\Sigma) = \{\Gamma(\mathbb{L}_1, \mathbb{C}_{0_1}), \ldots, \Gamma(\mathbb{L}_g, \mathbb{C}_{0_g})\}$$

such that $\bigcup_{i=1}^{g} \mathbb{L}_i$ equals the whole set of links in Σ. $\mathbf{\Gamma}(\Sigma)$ is a *prospection coverage* of Σ.

Algorithm 2. (*Far-sighted Reconstruction*)

The far-sighted reconstruction algorithm is a variation of Algorithm 1. First, the $\mathcal{Q}$ field of a node denotes a record of g states relevant to the g prospection graphs in the prospection coverage $\mathbf{\Gamma}(\Sigma)$, namely

$$\mathcal{Q} = (\gamma_1, \ldots, \gamma_g).$$

Moreover, in the initial node $N_0 = (\sigma_0, \Im_0, \mathcal{Q}_0)$, $\mathcal{Q}_0$ is represented by the record of the initial states of the corresponding prospection graphs, namely $(\gamma_{0_1}, \ldots, \gamma_{0_g})$. Finally, Step 4 of Algorithm 1 is changed as follows:

For each i in $[1\,..\,n]$, for each transition T within the model of component C_i, if T is triggerable, that is, if the following two conditions hold

(i) T is consistent with $OBS(\Sigma)$;

Let $\Pi(T) = \{\bar{\Gamma}_1, \ldots, \bar{\Gamma}_r\}$ be the prospection graphs in $\mathbf{\Gamma}(\Sigma)$ that are relevant to links connected with terminals on which events are either consumed or generated by T; let $\bar{\mathcal{Q}}(N) = \{\bar{\gamma}_1, \ldots, \bar{\gamma}_r\}$ be the elements of $\mathcal{Q}(N)$ relevant to $\Pi(T)$:

(ii) $\forall i \in [1\,..\,r]$ ($\bar{\gamma}_i \xrightarrow{T} \bar{\gamma}_i'$ is an edge in $\bar{\Gamma}_i$),

then do the following steps:

(a) Create a node $(N' = (\sigma', \Im', \mathcal{Q}')) := N$;

(b) $\sigma'[i] :=$ the state reached by T;

(c) If T is observable, then $\Im := \Im + 1$;

(d) Replace the elements of $\mathcal{Q}'$ relevant to $\bar{\mathcal{Q}}(N)$ with the new prospection states;

(e) If $N' \notin \aleph$ then insert N' into $\aleph$;

(f) Insert edge $N \xrightarrow{T} N'$ into $\mathcal{E}$.

Essentially, Algorithm 2 exploits the knowledge about the consistency of link states by means of the prospection graphs generated off-line, thereby preventing the search from entering (possibly large) inconsistent parts of the space. Of course, such a prospection is finite, thereby not eliminating completely the backtracking. Besides, it allows for an efficient treatment of nondeterminism caused by uncertain events. Recall that, in short-sighted reconstruction, such situations can only be dealt with by mere enumeration of all possible new link states generated by the collection of output events of the current transition. For example, if T generated 3 uncertain events (on three different links), each of which represented by a disjunction of 2 values, then we would have 8 new nodes. Instead, since the prospection graphs are deterministic, with far-sighted reconstruction only one new node is generated, as at most one edge marked by T can leave each current state of the prospection graphs.

Proposition 1. *Let $\wp(\Sigma)$ be a diagnostic problem and $\|A\|$ denote the (possibly unbound) set of histories incorporated in an active space A. Let $Act^{\mathrm{s}}(\wp(\Sigma))$ and $Act^{\mathrm{f}}(\wp(\Sigma))$ denote the active spaces generated by Algorithm 1 and Algorithm 2, respectively. Then,*

$$\|Act^{\mathrm{s}}(\wp(\Sigma))\| = \|Act^{\mathrm{f}}(\wp(\Sigma))\|.$$

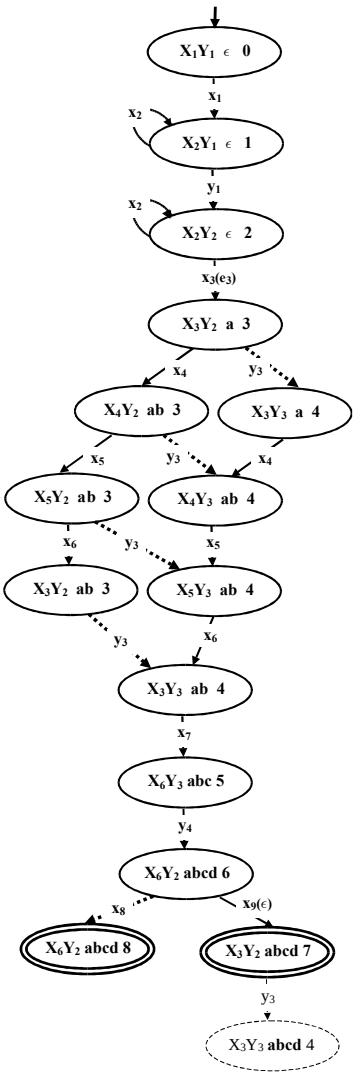


Figure 5. Far-sighted reconstruction space (see Example 4).

Example 5. Shown in Figure 5 is the reconstruction space for the diagnostic problem $\wp(\Psi) = (\langle a, b, c, d\rangle, (X_1, Y_1))$ based on the generalized prospection graph outlined on the right of Figure 4. It is striking comparing it with the short-sighted reconstruction (based on Algorithm 1) displayed in Figure 2. While the number of consistent states (15) is necessarily equal in both reconstructions, the far-sighted reconstruction space includes one inconsistent state only, against the 14 inconsistent states of the short-sighted reconstruction space. In fact, while the two states on top of both graphs are the same, there is a right branch stemming from the latter of such states in the short-sighted reconstruction which is missing in the far-sighted reconstruction. This branching is actually disabled by prospection graph $\Gamma(\{L_1, L_2\}, (X_1, Y_1))$, which constraints the occurrence of all the transitions involved in event exchange on the links of system Ψ: according to this prospection graph, only transition y_1 is allowed to follow x_1, while y_2, the responsible for the blind alley in Figure 2, is not. □

5 CONCLUSION

Referring to the active system approach [2, 3] to diagnosis of DESs, this paper has shown how the off-line compilation of knowledge

about event exchange between components brings a computational advantage on-line in terms of reduction of the number of backtracking steps performed by the history reconstruction algorithm. This advantage is expecially tangible when relaxing a strong assumption of all the state-of-the-art approaches to diagnosis of DESs, namely, the preciseness of events. In this work, all input and output events in behavioral models, and not only observable events, as instead in [14], may have an imprecise value ranging over a set of labels, namely an *uncertain* value. In presence of uncertain events, the search performed by short-sighted diagnosis is nondeterministic, while that carried out with the support of prospection knowledge is deterministic. Moreover, prospection graphs, once generated off-line, can be reused several times on-line for different diagnostic problems inherent to the same system, or even for the same diagnostic problem in case there are repetitive link patterns in the system structure.

A previous proposal [13], based itself on knowledge compilation, transforms the active system approach into a spectrum of approaches which, according to the classification in Section 1, range from a totally first category version, wherein an exhaustive simulation of the system evolution is performed off-line, while on-line activities are limited to rule-checking, to a totally second category version, i.e. the original approach wherein no computation is performed off-line. Each approach falling in between consists of both off-line and on-line processing. The contribution of this paper is orthogonal to that work, that is, it could be integrated within any version of the spectrum (with the exception of the exclusively on-line one) in order to reduce backtracking steps in any reconstruction.

The exchange of events among components dealt with in this paper, being both asynchronous and buffered, is peculiar only to the active system approach. One might argue that providing for a specific modeling primitive, namely the link, for the structural objects that implement asynchronous buffered communication between components, along with specific methods for dealing with them, just increases the expressive power of the method but does not alter its computational power at all. In fact, each link could be replaced by a common component, whose behavioral model represents the link behavior, and, therefore, synchronous composition of automata would suffice. This is correct in principle but scarcely feasible in practice, for many reasons. First, the size of the behavioral model of such a component depends not only on the capacity of the link buffer but also on the number of distinct kinds of events that can be transmitted on the link. For instance, let us consider a link with capacity equal to three, on which four kinds of events, say a, b, c, and d, can be transmitted. As each state of the component representing the link is univocally identified by the sequence of events in the buffer, the behavioral model of such a component has $\sum_{k=0}^{3}(4^k) = 85$ states! So large a model is a burden for history reconstruction. In fact, the model may be unduly large as it includes even states that are physically impossible given the system structure, since corresponding to sequences of events that cannot be generated.

Besides, as remarked above, such a model depends on the kinds of events that can be transmitted on the link, that is, it depends on the producer component of the link at hand. This is somewhat in contrast with the philosophy of compositional modeling, according to which individual component models are reciprocally independent.

Instead, in the active system approach and, consequently, in this paper, a link is just the instantiation of a model, encompassing only the terminals, capacity, and policy of the link, and such a model is independent of the structure of the system in which the link is instantiated. Of course, notwithstanding the modeling simplicity, link states are bound to emerge in the computation, sooner or later. The methods introduced in this paper are actually aimed at minimizing the number of physically impossible link states (and, hence, since a link state is a part of any active system state, the number of active space states) visited by the history reconstruction search algorithm. In short-sighted diagnosis, where a link state is represented as a sequence of events, not all sequences of events are considered but only those that can be generated given the system structure. In far-sighted diagnosis, where the state of one or several links becomes a record of indexes, the number of visited link states is further reduced: only those states are generated that can evolve towards a state wherein the link is empty.

REFERENCES

[1] A. Aho, R. Sethi, and J.D. Ullman, *Compilers – Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986.

[2] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, 'Diagnosis of large active systems', *Artificial Intelligence*, **110**(1), 135–183, (1999).

[3] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, 'Diagnosis of a class of distributed discrete-event systems', *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **30**(6), 731–752, (2000).

[4] C. Barral, S. McIlraith, and T.C. Son, 'Formulating diagnostic problem solving using an action language with narratives and sensing', in *Seventh International Conference on Knowledge Representation and Reasoning – KR'2000*, pp. 311–322, Breckenridge, Colorado, (2000).

[5] L. Console, C. Picardi, and M. Ribaudo, 'Diagnosis and diagnosability using PEPA', in *Fourteenth European Conference on Artificial Intelligence – ECAI'2000*, pp. 131–135, Berlin, D, (2000).

[6] M.O. Cordier and C. Largouët, 'Using model-checking techniques for diagnosing discrete-event systems', in *Twelfth International Workshop on Principles of Diagnosis – DX'01*, pp. 39–46, San Sicario, I, (2001).

[7] R. Debouk, S. Lafortune, and D. Teneketzis, 'Coordinated decentralized protocols for failure diagnosis of discrete-event systems', *Journal of Discrete Event Dynamical Systems: Theory and Application*, **10**, 33–86, (2000).

[8] P.M. Frank, 'Analytical and qualitative model-based fault diagnosis – a survey and some new results', *European Journal of Control*, **2**, 6–28, (1996).

[9] *Readings in Model-Based Diagnosis*, eds., W. Hamscher, L. Console, and J. de Kleer, Morgan Kaufmann, San Mateo, CA, 1992.

[10] R. Isermann, 'Supervision, fault detection and fault-diagnosis methods – an introduction', *Control Engineering Practice*, **5**(5), 639–652, (1997).

[11] J. Kurien and P.P. Nayak, 'Back to the future for consistency-based trajectory tracking', in *Eleventh International Workshop on Principles of Diagnosis – DX'00*, pp. 92–100, Morelia, MX, (2000).

[12] P. Laborie and J.P. Krivine, 'Automatic generation of chronicles and its application to alarm processing in power distribution systems', in *Eighth International Workshop on Principles of Diagnosis – DX'97*, Mont St. Michel, F, (1997).

[13] G. Lamperti and M. Zanella, 'Generation of diagnostic knowledge by discrete-event model compilation', in *Seventh International Conference on Knowledge Representation and Reasoning – KR'2000*, pp. 333–344, Breckenridge, Colorado, (2000).

[14] G. Lamperti and M. Zanella, 'Uncertain temporal observations in diagnosis', in *Fourteenth European Conference on Artificial Intelligence – ECAI'2000*, pp. 151–155, Berlin, D, (2000).

[15] J. Lunze, 'Diagnosis of quantized systems based on timed discrete-event model', *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **30**(3), 322–335, (2000).

[16] R.J. Patton and J. Chen, 'A review of parity space appproaches to fault diagnosis', in *IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes – SAFEPROCESS'91*, Baden-Baden, D, (1991).

[17] Y. Pencolé, 'Decentralized diagnoser approach: application to telecommunication networks', in *Eleventh International Workshop on Principles of Diagnosis – DX'00*, pp. 185–192, Morelia, MX, (2000).

[18] L. Rozé, 'Supervision of telecommunication network: a diagnoser approach', in *Eighth International Workshop on Principles of Diagnosis – DX'97*, Mont St. Michel, F, (1997).

[19] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis, 'Diagnosability of discrete-event systems', *IEEE Transactions on Automatic Control*, **40**(9), 1555–1575, (1995).

[20] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis, 'Failure diagnosis using discrete-event models', *IEEE Transactions on Control Systems Technology*, **4**(2), 105–124, (1996).

Model-based Monitoring of Piecewise Continuous Behaviors using Dynamic Uncertainty Space Partitioning[1]

Bernhard Rinner[2] and **Ulrich Weiss**[2]

Abstract. Monitoring gains importance for many technical systems such as robots, production lines or anti lock brakes. A monitoring system for technical systems must be able to deal with incomplete knowledge of the supervised system, to process noisy observations and to react within predefined time windows. This paper presents a new approach to monitoring technical systems based on imprecise models. Our approach repeatedly partitions the uncertainty space of an imprecise model and checks the derived model's state for consistency with the measurements. Inconsistent partitions are then refuted resulting in a smaller uncertainty space and a faster failure detection. This paper further focuses on the extension of our basic approach to monitoring systems that exhibit both continuous and discrete behaviors. Our monitoring system has been implemented using COTS components and has been demonstrated in online monitoring of a non-trivial heating system.

Keywords: fault detection; hybrid systems; imprecise models; residual generation

1 INTRODUCTION

The primary objective of a monitoring system is to detect abnormal behaviors of a supervised system as soon as possible to avoid shutdown or damage. Technical systems such as robots, production lines or anti lock brakes provide a vast number of challenges for a monitoring system, i.e., it must be able to deal with incomplete knowledge about the supervised system, to process noisy observations and to react within predefined time windows.

A particularly important and widely-applied approach is *model-based monitoring* [6, 5] which relies on a comparison of the predicted behavior of a model with the observed behavior of the supervised system. Our approach using dynamic uncertainty space partitioning [12] is based on imprecise models where the structure of the models is known and the parameters may be imprecisely given as numeric intervals. These parameter intervals span the uncertainty space of the model. From an imprecise model based on intervals only bounds on the trajectory (envelopes) can be derived. Dynamic uncertainty space partitioning keeps the envelopes small by exploiting the measurements from the supervised system as soon as possible. Whenever new measurements arrive residuals are generated at the "corner points" of the uncertainty space and checked for consistency by comparing their signs. This results in a fast fault detection [12].

The fundamental assumption of dynamic uncertainty space partitioning is that the model's state values are monotonic within the range of the uncertainty space. Discontinuous transitions in the system's model may introduce non-monotonic behaviors in the state values and, therefore, violate our assumption for the consistency check. In order to preserve a conservative monitoring approach for hybrid systems, we have to extend our consistency check by a *monotonicity check*. Whenever the monotonicity of the state values is given the consistency check can be performed potentially resulting in a refutation of the imprecise model. If the monotonicity is not known the consistency check is simply ignored and no model is refuted.

The remainder of this paper is organized as follows. Section 2 describes the technical details of uncertainty space partitioning and the consistency check. Section 3 discusses the necessary extensions of our approach to monitoring systems which exhibit both continuous and discrete behaviors. Section 4 presents experimental results of our monitoring approach in a real-world system with several changes of a input value. A discussion and a summary of related work conclude this paper.

2 MONITORING BASED ON UNCERTAINTY SPACE PARTITIONING

2.1 Overview

Monitoring methods based on imprecise models can reason with incomplete knowledge in the model as well as with noisy measurements. A main drawback of this approach, however, is that the envelopes may diverge very rapidly which delays or even inhibits a fault recognition. We have revised this interval approach to model-based monitoring with the primary goal to keep the resulting envelopes as small as possible.

In our approach, we exploit the measurements from the supervised system as soon as possible to refine the uncertainty in the model and the derived envelopes. The key step in our approach is to partition the uncertainty space of the model into several subspaces. The trajectories derived from each subspace are then checked for consistency with the measurements. Each inconsistent subspace is refuted and excluded from further investigations. Partitioning and consistency checking are continued resulting in a smaller uncertainty space of the model. When all subspace are refuted, a discrepancy between model prediction and observation has been recognized and a fault has been detected.

2.2 Subspace Partitioning and Consistency Checking

In general, a technical system can be modeled as

$$\begin{aligned} \mathbf{x}_t &= \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{p}_{t-1}) \\ \mathbf{y}_t &= \mathbf{g}(\mathbf{x}_t, \mathbf{p}_t) \end{aligned} \tag{1}$$

[1] This work has been supported by the Austrian Science Fund under grant number P14233-INF. The authors are in alphabetical order.

[2] Institute for Technical Informatics, Graz University of Technology, AUSTRIA; email:[rinner,uweiss]@iti.tu-graz.ac.at.

where $\mathbf{x}_t$ is the state vector at discrete time t, $\mathbf{u}_t$ is the input vector at time t, $\mathbf{p}_t$ is the parameter vector at time t, $\mathbf{y}_t$ is the output vector at time t, and $\mathbf{g}$ and $\mathbf{f}$ are vector functions. In an exact model, $\mathbf{p}_t$ is a vector of real numbers. However, in a model with uncertain parameters, $\mathbf{p}_t$ is replaced by a vector of intervals $\tilde{\mathbf{p}}_t = [(\underline{p}_{1,t}, \overline{p}_{1,t}), (\underline{p}_{2,t}, \overline{p}_{2,t}), \cdots, (\underline{p}_{K,t}, \overline{p}_{K,t})]^T$, where K is the number of uncertain parameters. A model with uncertain parameters, i.e., an *imprecise model*, can therefore be described as:

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \mathbf{f}(\tilde{\mathbf{x}}_{t-1}, \mathbf{u}_{t-1}, \tilde{\mathbf{p}}_{t-1}) \\ \tilde{\mathbf{y}}_t &= \mathbf{g}(\tilde{\mathbf{x}}_t, \tilde{\mathbf{p}}_t) \end{aligned} \tag{2}$$

Equation 2 is the starting point of our approach. It defines an imprecise model of the supervised system with K uncertain parameters. Thus, this model has a K-dimensional uncertainty space. In order to divide this uncertainty space we have to define a *partition* $\tilde{\mathbf{q}}_t = [(\underline{q}_{1,t}, \overline{q}_{1,t}), (\underline{q}_{2,t}, \overline{q}_{2,t}), \cdots, (\underline{q}_{K,t}, \overline{q}_{K,t})]^T$ with $\tilde{\mathbf{q}}_t \subseteq \tilde{\mathbf{p}}_t$. A complete partitioning of the uncertainty space at any time t into M partitions must satisfy the following condition $\bigcup_m \tilde{\mathbf{q}}_t^{(m)} = \tilde{\mathbf{p}}_t$ where $m = 1, \ldots, M$. A model based on a partition of the uncertainty space is referred to as *subspace model*. From the definition of a partition, we can finally define the state of a subspace model m:

$$\begin{aligned} \tilde{\mathbf{x}}_t^{(m)} &= \mathbf{f}(\tilde{\mathbf{x}}_{t-1}^{(m)}, \mathbf{u}_{t-1}, \tilde{\mathbf{q}}_{t-1}^{(m)}) \\ \tilde{\mathbf{y}}_t^{(m)} &= \mathbf{g}(\tilde{\mathbf{x}}_t^{(m)}, \tilde{\mathbf{q}}_t^{(m)}). \end{aligned} \tag{3}$$

With the monotonicity assumption of $\mathbf{f}$ and $\mathbf{g}$ with regard to the parameters $\mathbf{p}_t$ over the range of the intervals, the (uncertain) state of a subspace model can be represented by the (exact) state of the *corner points* of a subspace. The corner points of a subspace are defined as all combinations of upper and lower bounds of a partition $\tilde{\mathbf{q}}_\mathbf{t}$ and can be represented as set $Q_t^{(m)} = \{\tilde{\mathbf{q}}_{t,i}^{(m)}\}$ with $i = 1, \ldots, 2^K$. Thus, an uncertainty space of dimension K results in 2^K corner points. The states at the corner points can be represented as set

$$\begin{aligned} \mathbf{X}_t^{(m)} &= \{\mathbf{x}_{t,i}^{(m)} : \mathbf{x}_{t,i}^{(m)} = \mathbf{f}(\mathbf{x}_{t-1,i}^{(m)}, \mathbf{u}_{t-1}, \mathbf{q}_{t-1,i}^{(m)})\} \\ \mathbf{Y}_t^{(m)} &= \{\mathbf{y}_{t,i}^{(m)} : \mathbf{y}_{t,1}^{(m)} = \mathbf{g}(\mathbf{x}_{t,i}^{(m)}, \mathbf{q}_{t,i}^{(m)})\} \end{aligned} \tag{4}$$

where $\mathbf{q}_{t,i}^{(m)}$ is an exact parameter vector at time t from the subspace m and at corner $i = 1, \ldots, 2^K$ of this subspace. Note, that $\mathbf{x}_{t,i}^{(m)}$ are state vectors, and also $\mathbf{y}_{t,i}^{(m)}$ are output vectors with exact values. Note that this approach assumes that the parameters of the system are constant, and are not varying in time. This assumption will be discussed later.

This representation of an uncertain state is directly exploited by our consistency check for a given subspace m. First, a *residual* is calculated for each state at a corner point using the measurements at time t, i.e., $\mathbf{r}_{t,i}^{(m)} = \mathbf{y}_{t,measured} - \mathbf{y}_{t,i}^{(m)}$, where $\mathbf{r}_{t,i}^{(m)}$ has the same dimension J as $\mathbf{y}_{t,measured}$ and $\mathbf{y}_{t,i}^{(m)}$. Then, the minimum and maximum values of the residual are determined as

$$r_{t,min,j}^{(m)} = \min_i \{r_{t,i,j}^{(m)}\} \tag{5}$$

$$r_{t,max,j}^{(m)} = \max_i \{r_{t,i,j}^{(m)}\} \tag{6}$$

with $i = 1, \ldots, 2^K$, and $j = 1, \ldots, J$. Finally, subspace model m is checked for consistency simply by comparing the signs of $r_{t,min,j}^{(m)}$ and $r_{t,max,j}^{(m)}$. The subspace model m is consistent with the measurements, iff

$$\text{sgn}(r_{t,min,j}^{(m)}) \neq \text{sgn}(r_{t,max,j}^{(m)}) \tag{7}$$

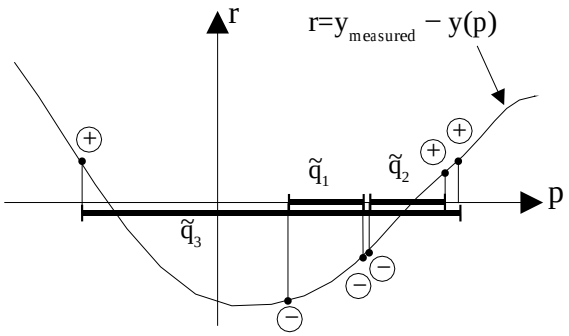


Figure 1. Consistency check with one uncertain parameter p and three subspaces $\tilde{q}_1$, $\tilde{q}_2$, and $\tilde{q}_3$. The residuals at the corner points of subspace $\tilde{q}_1$ are both negative, therefore, the model with the subspace $\tilde{q}_1$ is inconsistent with the measurement. In subspace $\tilde{q}_2$, the residuals at the corner points have different signs. Thus, $\tilde{q}_2$ is consistent. For the parameter range of subspace $\tilde{q}_3$ the monotonicity assumption is violated. In this case, checking the residuals' signs at the corner points is not feasible.

holds for all elements $j = 1, \ldots, J$.

Informally, Equation 7 checks whether the zero vector lies within the "residual subspace" (see Figure 1). If this equation is violated, the subspace model m is refuted. This simple consistency check holds also if not all elements of $\mathbf{y}$ are included in the measurements. In this case, a comparison with the missing elements is simply ignored. Since this technique is based on the calculation of an exact state (at corner points), we can use standard numerical methods for computing the solution of differential equations. Note that subspaces are only refuted when they are genuinely inconsistent with the measurements.

Due to the uncertainty in the parameters this method may result in diverging envelopes. This deviation of the predicted value to the "correct" value over time is referred to as *accumulation uncertainty*. In order to keep this deviation small we have also introduced a *dynamic* partitioning of the subspace models. During monitoring consistent subspaces are further partitioned resulting in smaller subspace models that potentially describe the supervised system more precisely [12].

3 MONITORING PIECEWISE CONTINUOUS BEHAVIORS

3.1 Monotonicity at Transitions

In order to extend our approach to monitoring piecewise continuous behaviors and discrete transitions, we must have a closer look at our monotonicity assumption. Remember that the result of our consistency check is only valid if the state values within the subspace are monotonic.

In general the monotonicity of the state values with regard to the parameters is not guaranteed by the monotonicity of the system equations $\mathbf{f}$ and $\mathbf{g}$. The monotonicity is only given when the following assumptions also hold:

1. the system input $\mathbf{u}$ does not change, and
2. the initial values of a subspace model are the same over its complete uncertainty space.

Both assumptions are important for monitoring discrete and continuous behaviors. The first assumption is especially relevant for transitions because they are often triggered by stepwise changes of the

system input (e.g., caused by operator actions). Such transitions violate, therefore, the first assumption. The second assumption is a simple consequence of the integration of the given differential equation:

$$\mathbf{x}(t) = \mathbf{x}_{t_0} + \int_{t_0}^{t} \dot{\mathbf{x}}(\tau) d\tau \tag{8}$$

If the initial states $\mathbf{x}_{t_0}$ are different at some corners in the subspace model, the state values $\mathbf{x}_t$ may not be monotonic (even if $\dot{\mathbf{x}}$ is monotonic). However, monotonicity is guaranteed after some time.

As discussed above discontinuous transitions may result in a non-monotonicity of the state values with regard to the parameters (for a limited period of time), which in turn leads to an incorrect consistency check. Thus, to maintain a correct (and conservative) monitoring technique we must extend the consistency check by a check for monotonicity. If the monotonicity is not guaranteed the consistency check is simply ignored and this subspace can not be refuted. At some time after the transition the subspace may become monotonic again and the consistency check can be applied again.

3.2 Checking for Monotonicity

The monotonicity of the state values for an individual subspace is checked by the following method.

We define a matrix $\mathbf{B}(t, \mathbf{x}, \mathbf{p})$ with the elements

$$b_{ij}(t, \mathbf{x}, \mathbf{p}) = \frac{\partial \dot{x}_i(t, \mathbf{x}, \mathbf{p})}{\partial p_j}, \tag{9}$$

where t is the time, $\mathbf{x}$ the state vector, and $\mathbf{p}$ the parameter vector with its elements p_j. We also define the matrix $\mathbf{C}(t, \mathbf{x}, \mathbf{p})$ with the elements

$$c_{ij}(t, \mathbf{x}, \mathbf{p}) = \frac{dx_i(t, \mathbf{p})}{dp_j}. \tag{10}$$

The matrix $\mathbf{C}(t, \mathbf{x}, \mathbf{p})$ is calculated by

$$\frac{d\mathbf{C}(t, \mathbf{x}, \mathbf{p})}{dt} = \mathbf{A}(t, \mathbf{x}, \mathbf{p})\mathbf{C}(t, \mathbf{x}, \mathbf{p}) + \mathbf{B}(t, \mathbf{x}, \mathbf{p}), \tag{11}$$

where $\mathbf{C}(0, \mathbf{x}_0, \mathbf{p}) = \mathbf{0}$ (the empty matrix), and the matrix $\mathbf{A}(t, \mathbf{x}, \mathbf{p})$ is defined as

$$a_{ij}(t, \mathbf{x}, \mathbf{p}) = \frac{\partial \dot{x}_i(t, \mathbf{x}, \mathbf{p})}{\partial x_j}. \tag{12}$$

The elements $c_{ij}(t, \mathbf{x}, \mathbf{p})$ give us the trend of the state value $x_i(t, \mathbf{p})$ with regard to the parameter p_j. This is exploited by our *monotonicity check*: The state values of a subspace model are monotonic, iff

$$\text{sgn}(c_{ij,min}) = \text{sgn}(c_{ij,max}) \tag{13}$$

holds for all state values $i = 1, \ldots, I$ and all directions of the uncertainty space $j = 1, \ldots, K$. $c_{ij,min}$ are the appropriate values of $c_{ij}(t, \mathbf{x}, \mathbf{p})$ at the corner min, and $c_{ij,max}$ are the values of $c_{ij}(t, \mathbf{x}, \mathbf{p})$ at the corner max of that subspace model (as described with Equations 5 and 6).

Figure 2 depicts the monotonicity check. In general, the information at the corner points is not sufficient to decide on monotonicity. However, assuming the monotonicity of the functions $\mathbf{f}$ and $\mathbf{g}$ with regard to the parameter, the monotonicity check becomes sufficient.

The calculation of the monotonicity check implies a numerical solution of the differential equation (Equation 12). However, since we use also a differential description of the system ($\mathbf{f} = \dot{\mathbf{x}}$), the monotonicity check does not significantly increase the computational load. Note that matrix $\mathbf{A}$ is constant for linear systems.

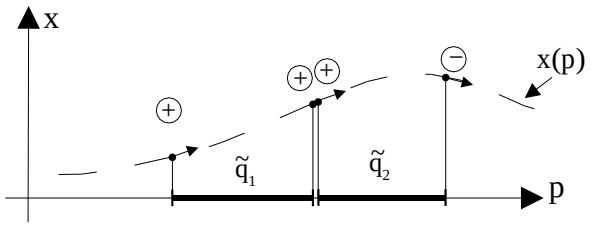


Figure 2. Monotonicity check with one state value and one parameter. To check the subspace model for monotonicity, the gradients of the state values with regard to the parameters are calculated at the corner points. In this example, the subspace $\tilde{q}_1$ is monotone and the subspace $\tilde{q}_2$ violates the monotonicity check.

4 THE MONOTONICITY CHECK IN A REAL-WORLD SYSTEM

We now examine the monotonicity behavior on a "real" technical system which is comprised of three heating/cooling components mounted on a thermal conductive plate. A process control computer (B&R 2003) controls the three heating/cooling components. The measured samples as well as the control actions issued are transferred to the monitoring system via a RS 232 interface.

Our model which includes the three components with heating elements is given as

$$\begin{aligned} \dot{T}_1 &= \tfrac{1}{C_1}(q_{i1} - L_1(T_1 - T_0) - L_{12}(T_1 - T_2)) \\ \dot{T}_2 &= \tfrac{1}{C_2}(q_{i2} + L_{12}(T_1 - T_2) - L_2(T_2 - T_0) \\ &\quad -L_{23}(T_2 - T_3)) \\ \dot{T}_3 &= \tfrac{1}{C_3}(q_{i3} + L_{23}(T_2 - T_3) - L_3(T_3 - T_0)) \end{aligned} \tag{14}$$

where T_i is the temperature of the three components, C_i is the mass of the components, q_i is the heat flow into the components, L_i the thermal conductivity between the component i and the environment, L_{ij} the thermal conductivity between the component i and j, and T_0 the temperature of the environment. We can reduce the complexity of this model by exploiting the symmetric construction of the heating system ($L_3 = L_1, L_{23} = L_{12}, C_3 = C_1$) resulting in a total of five uncertain parameters.

The state vector is given as $\mathbf{x} = (T_1, T_2, T_3)^T$, the input vector as $\mathbf{u} = (q_{i1}, q_{i2}, q_{i3}, T_0)^T$, and the output vector as $\mathbf{y} = (T_1 + n_1, T_2 + n_2, T_3 + n_3)^T$, where n_i is the noise of each temperature sensor. The noise parameters are also included in the uncertainty space resulting in a total of eight uncertain parameters. Note that noise parameters are not dynamically partitioned into smaller intervals and they are not considered by the monotonicity check.

We have measured the input values with $q_{off} = 1.24W$ and $q_{on} = 34.8W$ (heating element is either turned off or turned on). With an initial refinement step, we get the parameter intervals as $L_1 = [0.12, 0.13]$, $L_2 = [0.15, 0.18]$, $L_{12} = [0.62, 0.73]$, $C_1 = [51, 54]$, $C_2 = [61, 65]$. The refinement step is performed in a single continuous behavior segment [12].

To examine the non-monotonic behavior in the system, we observe the system after a transition, and count the subspace models,

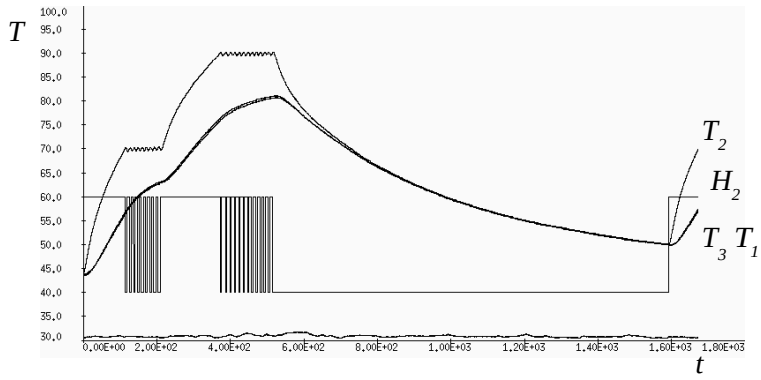


Figure 3. Measurements from the heating system used for monotonicity checking. The input H_2 is generated by the process control computer and sent to the monitoring system.

which are marked as non-monotonic. Over time, this gives us a picture, how the transition produce non-monotonicity in the state values. We choose the following scenario:

Control state 1: Heat T_2 until T_2 reaches 70. Then go to state 2.
Control state 2: Heat T_2, if $T_2 < 70$. If $t_{state2} \geq 100sec$, go to state 3.
Control state 3: Heat T_2, if $T_2 < 90$. If $t_{state3} \geq 100sec \wedge T_2 \geq 90$, go to state 4.
Control state 4: Do not heat. If $T_2 \leq 50$, go to state 1.

Figure 3 plots the resulting measurements for this scenario. The heating flag H_2 (generated by the PCC) is used, to get a discrete change of an input. To implement the heating element characteristic, we assume an additional mass C_h and a thermal conductivity L_{2h} between component 2 and the heating mass:

$$\dot{T}_{h2} = \frac{1}{C_h}(32.82H_2 - L_{2h}(T_{h2} - T_2)) \quad (15)$$

$$q_{i2} = 1.24 + L_{2h}(T_{h2} - T_2) \quad (16)$$

To demonstrate the non-monotonic effect after a stepwise change of an input, we check the monotonicity of all subspace models, and count non-monotonic subspace models, i.e., which violate Equation 13. Figure 4 shows a part of the scenario, where the temperature of component 2 is hold at 90 degree (control state 3). For this plot, we have started with 128 subspace models, and no dynamic partitioning is introduced. Due to the discrete controller the heating is turned on and off several times. At each transition about 40 subspace models are non-monotonic. An interesting observation in this figure is, that the non-monotonic subspaces disappear quickly, if the heating flag is turned off only for a short time.

Figure 5 shows the number of the non-monotonic subspace models after control state 3. The peak here is about 30 subspace models. It shows, that non-monotonic subspace models are also existing for a "longer" time period (here about 400 seconds) after the last discrete change of an input.

Non-monotonic subspace models are not refuted, and, therefore, do not make any contribution to decrease the uncertainty space. Although the number of non-monotonic subspace models are quite high (about 50 percent of the current subspace models) for some times. it has not a significantly influence to the refutation. The reason is, however, that such peaks does not hold for long time, so the consistency check soon becomes valid again. At this example the number of consistent subspace models at the end of the scenario is about 20.

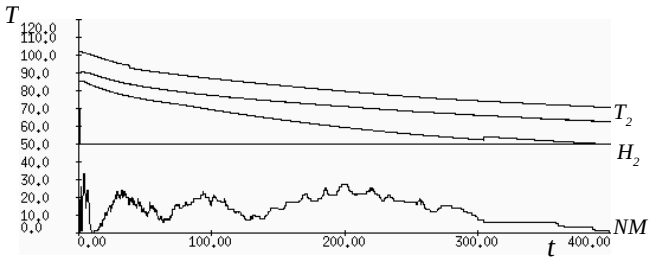


Figure 5. The non-monotonicity after the switching period. Drawn are (as same as in figure 4) the measurement and the envelopes of T_2, the heating flag H_2 and the number of non-monotonic subspace models NM. Some subspace models are non-monotonic after the heating period.

5 DISCUSSION

In this paper, we have presented a model-based monitoring approach based on uncertainty space partitioning. The fundamental assumption of this approach is the monotonicity of the state values with regard to the range of the parameters. In systems which exhibit both discrete and continuous behaviors the monotonicity can not be guaranteed only by the monotonicity of the vector functions. Thus, in order to apply our basic approach to monitor hybrid systems, we have introduced a monotonicity check for the state values.

Note the difference of monitoring based on pre-calculated envelopes with our approach. With pre-calculated envelopes, the envelopes remain constant over the complete monitoring process. In our approach, the envelopes may become smaller than the initial ones due to the refutation of inconsistent subspaces during monitoring. This results in an earlier detection of faults. However, there is a significant increase in the computational load of subspace partitioning.

Our approach is based on computing the envelopes of differential equations. For complex models, the overall runtime of our monitoring algorithm is dominated by solving the differential equations, especially when a high-precise method such as Runge-Kutta is used. The computational complexity of our algorithm for a single time-step can be estimated as

$$\mathcal{O}(M2^K(\rho + \mu)) \quad (17)$$

where M is the number of partitions, K is the number of uncertainty parameters, ρ is the time of the Runge-Kutta algorithm, and μ is the time of the matrix multiplication according to Equation 11. The time ρ strongly depends on the dynamic properties of the system, and for high dynamic systems, the assumption $\rho > \mu$ holds.

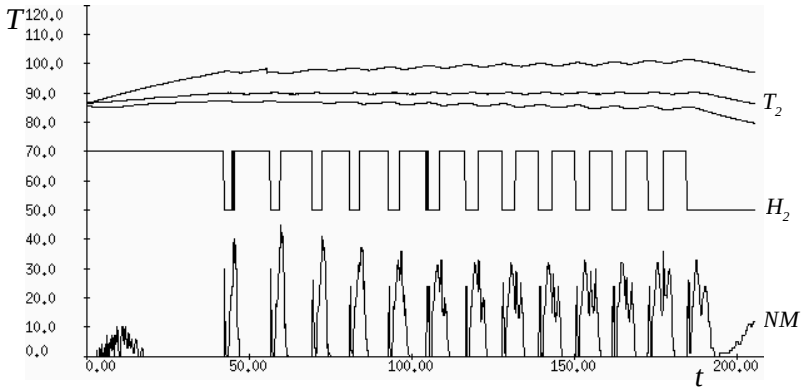


Figure 4. First overview of the monotonicity of the technical system. Drawn are the measured T_2 with its envelopes, H_2 is the heating flag for the second component, and NM is the number of the non-monotone subspace models. The discrete change of the input makes a directly effect to the monotonicity of the state values.

This approach can also be seen as *system identification*, because refuting subspace models reduces the uncertainty space, resulting in smaller bounding intervals on the parameters. Measurement noise can also be handled by introducing additional uncertainty parameters into the model.

However, this approach is in contrast to traditional system identification where the model space is specified by a parameterized differential equation. Identification selects numerical parameter values so that simulation of the model best matches the measurements. By using refutation instead of search our method is able to derive *guaranteed* bounds on the trajectories.

Model-based monitoring using uncertainty space partitioning is related to the interval identification algorithm of Schaich et al. [13]. In their approach the consistency check is only performed at the qualitative level. Thus, valuable detection time is lost, as long as the fault is only manifested in a quantitative value. Petridis an Kehagias [10] have also developed an algorithm with subspace partitioning. The partitioning is only performed in advance and the consistency check is based on probabilities depending on the noise in the system. Other work in monitoring [7, 9, 3] uses multiple models for fault detection. These models represent known faults of the supervised system. From the viewpoint of system identification, our approach is closely related to semi-quantitative system identification [8]. Identification of both approaches are grounded on the refutation of subspace models that are known to be inconsistent with the measurements. Semi-quantitative system identification performs refinement at the qualitative and interval level. Semi-quantitative system identification has also been applied to model-based monitoring [11]. Bonarini and Bontempi [4] have developed a quite similar approach to our consistency check. However, they have focused on uncertainty initial state values, which are given as intervals. Also related to our work is Armengol et al. [1, 2]. The simulation is based on modal interval arithmetics, which produces *overbounded* and *underbounded* envelopes of a technical system. To minimize the rate of false and missed alarms, the uncertainty space is only partitioned at critical measurements (which are between the underbounded and overbounded envelopes). In comparison to our approach, we simulate at each corner of the uncertainty space, which leads to exact envelopes (no false and missed alarms, according to observability) for linear systems.

Directions for future work include (i) the incorporation of (unknown) discontinuous transitions in our monitoring approach, (ii) further investigations on the monotonicity properties after a discontinuous transition, especially in the context of non-linear systems, and (iii) the improvement of the dynamic uncertainty space partitioning.

REFERENCES

[1] Joaquim Armengol, Louise Trave-Massuyes, Josep Vehi, and Josep Lluis de la Rosa, 'A Survey on Interval Model Simulators and their Properties related to Fault Detection', *Annual Reviews in Control*, **24**, 31–39, (2000).

[2] Joaquim Armengol, Josep Vehi, Louise Trave-Massuyes, and Miguel Angel Sainz, 'Application of Multiple Sliding Time Windows to Fault Detection Based on Interval Models', *12th International Workshop on Principles of Diagnosis (DX-01)*, 9–16, (2001).

[3] S. Bogh, 'Multiple Hypothesis-Testing Approach to FDI for the Industrial Actuator Benchmark', *Control Eng. Practice*, **3**(12), 1763–1768, (1995).

[4] Andrea Bonarini and Gianluca Bontempi, 'A Qualitative Simulation Approach for Fuzzy Dynamic Models', *ACM Transactions on Modeling and Computer Simulation*, **4**(4), 285–313, (1994).

[5] Jie Chen and Ron J. Patton, *Robust Model-Based Fault Diagnosis for Dynamic Systems*, Kluwer, 1999.

[6] *Readings in Model-based Diagnosis*, eds., Walter Hamscher, Luca Console, and Johan de Kleer, Morgan Kaufmann, 1992.

[7] Peter D. Hanlon and Peter S. Maybeck, 'Multiple-Model Adaptive Estimation using a Residual Correlation Kalman Filter Bank', *IEEE Transactions on Aerospace and Electronic Systems*, **36**(2), 393–406, (2000).

[8] Herbert Kay, Bernhard Rinner, and Benjamin Kuipers, 'Semi-Quantitative System Identification', *Artificial Intelligence*, **119**(1-2), 103–140, (May 2000).

[9] Raman Mehra, Constantino Rago, and Sanjeev Seereeram, 'Autonomous Failure Detection, Identification and Fault-tolerant Estimation with Aerospace Applications', *Proceedings of the IEEE Aerospace Applications Conference*, **2**, 133–138, (1998).

[10] V. Petridis and Ath. Kehagias, 'A Multi-model Algorithm for Parameter Estimation of Time-varying Nonlinear Systems', *Automatica*, **34**(4), 469–475, (1998).

[11] Bernhard Rinner and Benjamin Kuipers, 'Monitoring Piecewise Continuous Behaviors by Refining Semi-Quantitative Trackers', in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 1080–1086, Stockholm, Sweden, (August 1999). Morgan Kaufmann.

[12] Bernhard Rinner and Ulrich Weiss, 'Model-based Monitoring using Uncertainty Space Partitioning', in *Proceedings of the 21st IASTED International Conference on Modelling, Identification and Control (MIC 2002)*, (February 2002).

[13] David Schaich, Rudibert King, Uwe Keller, and Mike Chantler, 'Interval Identification - a Modelling and Design Technique for Dynamic Systems', in *QR99, 13^{th} International Workshop on Qualitative Reasoning*, (June 6-9 1999).

Object-Oriented Dynamic Bayesian Network-Templates for Modelling Mechatronic Systems

Harald Renninger and **Hermann von Hasseln**

DaimlerChrysler AG Research and Technology (REM/E)
D-70546 Stuttgart, Germany
{harald.renninger, hermann.v.hasseln}@daimlerchrysler.com

Abstract

The object-oriented paradigma is a new but proven technology for modelling mechatronics, i.e. multidisciplinary modelling. For many reasons the object-oriented approach is very much desirable also for qualitative models in system design, diagnosis or verification. Bayesian networks are a very robust technology for qualitative probabilistic modelling. In this paper we present a first approach in using the Bayesian networks modelling technique with the quantitative object-oriented method. Analogous to Modelica, an object-oriented modelling language, we constructed a Bayesian network library for modelling hydraulic systems. These Bayesian networks are called Object Oriented Dynamic Bayes Nets (OODBNs). Our method is easily transferable to any other physical domain or logic. In this contribution our motivation and the construction steps are described. Simulation results for a sample hydraulic system are given.

Introduction

Future system architectures will be characterized by highly modular and reusable components, and by abstract description languages widely independent of implementation details. Typical components of system architectures are software and hardware (sub-)systems. On the Software side the object-oriented paradigma is by now (at least in industrial applications) the *de facto* description or modelling language standard, mostly represented by the Unified Modelling Language (UML). On the Hardware side, which is our focus here, we have mechatronic hardware components, the constituent parts of which are control logics and controlled physical or chemical systems. Modelling mechatronic systems challenges the engineer due to different physical domains. In order to reach the goal of a truly unified description of system architectures comprising Software and Hardware systems, the description or modelling languages of mechatronic systems have to be lifted to a similar abstract level as their Software counterparts.

Model based techniques play an important role in concurrent and future engineering processes. Models and simulations are a basis for system design and analysis, e.g. for geometric layout of hydraulic systems. On the other hand, model based control and model based diagnosis are state of the art.

Many different philosophies have been developed to support the modelling task. In the control engineering area tools like Matlab/Simulink [1] or MatrixX SystemBuild [2] are widespread. For modelling mechanical systems ADAMS [3] or SIMPACK [4] are frequently used. For electronic systems PSPICE [5] is an appropriate tool. Other specific tools are used to solve modelling tasks in flow dynamics, thermal flow or chemical processes. Each of these programs are specially tailored for the specific domain.

A mechatronic system consists of a control logic, electronics and a controlled mechanical, hydraulic or any other physical or chemical system. The entire system is composed of subsystems of different domains. This shows the restriction of all classical modelling systems, since the control part can be easily described for example in Matlab/Simulink, but it is nearly impossible to model an electrical subsystem. So, a method is needed for a multidisciplinary modelling.

Methods and tools, e.g. Omola, Dymola or Smile, have been developed which allow multidisciplinary modelling. Modelica [6], [7] is the latest step in this direction. It is a standardized object-oriented modelling language which is supported by the tool Dymola [8] for example.

Dymola/Modelica comes with libraries for different physical domains like electrics/electronics, mechanics, thermal flow or hydraulics, see Figure 1. It also contains a signal block and a Petri net library. A library consists of a set of templates for different physical or logical objects. The user can extend a library for example by inheritance or can create completely new libraries. A model is described by an object diagramm. Most tools contain a graphical interface with a simple drag and drop technique for the templates and interconnections at the object interfaces. The interconnections have the meaning of constraints. More precisely, two types of equations are generated when two physical objects are connected: a flow and a potential equation. With the definition of the flow and the potential variables, the *energy flow* in the interface is uniquely defined. This is valid for all lumped parameter systems. The great advantage is that the system can now be modelled by *local* behaviour and not by global analysis [9], which supports the general idea of modularity.

Qualitative Models and Bayesian Networks

Qualitative modelling offers many well-known advantages for system design, diagnosis or verification, see [14] for a very extensive survey of techniques and applications. Some of these advantages are:

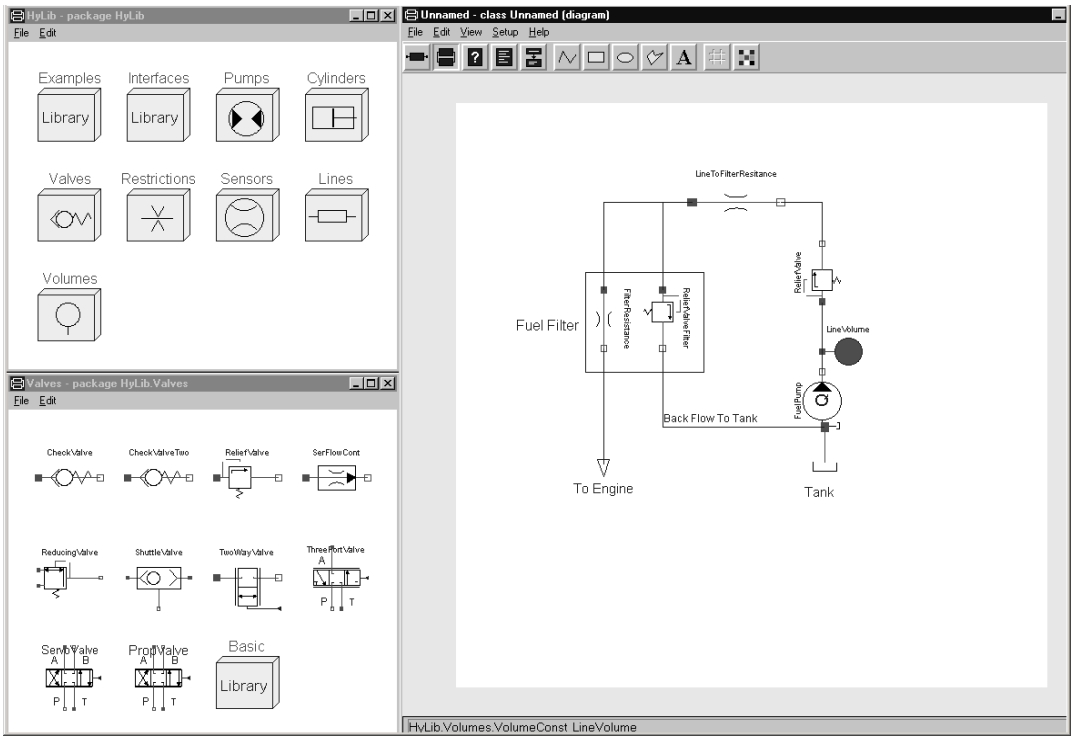


Figure 1: The hydraulics library and an object diagram in Dymola.

- handling of incomplete and imprecise knowledge,
- robustness,
- easy comparison of system alternatives, e.g. parameters variations,
- direct interpretation of simulation results,
- complexity.

Our vision is an object-oriented method using Bayesian networks for modelling physical systems, especially system dynamics. Bayesian networks are a well-suited method for handling imprecise knowledge in a consistent way. Efficient learning and adaption algorithms are known for Bayesian networks, which is a very interesting option for automatic model calibration. The definition of a Bayesian network *BN* is as follows: BN = {DAG, CPDs}, where DAG is a directed acyclic graph, consisting of nodes and directed edges or links, and CPDs are conditional probability distributions. The nodes in a Bayesian network represent propositional variables of interest (e.g., the temperature of a device). The links of a *BN* represent informational or causal dependencies among the variables. These dependencies are quantified by conditional probabilties (the CPDs) for each node given its parental nodes in the DAG. We do not cite the Bayesian networks fundamentals in this paper, but refer to the relevant literature, see [10] for some Bayesian networks basics or [11] for an excellent textbook.

Object-oriented Bayesian networks were introduced in [13], and are now supported by the newest version of the commercial Software tool *HUGIN* [12] for example.

Template construction

In this section we describe the conversion steps from Modelica to Bayesian network templates. The conversion will proceed in four major steps. First, given a dynamic component, the differential equations will be discretized in time using Euler's rule. Second, the equation part of a Modelica template will be reformulated with qualitative operators. Third, the qualitativ landmarks have to be chosen for each state variable and each parameter. Fourth, the resulting qualitative equations will be graphically programmed with Bayesian networks.

An fuel reservoir called "VolumeConst" will serve as an example. The icon used in the Modelica HyLib library [15] is shown in Figure 2. Note, that the component VolumeConst has one port (portA) and that the flow into the component has a positive sign . PortA can be viewed as a real physical flange with some pressure p and an oil flow q. The behavior of the component VolumeConst is described in Modelica by the equation block. Other definition blocks like the graphical, interfaces or parameter block are omitted.

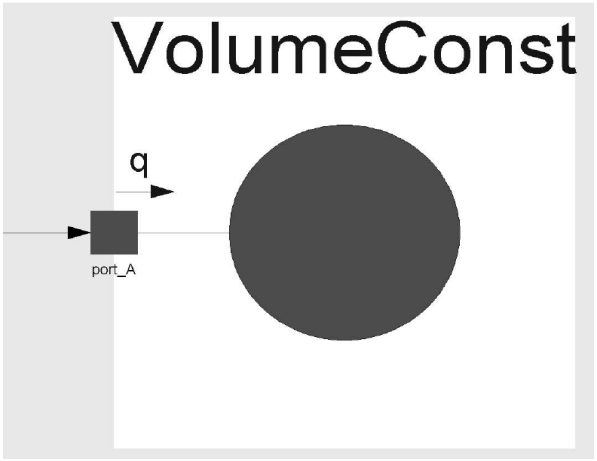


Figure 2: The Dymola representation for a fuel reservoir.

```
model VolumeConst
    graphical block
    interfaces block
    parameter block
    equation
    der( portA.p)  =  beta/volume * portA.q;
end VolumeConst
```

The equation block consists of one differential equation with *beta* and *volume* being fixed parameters defining the effective bulk modulus of the liquid and the volume in square meters, respectively. After chosing a time step *h* the time discrete version is as follows:

```
model VolumeConstDiscr
     equation
    1\h  ( portA.p(t) – portA.p(t-h) ) =
   (beta/volume * portA.q(t) )
end VolumeConstDiscr
```

Next, qualitative operators are inserted.

```
model VolumeConstQual
     equation
     portA.p(t) ⊕ -portA.p(t-h) =
    const ⊗ portA.q(t)
end VolumeConstQual
```

Now we have to choose a quantity space, the "landmarks", for the variables and parameters. For clearness, we choose a three valued quantity space $x \in \{-, 0, +\}$ for all variables x. Some qualitative calculus has to be defined for the chosen quantity space. Qualitative addition $\oplus$ for the three valued quantity space can be defined straightforward [14] as in Table 1.

$\oplus$	$x = -$	$x = 0$	$x = +$
$y = -$	$z = -$	$z = -$	$z = ?$
$y = 0$	$z = -$	$z = 0$	$z = +$
$y = +$	$z = ?$	$z = +$	$z = +$

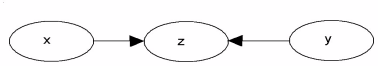
Table 1: Qualitative addition defined.

The $z = ?$ entry marks the ambiguity of the result, when the $\oplus$ operator is applied on $x = -$ and $y = +$ or vice versa, respectively.

Now the Bayesian network template for VolumeConst can be constructed. The basic idea is to identify each qualitative variable with a Bayesian network node, the qualititative values with the states of this node, and the qualitative calculus with CPDs.

We give an example for the $\oplus$ operator applied on variables x and y. The principle Bayesian network is shown in Figure 3. The entries in the CPD table in Figure 3 are probabilities, where each column sums to 1. The $z = ?$ entries in the operator table can be represented by the colomns with the uniform distribution, i.e. 1/3 for each entry in this case.

Any other algebraic operation can also be reformulated as a Bayesian network fragment. In this way the complete template for VolumeConst is constructed. The result is shown in Figure 4. The port nodes, which correspond to the port variables in portA are marked with a rectangle. When the Bayesian network template is instantiated in a system model only the input and output nodes, i.e. the port nodes, are visible. Note that this is a dynamic Bayesian network, because node PA0 carries the state of the pressure at time slice *t-h* and PA1 the state at time slice *t*. The difference is calculated in node dP01.



x	+			0			-		
y	+	0	-	+	0	-	+	0	-
z=+	1	1	1/3	1	0	0	1/3	0	0
z=0	0	0	1/3	0	1	0	1/3	0	0
z=-	0	0	1/3	0	0	1	1/3	1	1

Figure 3: Bayesian network fragment and the CPD for the $\oplus$ operator.

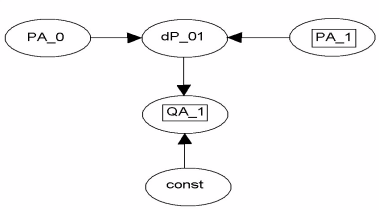
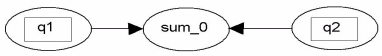


Figure 4: Bayesian network template for VolumeConst.

What is missing yet are the constraint templates, which serve as connectors between components. We need two different templates, one expressing that there is equal pressure at two connected ports, and a second one, expressing that the flows sum up to zero at a hydraulic node. We present these two templates with the CPDs in Figure 5 and Figure 6, respectively. For the pressure we assume three values: zero pressure (0), low pressure (+), high pressure (++). Note, that the arcs are directed to the "inner" constraint node, such that the resulting Bayesian network model is always acyclic. In



q1	+			0			-		
q2	+	0	-	+	0	-	+	0	-
sum_0=true	0	0	1	0	1	0	1	0	0
sum_0=false	1	1	0	1	0	1	0	1	1

Figure 5: Bayesian network template ZeroSumFlows2 for the flow constraint.

p1	0			+			++		
p2	0	+	++	0	+	++	0	+	++
eq_p=true	1	0	0	0	1	0	0	0	1
eq_p=false	0	1	1	1	0	1	1	1	0

Figure 6: Bayesian network template EqPressure2 for the pressure constraint.

Figure 5 and Figure 6 we show the simplest scenario that two components are connected in series. In Figure 1, on the right hand side, this is the case for the "ReliefValve" and the "LineToFilterResistance". In the Bayesian network for this tank system, the ReliefValve-template and the LineToFilter-Resistance-template will be connected via the flow constraint and the pressure constraint, see also Figure 7. Note that the table entries are hard 0/1 decisions. Before propagating the Bayesian network, the "true"-state of the sum_0 and the eq_p nodes must always be set evident. Doing this, the pressures on both sides are forced to be equal. The flow constraint then simply states, that the mass flow coming out of the first component equals the mass flow into the second component. In the general case, where more than two components meet, for example the "LineToFilterResistance", the "FilterResistance" and the "ReliefValveFilter", the flow constraint template must be assembled from the $\oplus$ -operation fragment, see Figure 3, and the flow constraint template of Figure 5. This new object is then called ZeroSumFlows3 and is shown in Figure 7.

Results

A basic library for constructing simple hydraulic circuits has been developed. It contains an ideal flow source, a reservoir, a hydraulic resistance, a tank, a relief valve, a real flow source, and the constraint templates. Differing from the previously discussed three state nodes, each dynamic variable here has five states. We used the object-oriented Bayesian network software Hugin. Currently, only discrete valued nodes are used. This is reasonable, because many mechatronic systems are hybrid or switching. Discrete valued nodes allow us to model arbitray dynamics, whereas continuous valued node models result in Kalman models, thus linear models.

We will shortly discuss the hydraulic library. The ideal flow source has two ports, namely A and B, or a "positive" and a "negative" port, with only one flow variable which can be controlled, i.e. set evident. A real flow source called RealFuelPump is derived from this ideal flow source. Additionally it contains the volume model, which was described above. So the port B of RealFlowPump delivers a pump flow and a pump pressure. The tank model has only a flow variable at the ports A and B. It is dynamic, modelling the change of the fuel volume over time.

The relief valve has a switching behaviour in Modelica. Pressure and flow is specified at the ports. In Modelica the valve logic is modelled with a state machine. We modelled this valve logic with a Markov model, having the two states "open" and "closed". At last, the hydraulic resistance has two ports specifying pressure and flow. It models laminar flow, i.e. the pressure drop over a hydraulic line.We used this element also to model the resistance of the fuel filter.

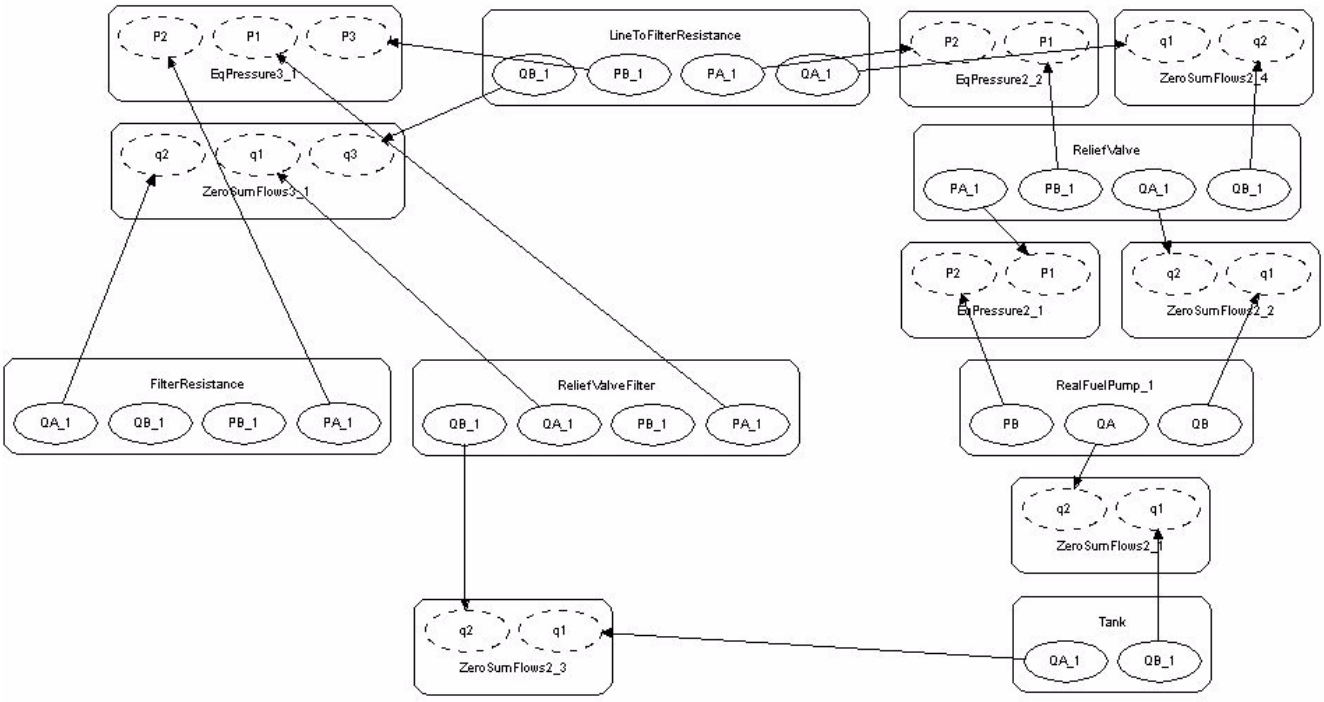


Figure 7: The Bayesian network tank system model.

We present a little hydraulic circuit, that is, a fictitious tank system. This system was first modelled for reference in Dymola/Modelica, see Figure 1 on the right hand side. Then we built this system using the dynamic Bayesian network templates. The Bayesian network tank system is shown in Figure 7. For the dynamic simulation, we set evident all constraint nodes and the pump flow. All other nodes are hidden, that is, they were calculated by propagation. The results for 100 time steps are shown in Figure 8, Figure 9, and Figure 10. These figures show the evolution of the probability distributions. The darker the colour bars are, the higher the probability. We added mean values for conveniance. The plots were produced using the Qualitative Modelling Toolbox for Matlab SIMULINK [16].

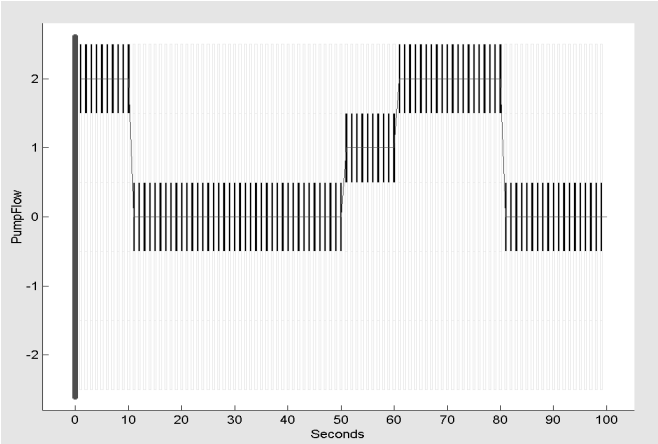


Figure 8: The pump flow (evident node).

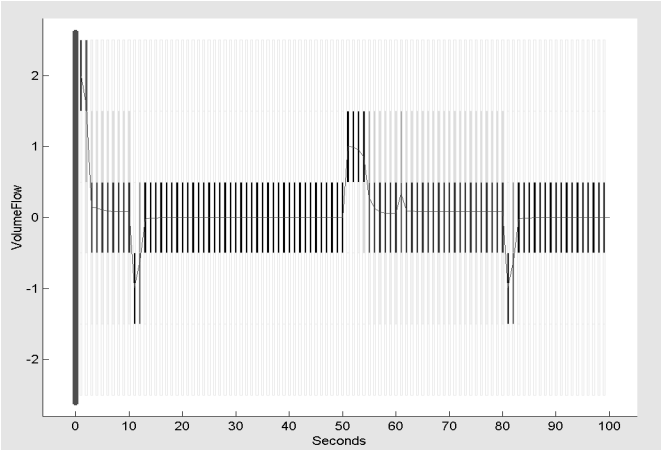


Figure 9: The flow into the fuel reservoir VolumeConst (inside the RealFuelPump template - hidden node).

Conclusion and future work

In this contribution we have motivated the need for intelligent modelling techniques. For system design, diagnosis or verification qualitative models are a very good choice. We favor the Bayesian network technology due to their robustness, intuitivity and practicability.

We seeked a qualitative modelling technique for mechatronic systems, i.e. for dynamic, multidomain systems. The object-oriented physical modelling technique gave us the hint for the construction of our OODBNs. The simulation results encourage us to proceed in this direction. Recent success has been made to select the states (the "landmarks") upon measurements or quantitative simulations, using a simple heuristic from system identification. Furthermore, learning respectively adapting the CPDs using HUGINs adaption API was very promising.

References

[1] SIMULINK. Homepage: http://www.Mathworks.com/

[2] SYSTEMBUILD. Homepage: http://www.isi.com/

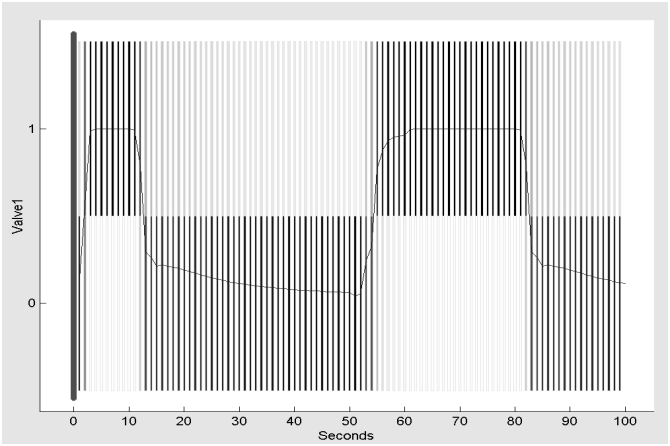


Figure 10: The ReliefValve behavior (hidden node), where ’0’ means closed, and ’1’ means open.

Products/MATRIXx/

[3] ADAMS. Homepage: http://www.adams.com/

[4] SIMPACK. Homepage: http://www.simpack.de/

[5] PSPICE. Homepage: http://www.microsim.som/

[6] Elmqvist, H.; Mattsson, S.E.; Otter, M.1999. Modelica - A Language for Physical System Modeling, Visualization and Interaction. IEEE Symposium on Computer-Aided System Design, CACSD’99, Hawaii, August 22-27, 1999.

[7] Modelica. Homepage http://www.modelica.org

[8] Dymola. Homepage http://www.dynasim.se

[9] Otter, M. et al. 1999. Objektorientierte Modellierung Physikalischer Systeme, Teil 1. at - Automatisierungstechnik 47, pp.A1-A4. Continued in the following issues of the at.

[10] Jensen, F. 1996. Bayesian networks basics. AISB Quarterly 94, pp. 9-22.

[11] Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J. 1999. Probabilistic Networks and Expert Systems. Statistics for Engineering and Information Science. Jordan, Lauritzen, Lawless, Nair (eds.), Springer-Verlag New York Inc. ISBN 0-387-98767-3.

[12] Hugin. Homepage: http://www.hugin.com/

[13] Koller, D., Pfeffer, A. 1997. Object-oriented bayesian networks, Proceedings of the thirteenth conference on Uncertainty in Artificial Intelligence, UAI 97.

[14] Dague, P.et al. 1995. Qualitative Reasoning: A Survey of Techniques and Applications.AICOM Vol. 8, Nrs. 3/4, Sept./Dec. pp. 119--192.

[15] Beater, P. 1998. Object-oriented Modeling and Simulation of Hydraulic Drives. Simulation News Europe, SNE 22, pp. 11-13.

[16] G. Lichtenberg et.al. Qualitative Modelling Toolbox User’s Guide. Release 5.2. Download: http://pcweb.rts.tu-harburg.de/inter/homepage.nsf

Development Tool for Distributed Monitoring and Diagnosis Systems

M. Albert, T. Längle, H. Wörn
Institute for Process Control and Robotics
University of Karlsruhe
D-76128 Karlsruhe
Germany

Abstract

This paper introduces a concept for building up distributed monitoring and diagnostic systems for complex industrial applications. The diagnostic process, from accessing sensor data up to the visualization within a graphical user interface is described by universal applicable formalisms. Generic mechanisms were identified to improve the quality of a diagnosis by integrating legacy diagnostic engines and handling different diagnostic mechanisms in parallel. For this purpose, a modular multi-agent architecture and a set of development tools were implemented. This software architecture for monitoring and diagnosis was developed within the framework of the EU Esprit Program: "**DIAMOND**: **DI**stributed **A**rchitecture for **MON**itoring and **D**iagnosis".

1 Introduction

To compete within industry, manufacturers are demanded to optimize their productive processes. In order to achieve this efficiency, a high value has to be set on the quality of the industrial equipment as well as on the industrial process itself. Monitoring and diagnostic systems (M&D systems) support this objective by predicting failures, or if a failure occurred, by identifying the reason for this fault. Thereby it is possible to reduce down-time costs of the production process. To achieve an overall reduction of the production costs, the development expense for a powerful M&D system has to be minimized. For this purpose, a modular concept was realized that is based on a distributed multi-agent approach.

A complex industrial application is built by a set of different physical units. These units may be provided by different vendors, having detailed knowledge about the behavior of their unit. Furthermore, there are often different diagnostic methods for the same unit available. To realize a powerful diagnosis, the knowledge about the different physical units and about various diagnostic methods should be merged together within an overall framework. Therefore, new strategies were required to treat these supplementing diagnostic knowledge about an industrial process.

This paper presents the generic aspects of the underlying infrastructure and describes the multi-agent framework. It is neither deemed to explain any diagnostic algorithms that may be applied nor to present the methodologies in detail that are employed to handle different diagnostic results in parallel.

2 Related work

Interest in recent research on distributed approaches for diagnostic purposes can currently be seen in Europe, Japan and in the United States. A general overview about distributed artificial intelligence in industry is given in [Par94]. This paper reviews the industrial needs for Distributed Artificial Intelligence, giving special attention to systems for manufacturing, scheduling and control. It gives case studies of several advanced research applications, actual industrial installations and identifies steps, need to be taken to deploy these technologies more broadly.

In [Fro96] there is a distinction between semantically distributed diagnosis and spatially distributed diagnosis. Semantically distributed diagnosis refers to a heterogeneous group of agents, in which each agent has its own view of the system. This can either mean that each agent focuses on a specific area of the system or that the agents model different aspects of the system or use different diagnostic methods, e.g. one agent models the structure of the system and another one models the performance. Spatially distributed diagnosis refers to a group of agents which jointly monitor and diagnose a spatially distributed system with relationships between those equipments. Each agent has detailed knowledge about a small part of the system.

Different concepts to realize a distributed approach are proposed in the literature, ranging from classical client server application over blackboard technologies to a few multi-agent frameworks. Most distributed applications employ a classical client server approach with distributed clients, communicating with a central server. This well known technology is continuously advanced and applied to distributed management applications. The standardi-

zation of distributed management tasks like information exchange, monitoring and diagnostics is aimed by the Common Diagnostic Model (CDM), developed within the "distributed management task force" (DMTF) [DMTF]. This framework is based on software clients which perform their tasks nearly automatically and report information to a central server.
Another concept is based on a blackboard technology. A central blackboard is used to store all available information about an industrial process. These data can be accessed by other software units, possibly software agents, in order to perform a diagnosis or to visualize the results. [Lee97] outlines the conceptual foundations for next generation industrial remote diagnostics and product monitoring systems. It extends the multi-agent framework research to include new classes of product population and diagnostic agents within a distributed Embedded Web and Electronic Commerce infrastructure. The products to be diagnosed are for example printers, copiers and vehicles.
[Ben97] introduces a KQML-CORBA (Knowledge Query and Manipulation Language) [KQML] based architecture to implement a multi-agent system for network and service management. The paper adopts this architecture and applies it to diagnosis and monitoring of machines and components in the production environment by using a multi-agent approach, combined with semantically distributed diagnosis. Up to now there are only few other implementations of KQML, one over TCP/IP in C which has been developed Lockheed-Martin [Fin99] and one in JAVA [Fro96]. The use of the agent communication language developed by FIPA [FIPA98] together with the CORBA standard for distributed computing is a widely used strategy to realize an agent interaction [Orf97].

3 Units of Monitoring and Diagnostic System

High value and cost effective diagnosis system can be developed by distinguishing between the tasks that have to be performed within a M&D system. Some tasks are general for all monitoring and diagnostic systems, some are specific for the employed production equipment and others are specific for the considered production process. These three units should to be treated in different ways:

3.1 Generic M&D units

Independent to specific requirements of an application area, a set of tasks are generic for all monitoring and diagnostic systems. Functionalities like interfacing different units, storing data, formatting measurements and diagnosis results, configuring the system, managing the interaction and some more have to be performed in all M&D systems. It is obviously feasible to reuse a set of existing software units whenever a specific Monitoring and Diagnostic system has to be built. This allows the integration of highly developed and well tested units extremely fast and low priced. Together with a set of well defined interfaces, a general architecture for monitoring and diagnosis becomes realized.

3.2 Specific to production equipment

Some parts of monitoring and diagnostic software are specific to the used production equipment. The way how to access sensor values and how to use these data to diagnose a physical component is specific to the production environment. The component manufacturer is interested to equip its component with monitoring and diagnostic capabilities that are compatible with a generic architecture. The usage of software agents allows to encapsulate legacy diagnostic tools in order to become interoperable to the overall system (see section 5). These parts are reusable whenever the same production equipment is used and should not be developed each time a M&D system is built from scratch.

3.3 Specific to production process

All remaining parts of the complete monitoring and diagnosis system are specific to the production process. The interaction between different physical components, and adoptions of the operator interface are examples for units that have to be develop individually.

4 DIAMOND – distributed multi-agent architecture

The architecture, proposed in this paper, tries to merge the three different parts to a consistent monitoring and diagnosis system. All generic units of a M&D system are realized by the utilization of software agents, each responsible for a specific task. An integration of all parts that are specific to the production environment or to the production process itself is supported by encapsulating these functionalities into different agents, able to interact with the overall framework. The DIAMOND architecture specifies the information that are required to integrate these parts and supports the development process by offering a set of tools (see section 5).
The structure and the interaction of the software agents that are able to perform all generic tasks and that are used to encapsulate all specific tasks are described in the following chapter.

4.1 Structure of DIAMOND Agent

All agents that are used within DIAMOND are built of two different software units. One is responsible for the communication with other DIAMOND agents within the M&D framework. This unit is called 'Wrapper' and is identical for all agents. The second software unit is called 'Brain'. This part performs all tasks that are specific for the type of the agent.
The Wrapper is responsible for handling the communication of this agent with other agents in the M&D system by applying the CORBA middleware. Information can be exchanged by applying CORBA events or by using

CORBA call-backs. The wrapper registers its CORBA objects at the ORB to become visible for other agents. However, the wrapper does not initiate or analyze any information exchange with another agent by itself. This is handled by the brain part.
The agents brain performs all tasks that are specific to the type of the agent. Some of the agents that are utilized within the framework perform exclusively generic tasks that are similar for all industrial environments. The implementation of the brain for these agents is uniform for each application where DIAMOND is applied. The tasks that have to be performed by the monitoring and diagnostic agents (see next chapter) are partly independent on the application. One part handles generic monitoring or diagnostic capabilities. This part of the monitoring agent brain and the diagnostic agent brain is provided by the DIAMOND development toolkit (see section 5.4 and section 5.5). All further capabilities depend on the specific application and have to be implemented afterwards individually by accessing well defined interfaces. This absence of an explicit implementation enables the integration of legacy monitoring and diagnostic tools inside the brain of a monitoring agent or inside a diagnostic agent.

4.2 Agent interaction

The Wrapper of each agent uses CORBA to exchange messages with other agents. It supports a synchronous call-back communication and an asynchronous event-based communication.
The wrapper exposes a CORBA remote interface to other agents that can use it, whenever they want to send a message to this agent by using a CORBA call-back. The cure, if an unknown protocol was received, if the agent is to busy to handle the message or if the agent will continue to process the message. Only in the last case, the message will be stored in an internal buffer that is part of the Wrapper. This buffer allows to sort incoming (and outgoing) messages according to their priority, their timestamp or in respect to a given time-out. The brain removes the message from the buffer as soon as it is able to process it. After processing the message, it specifies the answer, corresponding to the used protocol. This answer is stored in the internal buffer of the agent and forwarded to the remote CORBA object.
The message that is sent by an agent contains a set of pre-defined parameters as it is specified by FIPA. These parameters are stored within a XML structure. Since a conversation must not only handle information exchange but also the exchange of attitude about the information, a 2-layered protocol is applied. The outer layer of a message represents the attitude about the information. These data are processed by the Wrapper. The information itself is part of an inner layer which is stored in the content parameter of a message. The message content, which is also encoded in the XML syntax, is processed by the brain of the agent.
If an agent wants to supply data quickly to the overall multi-agent framework without taking care about the receiving agents, an asynchronous event-based communication is more feasible. This mechanism is mainly used by the monitoring agents to supply measurements to all diagnostic agents that are interested in. Every agent is able to supply and to consume events, structured in XML, by connecting to different event channels. This CORBA functionality is accessed by the Wrapper.

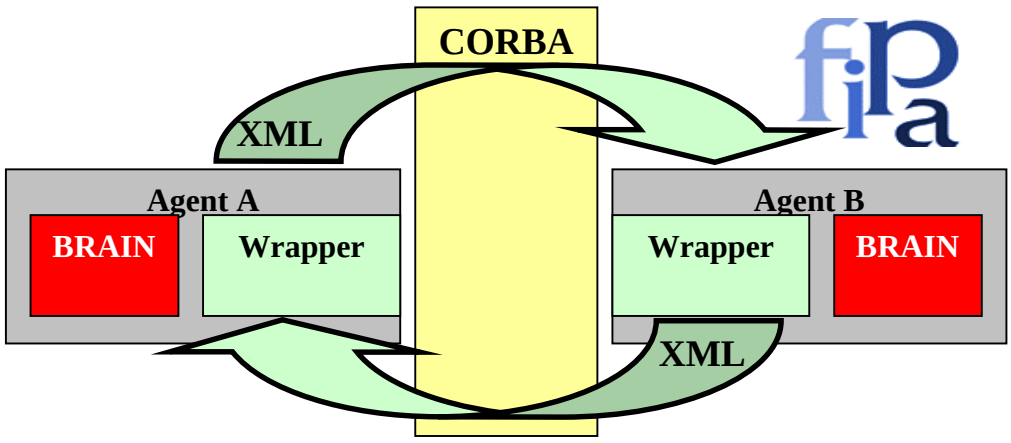


Figure 1 Agent Interaction

FIPA-Agent Communication Language (FIPA-ACL) [FIPA98] is used to restrict the interaction between communicating agents (see figure 1).The REQUEST, the QUERY-REF the SUBSCRIBE and the CANCEL protocol were identified to cover all required agent interactions. The call-back CORBA concept allows the receiving agent to return an integer value instantly if the structure of the message is invalid, if the message is not se-

4.3 Monitoring Agent

The interface between the physical state of the industrial application and the DIAMOND system is realized by a Monitoring Agent. This type of agent handles the measurements of the physical components and prepares them to be treatable by other agents within the framework. Each Monitoring Agent has to be adjusted to the sensors

of the industrial equipment that will provide the measurements. Furthermore, the Monitoring Agents are able to initiate a diagnosis of a component as soon as they have identified an irregular state of a measurement.

4.4 Diagnostic Agent

Different aspects of distribution are handled within the DIAMOND framework. First of all, the different tasks that have to be performed within the monitoring and diagnostic process are distributed to different agent types, each responsible for its specific task. The task that has to be performed by the Diagnostic Agents is also distributed again. The Diagnostic Agents are handling the measurements that are provided by the Monitoring Agents to identify the functional state of the physical components. This diagnosis may be performed by different Diagnostic Agents, each having a different view of the industrial application.

- This variation may be related with different temporal aspects of the behavior of the plant (temporal distribution).
- Often, there are different diagnostic algorithms available to identify the state of an industrial process. A development tool for a flexible M&D system has to be able to handle various diagnostic mechanisms in parallel. This is identified as a semantical distribution of the diagnosis.
- The entire diagnostic knowledge about the behavior of the plant is split to a set of smaller knowledge units, each associated with a physical part of the plant, called component. A single Diagnostic Agent does not know about the behavior of the complete plant, but about a single component. This knowledge may be provided by the manufacturer of the component. In this manner, the diagnostic task is spatially distributed.

When distributing the overall diagnostic task regarding temporal, semantical and spatial aspects, a flexible and clear framework is feasible. For diagnosing the overall process, the various diagnostic results, reported by different Diagnostic Agents have to be merged together. This additional task is performed by the Conflict Resolution Agent.

4.5 Conflict Resolution Agent

A conflict resolution mechanism is required to investigate, whether the diagnostic results, reported by different Diagnostic Agents are contradicting or completing each other. The Diagnostic Agents do not communicate with each other to merge their knowledge, but do report their diagnosis to a Conflict Resolution Agent. According to the different types of distribution, temporal, semantical and spatial conflicts have to be considered. For this purpose, the relations between the components and between the possible failures which may be related within the components have to be well known (section 5.1 and 5.3). The knowledge is represented by a Graph. An adjacent vector, where each element represents a component is used to build the graph [ALG94]. Each node (component) consists of:

- Vector of topological arcs with other components
- Vector of relationships between the same failure in different components
- Vector of relationships between different failures in different components.

The overall conflict resolution process is divided into different sequential steps:

- The reported failures are assigned to the nodes (components) of the graph conformed by the information specified in the structural knowledge base (chapter 5.1). This allows to identify semantical conflicts.
- Following, spatial and temporal conflicts are investigated in three different levels. Level 1 works with the topological information specified in the structural knowledge base, level 2 works with the relations between the same failure in different components (i.e. similar to level 1 but specific for a failure) and level 3 works with the relations between different failures in different components.

Details about these generic algorithms for handling temporal, semantical and spatial conflicts can be found in [DIAMOND].

4.6 Facilitator Agent

The Facilitator Agent is responsible for networking and mediating between the agents in the Multi-Agent framework. Large industrial applications may be federal and hierarchical structured. This structure is adopted to different “domains”. A domain is a subsystem of the DIAMOND architecture that is responsible for a part of the industrial application. Each “domain” is associated with a facilitator agent to facilitate the networking within this domain and with other Facilitator agents of other domains. Thus a diagnosis of a single domain as well as a diagnosis of the complete industrial application is feasible. Furthermore, the Facilitator agent is the mediator to the Graphical User Interface Agent.

4.7 Blackboard Agent

All diagnosis results that were reported within a well defined timeframe are stored in a blackboard that is implemented in a Blackboard Agent. Each domain has its own Blackboard Agent that is mediating with the Conflict Resolution Agent. The Blackboard Agent provides the results, reported by the Diagnostic Agents and triggers the conflict resolution process. The resolved diagnostic result that cover the state of all components that are part of the domain are forwarded to the Facilitator Agent. The Blackboard Agent is also in charge of storing all reported diagnostic results permanently.

4.8 Graphical User Interface Agent

The Graphical User Interface Agent is the human gateway to the DIAMOND system. The operator uses this

interface to get information about the state of the industrial application, to provide human accessible information to the Diagnostic Agent and to initiate diagnostic processes.

4.9 Overall Architecture

The hierarchical and federal structure of the industrial environment that has to be monitored and diagnosed is transferred to a hierarchical and federal structure of the software architecture. For this purpose, industrial components are grouped together to form a logical superior unit. A set of agents that are responsible for diagnosing this set of components are grouped within a “domain”. Only the Facilitator Agent of each domain is able to communicate with other Facilitator of other domains or with the Graphical User Interface Agent.
The main concepts of this DIAMOND architecture are summarized in figure 2.

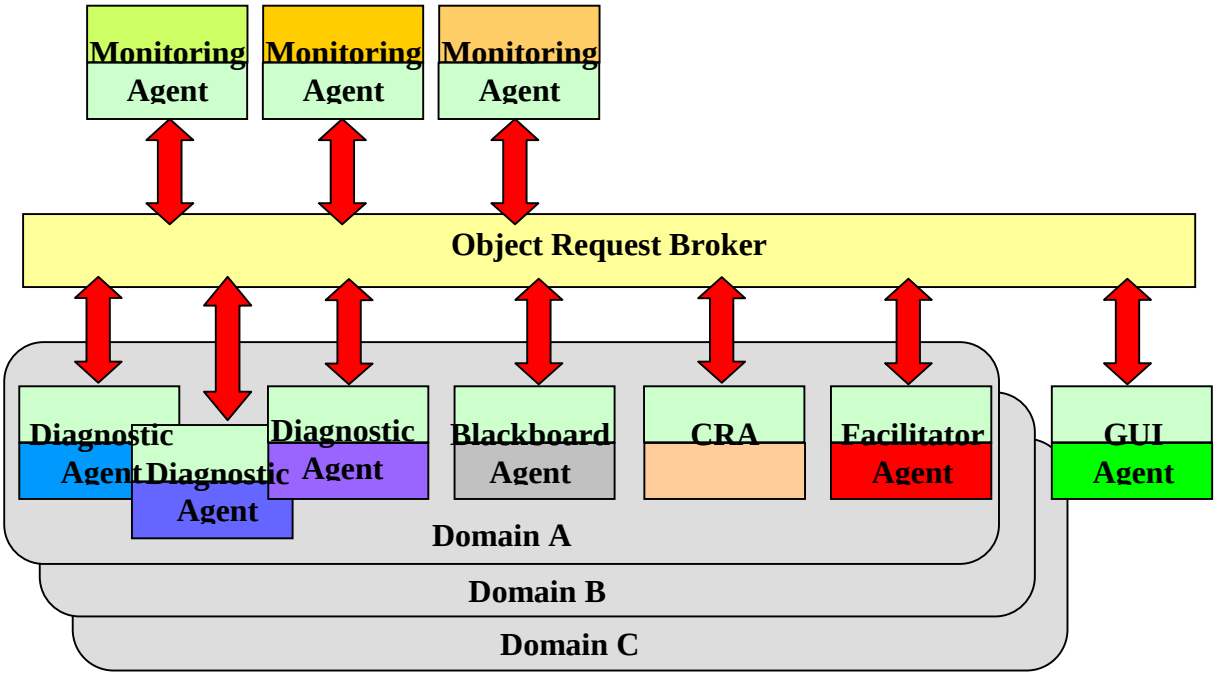


Figure 2 DIAMOND Architecture

All DIAMOND agents that are interacting are pictured as two colored boxes with the type of the agent written inside. The light green box indicates the Wrapper that is responsible for communication. This part is unique for all agents. The second box represents the Brain which is specific for the agents type. The agents are interacting by using the Object Request Broker (CORBA). This middleware is pictured as the yellow bar in the middle of the figure.
The figure indicates three different domains (Domain A, B and C). These domains are grouping the Diagnostic Agents, a Blackboard Agent, a Conflict Resolution Agent and a Facilitator Agent together. The Monitoring Agents are not associated to one single domain. They are able to provide measurements that may be used by the Diagnostic Agents of different domains.

5 How to build a monitoring and diagnosis system

This chapter describes the steps that are required to build up a complete monitoring and diagnosis system by using the results provided by the DIAMOND architecture.

5.1 Identify semantic structure of the industrial application

The first step while building up a monitoring and diagnosis system is to define all physical components and their relations of the automated industrial system that have to be supervised. There should be no overlap between components, nor should there be “white spaces” of the system being diagnosed not covered by a component at all.

The components of the industrial application may be hierarchical or federal related with each other. If there is a set of components that are building a logical unit which is widely self-contained, they have to be grouped together into a domain. The knowledge about the components and domains is fixed for a specific industrial application and will not change during runtime. DIAMOND provides an ontology that defines the structure and the possible attributes of any component. This knowledge is stored in the structural knowledge base in the XML format.
The possible relations between components are expressed by the attributes of each “COMPONENT” element:

- INPUT_CONNECTED_TO specifies functional or logical output of another component which effects the behavior of this component.
- EXCLUSIVE identifies that the faults of both related components are mutual to each other.
- BELONGING_TO is used to express the topological relation between "parent" and "child" components.

Further attributes are the certainty of the identified relation and a possible time delay which describes the temporal behavior of related components.

5.2 Identify measurements

It has to be investigated whether there are any existing monitoring sources available which are able to access measurable states of the plant. These sources may be accessed by a Monitoring Agent. It has to be investigated, which information about the measurable state of the industrial application is accessible and how to obtain them. In the case of integrating a legacy application for accessing the system variables, the interface to these applications have to be identified. All measurable states that are practical for a diagnosis have to be described as a measurement according to a well defined ontology for MEASUREMENT.

All measurements that will be used have to be associated with a Monitoring Agent that is able to access it. It is reasonable to associate all measurements that are provided by a single data acquisition source or by a specific mechanism how to access them with the same Monitoring Agent.

5.3 Identify failure modes

The M&D system is able to identify those potential faults of the industrial application that were specified in advance. Therefore, it has to be investigated, which legacy diagnostic tools may be used and which faults these modules are able to detect.

Attributes for each failure are specifying the conditions that have to be fulfilled, the names of rules that are feasible to identify the fault and potential recovery actions. Another attribute identifies whether the occurrence of this failure is related with another failure, either in the same or in another component. This allows to state whether different faults are contradicting or complementing each other, how they are temporally related and how a failure propagates.

All these information are stored within an XML structure. These data are mainly processed by the Conflict Resolution Agent to solve diagnostic conflicts. The diagnostic mechanisms and algorithms to identify the failure are specific to the industrial process and will be used by the diagnostic agents.

5.4 Build Monitoring Agents and connect with industrial environment

For building the Monitoring Agents, a shell is provided by DIAMOND that enables the creation of this agent. The connection with the monitoring and diagnostic infrastructure is automatically realized. The connection with the sensors of the industrial application has to be done afterwards. This task is specific for each application and for each sensor.

There are several predefined configuration parameters for a Monitoring Agent. These parameters may be set by using a DIAMOND toolkit.

5.5 Build Diagnostic Agents and connect with diagnostic engines

The measurements are used by the Diagnostic Agents to perform a diagnosis. For this purpose, each Diagnostic Agent needs to have a diagnostic engine. This may be a commercial expert system or any other kind of diagnostic engine to identify failures of the related component. The connection of the diagnostic engine with the M&D system is realized by using a development shell for creating the Diagnostic Agents. The interface to the diagnostic engine has to be implemented afterwards individually. There are only two methods that have to be implemented for interfacing:

One method enables the diagnostic engine to get a measurement value which is accessed from an internal buffer of the Diagnostic Agent. The engine does not keep care where to get the measurement from. All measurements that were identified in chapter 5.2 are accessible. After the engine has performed its diagnosis, it provides the diagnosis result to the Wrapper of the agent by using the second method of the interface. The Wrapper makes the result available for the infrastructure for further processing.

Integrating the diagnostic engine of a legacy diagnostic tool is possible, if this clear interface is realized. No further modifications, neither to the DIAMOND framework, nor to the diagnostic engine are required.

6 Evaluation Examples

The functionality of the presented multi-agent architecture was verified by integrating a specific monitoring and diagnosis system into two operational prototypes.

The first was concerned with the functional process of an automated welding cell, containing a control system, a robot with gas-metal arc welding equipment and a positioning device. To simulate faults that may occur in the welding system, a simulator was used that emulates the behavior of the welding equipment for different faulty situations. The measurements were accessed by using an ODBC interface and a DCOM interface. Several legacy case based reasoning engines, each responsible for another component, were applied to identify faulty components. This integration was suitable to present the capability to integrate different data accessing methods and various diagnostic engines within an integrative monitoring and diagnostic system easily. This M&D system was able to identify spatial conflicts and recognize the propagation of faults from one component to another one.

The second evaluation example took an existing expert system for diagnosing the water-steam cycle chemistry of a coal fired power plant (called SEQA, based on G2, Gensym) and re-worked it to operate in a modern diagnostic framework. To verify the behavior of the M&D system outside the power plant, a simulator based on a neural network model was used to either generate offline artificial anomalies overlapped to normal patterns or online to provide a set of normal behavior values against which measurements should be compared. The assimilation of a complete legacy expert system into a distributed M&D framework illustrated a complex tasks since there were many interfaces necessary for accessing data and for using the legacy diagnostic engines.

7 Conclusion

This paper describes a concept for building a distributed architecture for monitoring and diagnosing a complex industrial application. The presented M&D system uses a multi agent approach which warrants the flexibility, the extendibility and a cost effective development of the system.

One main extension to existing solutions is the possibility to integrate legacy diagnostic tools into the overall diagnosis system. This requires an extensive and exact specification of all components, measurements and possible failures of the industrial application as well as a specification of their relations to each other. This was realized by introducing a set of ontologies for the monitoring and diagnostic system.

Furthermore, several diagnostic engines can be utilized in parallel. They may refer to different components of the industrial application and they may apply different diagnostic mechanisms. By using different Diagnostic Agents, related to different components, the diagnostic knowledge can be provided by the component manufacturer. For applying different diagnostic methods, algorithms were developed to handle different diagnosis results in parallel and to investigate whether they are completing or contradicting each other.

Acknowledgment

This research work has been performed at the Institute for Process Control and Robotics, Prof. Dr.-Ing. H. Wörn and Prof. Dr.-Ing. R. Dillmann, Department of Computer Science, University of Karlsruhe (Germany) in collaboration with Union Fenosa Generación, S.A. (Spain), KUKA Roboter GmbH (Germany), GenRad Ltd (UK), Instituto de Investigación Tecnológica - Universidad Pontificia Comillas (Spain) and S.A.T.E. s.r.l.(Italy). The research was partly funded by the European Commission in the DIAMOND project under the Contract No. EP 28735.

References

[ALG94] "Introduction to Algorithms", T.H. Cormen, C.E. Leiserson, R.L. Rivest, the MIT Press, 1994.

[Ben97] Benech D., Desprats T., Raynaud Y., "A KQML-CORBA based Architecture for Intelligent Agents Communication in Co-operative Service and Network Management", IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Montréal, Canada (1997)

[CORBA] http://www.corba.org/ 2002 „Common Object Request Broker Architecture", object management group.

[DIAMOND] http://wwwipr.ira.uka.de/~kamara/diamond/ 2001 "Distributed Architecture for Monitoring and Diagnosis". EU Esprit Project No. 28735.

[DMTF] "Distributed Management Task Force", http://www.dmtf.org/ , develops standards for distributed management tasks, 2002

[Fin99] Yannis Labrou, Tim Finin, Yun Peng, University of Maryland, Baltimore County: "Agent Communication Languages: The Current Landscape". IEEE March/April (1999)

[FIPA98] http://www.fipa.org/ 1998 "FIPA 98 Specification", Foundation for intelligent Physical Agents (1997-2001).

[Fro96] Froehlich, P., Nejdl W., "Resolving Conflicts in Distributed Diagnosis", ECAI '96, 12th European Conference on Artificial Intelligence, John Wiley & Sons, Ltd., (1996)

[KQML] T. Finin, R. Fritzson, D. McKay, R. McEntire, "A language and protocol for Information Exchange", Technical Report CS-94-02, Computer Science Department, University of Maryland, UMBC

[Lee97] Lee B. H., "Agent-based Remote Diagnostics of Product Populations Across the Full Product Life Cycle An Industrial Multi-Agent System Approach in an Electronic Commerce Framework", Proceedings of The Seventh International Workshop on Principles of Diagnosis, October 13-16, 1996, Val Morin, Quebec, Canada (1997)

[Nej94] Nejdl W., Werner M., "Distributed Intelligent Agents for Control, Diagnosis and Repair", Technical Report, Informatik V, RWTH Aachen, Germany, (1994)

[Orf97] Orfali, R., Harkey D., "Client/Server Programming with Java and CORBA", John Wiley & Sons, Ltd., (1997)

[Par94] Parunak V., "Applications of Distributed Artificial intelligence in Industry", O'Hare and Jennings, eds., Foundations of Distributed Artificial Intelligence, Wiley Inter-Science, 1996, (1994)

[XML] 2001, „Extensible Markup Language", http://www.xml.org/

Using Supervised Learning Techniques for Diagnosis of Dynamic Systems

Pedro J. Abad[1], **Antonio J. Suárez**[1], **Rafael M. Gasca**[2], **Juan A. Ortega**[2]

Abstract. This paper describes an approach based on supervised learning techniques for the diagnosis of dynamic systems. The methodology can start with real system data or with a model of the dynamic system. In the second case, a set of simulations of the system is required to obtain the necessary data. In both cases, obtained data will be labelled according to the running conditions of the system at the gathering data time. Label indicates the running state of system: correct working or abnormal functioning of any system component. After being labelled, data will be treated to add additional information about the running of system. The final goal is to obtain a set of decision rules by applying a classification tool to the set of labelled and treated data. This way, any observation on the system will be classified according to those decision rules, having a return label indicating the currently running state of system. Returned label will be the diagnostic. This entire learning task is carried out off-line, before the diagnosing.

1 INTRODUCTION

Diagnosis determines why a system, correctly designed, doesn't work like it was expected. Explanation, for this erroneous behaviour, represents a discrepancy with the system design. One diagnosis task is to determine the system elements that could cause the erroneous behaviour according to the system observations. Monitoring process is fundamental to avoid non-real faults by small alterations in variables values. [1] Proposes a knowledge model for dynamic systems monitoring.

Fault detection consists on determining, starting from the system observations, when an incorrect operation of the observed system exists. When failure is detected then diagnosis will take the control to find the reasons of that incorrect behaviour.

Fault detection and diagnostic of faulty components are very important from the strategic point of view of the companies, due to the economic demands and environment conservation required to remain in competitive markets. This is one of the reasons causing that this is a very active investigation field. Components faults and process faults can cause systems damages and undesirable halt of the system. This causes the increase of costs and decrease of production. Therefore developing mechanisms to detect and to diagnose systems faults are needed to maintain the systems in levels of security, production and reliability.

Inside the Artificial Intelligent community the dynamic systems diagnosis task has been approached, in most of the cases, adapting the techniques coming from the static systems diagnosis to the dynamic behaviour of the systems. This way [2] or [3] try to add temporary information to GDE [4]

On the other hand, qualitative models have also been commonly used for this purpose [5] [6].

In [7] the fundaments of the based-models diagnosis, applied to the dynamic systems, are presented, and more recently [8] proposes a consistency-based approach with qualitative models.

Other techniques, coming from the AI, have also entered in the diagnosis field. Following this line, learning techniques tries to identify the system behaviour basing on a previous training.

Lately, some works using learning-based techniques have been presented, like stochastic methods [9], neural network based learning [10] and classification systems [11]. Neural network techniques have recently been applied in diverse fields, as medicine [12] or power supply [13].

Machine Learning techniques, inside the supervised learning field, are automated procedures based on logical operations that learn a task starting from a suite of examples. In the classification field the attention has been centred, concretely, in approaches with decision trees [14], where classification is the result of a series of logical steps. These approaches are able to represent the most complex problems if they have enough data. Applied to the diagnosis, we can find these methods used for the classification of temporary patterns [15] or in previous works to the current one [16] [17].

The present work is centred in quantitative models. It uses supervised learning techniques to obtain a rules-based model to diagnose dynamic systems by recognizing the correct behaviour models and faulty behaviour models. An approach to offer several fault causes, when there isn't an only clear cause, is presented.

Rest of the document has been organized in the following way: in the next section the used methodology will be exposed and the form to carry out the diagnosis. Next a problem application example is described for the developed approach. To illustrate the operation of these techniques a wide set of tests is presented. Lastly some improvements that are in development process are discussed.

2 PROPOSED METHODOLOGY

To carry out diagnosis of dynamic systems a set of decision rules should be generated. It can be done starting from the known

[1] Dpto de Ingeniería Electrónica, Sistemas Informáticos y Automática. Universidad de Huelva. E-Mail: {abadhe,asuarez@uhu.es}
[2] Dpto de Lenguaje y Sistemas Informáticos. Universidad de Sevilla. E- Mail: {gasca,ortega@lsi.us.es}

trajectories of the system or the simulations generated from a model.

Before starting with the methodology some concepts need to be defined.

2.1 Definitions and notation.

Definition 1: Behaviours Family. It is a finite group of trajectories having a similar behaviour from the point of view of the diagnosis.

Definition 2: Correct behaviour. It is the finite group of trajectories belonging to evolutions of the system without any fault type.

Definition 3: Perfect behaviour. It is the trajectory describing the system when all parameters take the central values of the ranges defined as correct.

Definition 4: Observation. It is a real trajectory of the dynamic system containing values of the observational variables in the system.

Definition 5: Diagnosis. It is the identification of the observed behaviour of the system as belonging to a certain behaviour family (diagnosis label) and according to decision rules.

Proposed approach can be generated from two different ways:

- Rules are generated starting from a group of different behaviour models.

  $\vee$ *Model (behaviour)* $\Rightarrow$ *labelled trajectories*

- Rules are generated starting from a group of experimental trajectories of dynamic system for the correct behaviour and possible fault behaviour.

  $\vee$ *Trajectories (behaviour)* $\Rightarrow$ *labelled trajectories.*

Leaving of one of these situations the process can continue like that:

1. Similar trajectories belonging to different behaviours family are identified. These trajectories are labelled again as belonging to both behaviours family.

   $\vee$ *Similar Trajectories (different behaviour family)* $\Rightarrow$ *relabelled trajectories.*

2. Decision rules are generated using a supervised learning tool.

   $\vee$ *Relabelled trajectories* $\Rightarrow$ *Decision rules*

3. Diagnosis consists in associating an observation as corresponding to behaviours family by using decision rules.

   Classification (observation, rules) $\Rightarrow$ *Diagnostic label*

2.2 Methodology

Proposed methodology to diagnose is an amplification of other one developed in [16]. This basic methodology may present some problems when the same system behaviours can be associated to different fault reasons. In order to don't diagnose incorrectly these cases, in this new approach, those behaviours will be associated with all the possible behaviours family that can cause this concrete behaviour. In this way several fault causes will be offered for observations that can correspond to different behaviours family.

Basic idea consists in obtaining a set of classification rules from a suite of system data in different behaviours modes: the correct behaviour and the faulty behaviours. After, those obtained classification rules can be used to associate an observation with model behaviour. Thus diagnosis of the observation is obtained.

Process can start with real system data or with a model of the dynamic system. In the second case, a set of simulations of the system is required to obtain the necessary data. In both cases, obtained data will be labelled according to the running conditions of the system at the gathering data time. Label indicates the running system state: correct working or abnormal function of any system component. Final result consists in a database containing all labelled trajectories.

Obtained database contains very similar trajectories corresponding to different behaviour family and therefore with different labels. To solve this problem the set of all similar trajectories will be relabelled with new labels. This new labels will be composed as a mix of the older labels. Thus, relabelled trajectories will be associated with anyone of the original behaviours family. The problem is to define when two or more trajectories are similar. Decision taken is that several trajectories

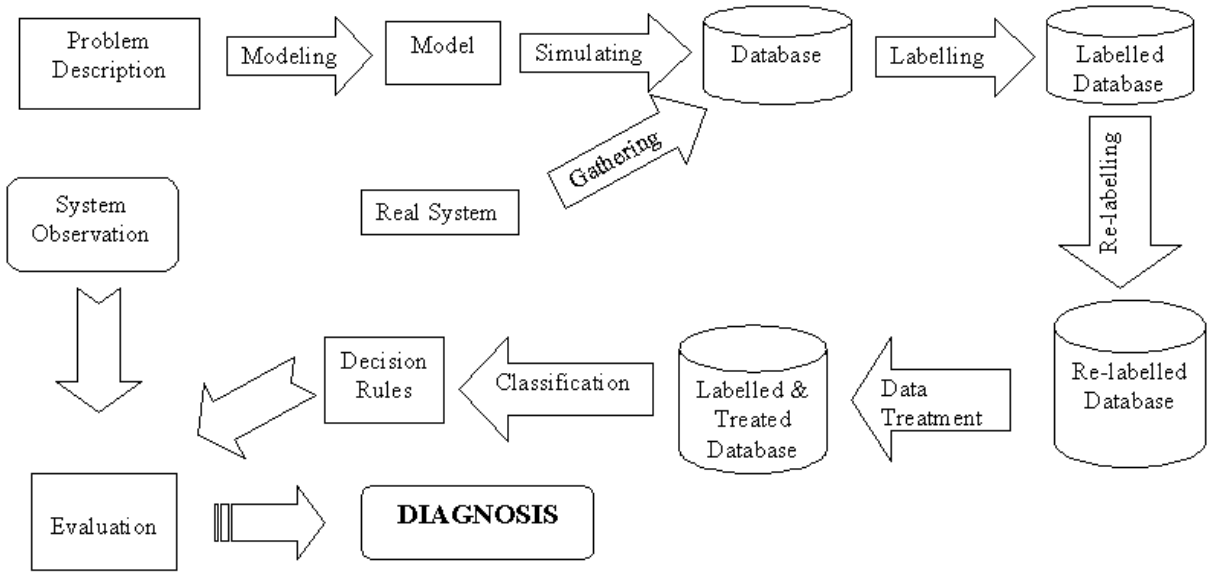


Figure 1. Proposed Methodology

are similar when distance between them is lower than a magnitude. That magnitude should be specified for each treated system. Used distance is Euclidean distance.

After being labelled and relabelled, trajectories data will be treated to add additional information about running of the system. This additional information will be very useful when classification tool tries to find decision rules, because available information will be greater. This additional information should characterize the system further than gathering data and it is specified for each treated systems.

A new database, which contains original trajectories plus new attributes and the corresponding label, is obtained.

Final step, to obtain decision rules, is to use a classification tool with the labelled and treated database.

An aspect to highlight is that all process, until this moment, have been development off-line, and time needed for this process is not important for the diagnosis process.

Diagnosis process consists on evaluating an observation with the obtained decision rules. Time spending to diagnose is only the time of evaluating obtained decision rules. Decision rules returns the label associated to the behaviour by correspondence between training data and observed data. This returned label is offered as diagnosis.

Next a case study will be presented to develop this methodology.

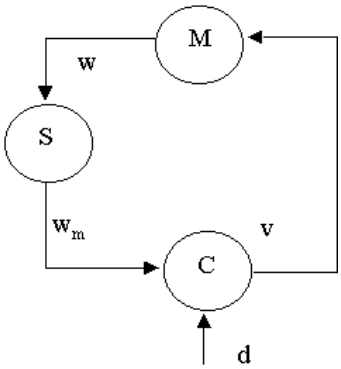


Figure 2. The example system

3 CASE STUDY

As it has been commented previously, methodology can be used with real system data or with obtained data of a model simulation. In our case, the methodology will be applied to a model, which is an idealized situation, but it offers us a clear idea of the way to act. In case of application on a real system, many difficult aspects, not mentioned here (as monitoring or small phase shift), need to be taken in account, but with the model we are only trying to present the approach.

As example of dynamic system to diagnose we consider the controller electric motor in [18] and [19]. Figure 2 represents treated system. The motor 'M', whose rotational speed is '*w*', is driven through a voltage '*v*' by the controller 'C' which acts based on the desired speed '*d*' and the speed 'w_m' measured by the revolution counter 'S'. Controller 'C' is considered as an I-controller.

System can be modelled by the following equations, which include a constant for each component that is used to model also the faulty behaviour of the component:

$$Motor: T * \frac{dw}{dt} = c_m * v - w \quad (1)$$

$$I - Controller: \frac{dv}{dt} = c_c * (d - w_m) \quad (2)$$

$$Sensor: w_m = c_s * w \quad (3)$$

Where T is the inertia of the motor, c_m is the constant of the motor; c_c is the constant of the controller and c_s is the constant of the revolution counter.

Component anomalous operation is caused, mainly, by the deviation of the component constant nominal value. These constants stray of the considered correct values range

Some faults represent that constants take values above the correct ones and others faults represent that constants take values below the correct ones. Diagnosis result should indicate, in addition to the faulty component, if taken values for the component constant are below correct values or above them.

Possible fault reasons that we want to identify are therefore: 'CmHigh' when values of Cm are above the correct ones; 'CmLow' when values of Cm are below the correct ones; 'CsHigh' when values of Cs are above the correct ones; 'CsLow' when values of Cs are below the correct ones; 'CcHigh' when values of Cc are above the correct ones and 'CcLow' when values of Cc are below the correct ones.

To describe the system correct behaviour, it is considered that values of all constants don' t have only one correct value, but rather they can take values inside an interval that will be considered as correct.

This way, operation flexibility is allowed and system real behaviour is better simulated, where there is not a correct value but rather correction margins are flexible. This produces that system doesn' t have an only correct behaviour, but rather a correct behaviours family. It represents all possible combinations of the constants values that are inside of the defined tolerance limit.

A correct behaviours family does the diagnosis more difficult, because it is necessary to recognize different behaviours as correct, but on the contrary it provides a more realistic vision of the system.

In our model the constant values considered as correct are:

Table 1. Values for OK behaviours

Cm	[0.98-1.02]
Cc	[0.98-1.02]
Cs	[0.98-1.02]

Other considered characteristics in our system are:

1. Fault is present from the beginning and it doesn' t evolve in the time.
2. Behaviour change occurs instantly and starting from here it doesn' t change again.
3. Once the wanted angular speed has been indicated, it doesn' t change until this angular speed is reached.

This way, diagnosis will be carried out when the desired angular speed (d) is changed. The way to diagnose is by checking the evolution to reach the final speed. It is necessary to keep in mind that in spite of existence of a failure in some component, I-controller is able to act on the motor to reach the required final speed. Of course evolution of the system to reach the desired final speed will be different. This difference in the behaviour will allow the diagnosis.

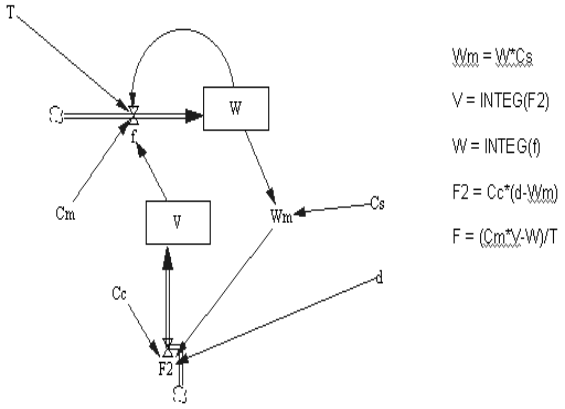


Figure 3. Forrester diagram

First step, therefore, is performing system simulations in different behaviours modes. In our case, system has been modelled as a Forrester diagram [20], to be able to simulate using the simulation tool VEMSIM®. Forrester diagram generated for the system is presented in figure 3.

Simulated behaviours will be those that we want to diagnose. They will be: OK for correct behaviour and CmHigh, CmLow, CsHigh, CsLow, CcHigh, CcLow for each component fault above mentioned.

A behaviour family will represent each one of these behaviours.

Simulations values are shown in table 2.

Table 2. System values for simulation

T	3
D	10
W	5
Time Step	0.1

For the correct behaviour the constant values are into [0.98-1.02]. Values to simulate behaviours above the correct one are into [1.02-5]. Values to simulate behaviours bellow the correct one are into [0-0.98].

Constants values for simulated behaviours have been elected by random with the Monte Carlo method following a uniform distribution. Number of simulations per behaviour will be 100.

Label corresponding to behaviour is placed to each one of the trajectories. This way, a database containing 700 labelled trajectories is obtained.

Trajectories are composed with values of the variable 'w_m' in each time step. Reason to select variable 'w_m' and not 'w' is that 'w_m' is the only observable variable in the real system.

In figures 4, 5 and 6 different system behaviours are shown.

Obtained database has similar trajectories belong to different behaviours. This way several very similar trajectories have different labels. This is a problem, because our final goal is to use a classification tool to obtain a set of decision rules, and if we have similar trajectories with different labels then classifier can't correctly work; that is to say, those similar trajectories will be incorrectly classified. Figure 7 shows an example of this.

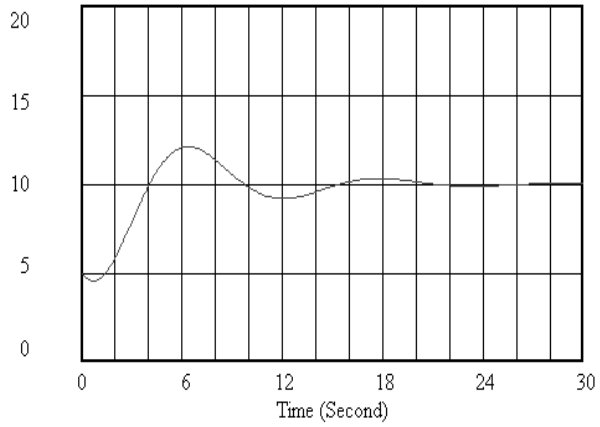


Figure 4. OK Behaviour

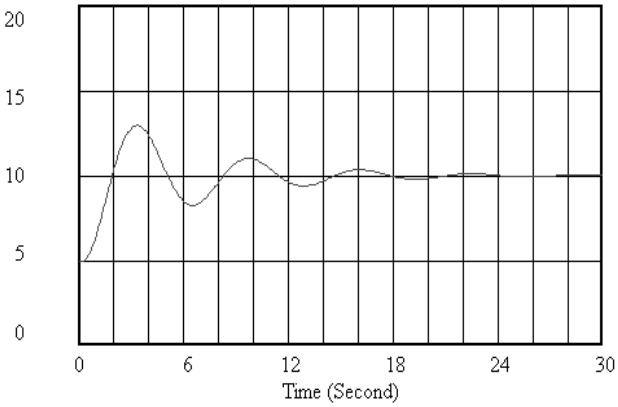


Figure 5. CmHigh Behaviour

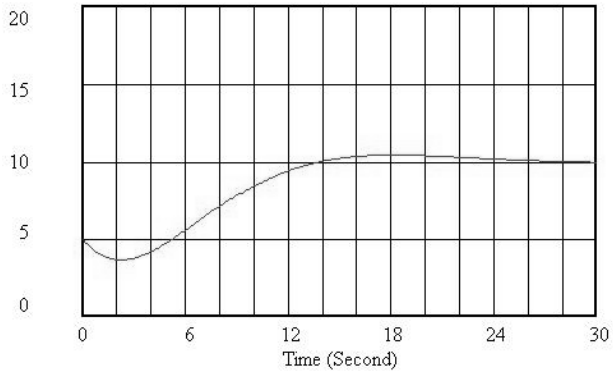


Figure 6. CcLow Behaviour

To solve this problem a new label will be assigned to very similar trajectories. A mixture of labels of all similar trajectories will compose the new label. This way, next step is to find all similar trajectories into the database and assigning a new label.

It is necessary to define when two or more trajectories are similar. Two trajectories are considered similar when distance between them is smaller than a magnitude. Distance between trajectories is measured as Euclidean Distance and magnitude chosen is 10% of the Euclidean distance between the two further away trajectories for the correct behaviour. This magnitude in our example is 0.45.

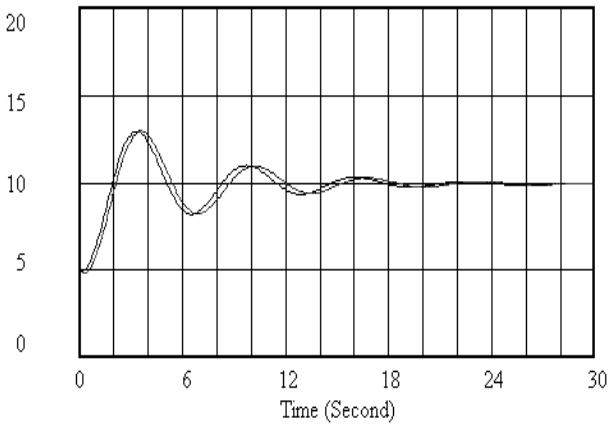


Figure 7. Behaviour CcHigh vs CmHigh

After this process we obtain a new database with all similar trajectories re-labelled as corresponding with all behaviours of the similar trajectories.

Next step is to calculate new attributes of each trajectory with the goal that classifier has more information to generate decision rules. These new attributes must be representative for each trajectory.

For each trajectory point next attributes have been calculated:

- Distance to perfect behaviour. It indicates how far away is current trajectory from perfect behaviour (above defined). It is calculated as:

$$DP(i) = Wm[i] - Wmpf[i] \quad (4)$$

  Where *Wm[i]* is the treated point in the current trajectory and *Wmpf[i]* is the correspondent point in the perfect behaviour.
- Integral. It is the magnitude returned by numerical integration between current point and the precedent one. It represents the closed area between them. It is calculated by approximating as follow:

$$I(i) = Ts \times \frac{p[i] - p[i-1]}{2} \quad (5)$$

  Where *Ts* is the time step in the simulation, *p[i]* is the current treated point and *p[i-1]* the precedent one.

In addition next attributes will be calculated for each trajectory:

- Rise Time (RT). It is the moment in which desired revolution speed is reached for first time.
- Steady state (SS). It is the moment in which desired revolution speed is reached definitively.
- Max speed (MS). It is the value of the highest revolution reached speed.
- Max speed time (MST). It is the moment in which the highest revolution speed is reached.

This way a new database containing trajectories plus new attributes is generated.

Data in new database have the following form:

RT, SS, MS, MST, Wm[1], DP[1], I[1], ……., Wm[n], DP[n], I[n], LABEL

Final step is performing supervised learning with the obtained database. Classification tool selected to perform the supervised learning is C4.5 [21]. What is gotten with this tool is to characterize each one of the behaviour families according to the values of the attributes that have been provided. Result is a decision tree and an equivalent set of decision rules. These rules will be the way to do the diagnosis. In our example classifier obtains 27 rules with an error rate of 1.2%. This mean that 1.2% of trajectories are not correctly classified with those rules.

3.1 Diagnosis

The way to do the diagnosis is evaluate the observed data with the obtained rules.

Because in rules appear attributes that have been calculated and not appear in observed data, same attributes should be calculated for observed data in order to be able to classify with those rules.

This way in the moment that one observed data is gathered all possible attributes should be calculated. After that, decision rules are evaluated with two possible results: a label is returned or information is insufficient to evaluate all rules. In the first case the returned label is the result of the diagnosis. In the second one we need to wait more information in further moments.

If we want to diagnose the system with another running conditions, we should have prepared the decision rules set for those specific conditions. I. e. if we want to diagnose this system when current rotational speed is 12 rad/sec and desired rotational speed is 7 rad/sec, we should have generated a set of decision rules for those conditions and we will use them in the diagnosis moment.

4 RESULTS ON THE EXAMPLE SYSTEM

To evaluate the proposed methodology a set of tests have been done.

Observational data have been obtained by simulating the system with specific conditions for the test. This way a test trajectory is obtained and the diagnosis correct result is known, because it must be the corresponding to the simulated conditions.

Conditions of the test are the same above mentioned. We remember them in table 3:

Table 3. Tests conditions

T	3
D	10
W initial	5
Time Step	0.1
Values for OK	[0.98 - 1.02]
Values for HIGH	[1.02 - 5]
Values for LOW	[0 - 0.98]

In table 4 we can see results for the tests:

Table 4. Tests results

VALUE OF THE CONSTANT			CORRECT DIAGNOSIS	DIAGNOSIS WITH SIMPLE LABELLED	DIAGNOSIS WITH RE-LABELLED
Cm	Cc	Cs			
1	1	1.03	CS HIGH	CS HIGH	CS HIGH
1	1	1.07	CS HIGH	CS HIGH	CS HIGH
1	1	1.1	CS HIGH	CS HIGH	CS HIGH
1	1	1.5	CS HIGH	CS HIGH	CS HIGH
1	1	2	CS HIGH	CS HIGH	CS HIGH
1	1	3	CS HIGH	CS HIGH	CS HIGH
1	1.03	1	CC HIGH	OK	OK
1	1.07	1	CC HIGH	CM HIGH	CC HIGH | CM HIGH
1	1.1	1	CC HIGH	CM HIGH	CC HIGH | CM HIGH
1	1.5	1	CC HIGH	CC HIGH	CC HIGH
1	2	1	CC HIGH	CC HIGH	CC HIGH
1	3	1	CC HIGH	CC HIGH	CC HIGH
1.03	1	1	CM HIGH	OK	OK | CS LOW
1.07	1	1	CM HIGH	CM HIGH	CC HIGH | CM HIGH
1.1	1	1	CM HIGH	CM HIGH	CC HIGH | CM HIGH
1.5	1	1	CM HIGH	CM HIGH	CM HIGH
2	1	1	CM HIGH	CM HIGH	CM HIGH
3	1	1	CM HIGH	CM HIGH	CM HIGH
1	1	0.97	CS LOW	OK	CS LOW | OK
1	1	0.93	CS LOW	CS LOW	CS LOW
1	1	0.89	CS LOW	CS LOW	CS LOW
1	1	0.85	CS LOW	CS LOW	CS LOW
1	1	0.5	CS LOW	CS LOW	CS LOW
1	1	0.1	CS LOW	CS LOW	CS LOW
1	0.97	1	CC LOW	OK	OK
1	0.93	1	CC LOW	CC LOW	CC LOW | CM LOW
1	0.89	1	CC LOW	CC LOW	CC LOW | CM LOW
1	0.85	1	CC LOW	CC LOW	CC LOW | CM LOW
1	0.5	1	CC LOW	CC LOW	CC LOW
1	0.1	1	CC LOW	CC LOW	CC LOW
0.97	1	1	CM LOW	OK	OK
0.93	1	1	CM LOW	CC LOW	CC LOW | CM LOW
0.89	1	1	CM LOW	CM LOW	CC LOW | CM LOW
0.85	1	1	CM LOW	CM LOW	CC LOW | CM LOW
0.5	1	1	CM LOW	CM LOW	CM LOW
0.1	1	1	CM LOW	CM LOW	CM LOW
0.99	0.98	1.02	OK	OK	OK
1	1.02	1.02	OK	OK	OK
0.98	1	0.98	OK	OK	OK
0.98	1.02	1.02	OK	OK	OK
0.99	1.01	1.01	OK	OK	OK
1.01	1	0.99	OK	OK	OK

We can see that diagnosis methodology with simple labelled doesn't offer a correct diagnostic in tests that are very near of the correct behaviour. In those cases the fault is not detected. Other times, methodology returns an incorrect diagnosis, but in general offered results are acceptable.

This occurs because there are very similar trajectories belonging to different behaviours, and classifier cannot correctly select the rules to difference them.

To solve this problem the new methodology proposes the re-labelled of all similar trajectories as have been above mentioned. Obtained results show that the new methodology offers a multiple diagnosis when the previous one can't find the correct fault. Among the multiple offered diagnoses, near to all tests return the correct one.

It is important to highlight that, in tests where behaviour is far of the correct one, offered diagnosis is the correct one.

In the set of presented tests the diagnosis is correct in 58.33 % of the cases. Correct diagnosis is offered, among others, in 30.55 % of the cases. An incorrect diagnosis is offered in 2.7 % of the cases. The fault is not detected in 8.33 % of the cases. Otherwise, never detect failure when failure doesn't exist.

5 CONCLUSIONS AND FURTHER WORKS

Presented methodology is able to perform diagnosis of dynamic systems and it is independent of the system type. In fact, one of further works is to apply this methodology to a non-linear dynamic system.

This capacity is due to the fact that the methodology is only centred in the evolution characteristics of the system for the correct behaviour or faulty behaviours.

Another characteristic of the methodology is that the diagnosis can be performed in a very simple way, and a very little computational time is required.

Certain systems, as the presented in the example, can produce similar behaviours for different fault reasons. This is due to relationship among variables that govern the system behaviour. This relationship, among system variables, can produce that an alteration of a variable would be compensated by the alteration of another variable in contrary sense. To solve this problem, methodology assigns multiple fault reasons to system behaviours that could be produced by different fault reasons. This way a multiple diagnosis is offered in those situations.

Another further work is to be able to diagnose dynamic system when multiple fault occurs at the same time, is to say, identifying system behaviours when more than one component is faulty.

ACKNOWLEDGMENTS

This work has been partiality financed by the Comisión Interministerial de Ciencia y Tecnología (DPI2000-0666-C02-02) and the Modelización Matemática Redes y Multimedia investigation group of the University of Huelva.

REFERENCES

[1] C. J. Alonso, J. A. Maestro, J. B. Pulido y C. Llamas. *Monitorización de Sistemas Dinámicos: hacia una Caracterización en el Nivel de Conocimiento*. In proceedings of the I Jornadas de Trabajo sobre Diagnosis. Valladolid 2001.

[2] W. Hamscher. *Diagnosis devices with hierarchic structure and known component failure models.* In proceedings of the 6th Conference on AI Applications.. 1990
[3] Dague, P y otros. *When Oscillators stop oscillating*. In proceedings of IJCAI-91
[4] J. De Kleer y B. Williams. *Diagnosing multiple faults.* Artificial Intelligence 32, 97-130, 1987
[5] K. Bousson, y L. Trave-Massuyes A computational causal model for process supervision. Technical Report 92147, LAAS-CNRS, Toulouse, France. 1992
[6] P. Mosterman *Hybrid dynamic systems: a hybrid bond graph modeling paradigm and its applications in diagnosis*. Tesis Doctoral Vanderbilt University, Nashville, Tennessee, USA. 1997.
[7] P. Struss. Fundamentals of model-based diagnosis of dynamic systems. Proc. IJCAI'97. 1997.
[8] B. Pulido, C. Alonso y F. Acebes. *Consistency-based Diagnosis of Dynamics Systems using quantitative models and off-line dependency-recording*. In proceedings of DX-01.2001
[9] A.D. Pouliezos y G.S. Stavrakakis. Real time fault monitoring of industrial process. Microprocessor-based systems engineering. Kluwer Academic Publishers, Dordrecht, 1994.
[10] V. Venkatusubramanian and K. Chan. *A neural network methodology for process fault diagnosis*. Journal of Artificial Intelligence in Chemical Engineering, 35:1993-2001. 1995
[11] S. Leonhart y M. Ayoubi. *Methods of fault diagnosis*. Control Engineering Practice, 5. 1997.
[12] A. Simón, L. Alonso y A. Antón. *Sistema híbrido borroso para ayuda del diagnóstico del glaucoma*. In proceedings of the I Jornadas de Trabajo sobre Diagnosis. Valladolid 2001.
[13] S. Saludes, A. Vargas y J. R. Perán. *Aplicación de la red neuronal SOM para la detección de fallos desconocidos en un grupo hidroeléctrico.* In proceedings of the I Jornadas de Trabajo sobre Diagnosis. Valladolid 2001.
[14] J. Ross Quinlan. *Induction of decision trees.* Machine learning, 1986
[15] Juan J. Rodríguez, Carlos J. Alonso y Q. Isaac Moro. *Clasificación de patrones temporales en sistemas dinámicos mediante Boosting y Alineamiento dinámico temporal.* In proceedings of the I Jornadas de Trabajo sobre Diagnosis. Valladolid 2001.
[16] Pedro J. Abad y Antonio J. Suárez. Diagnosis de Sistemas Dinámicos Basada en Aprendizaje Supervisado O*ff-line.* In proceedings of the I Jornadas de Trabajo sobre Diagnosis. Valladolid 2001.
[17] Antonio J. Suárez y Pedro J. Abad. *Aplicación de Técnicas de Aprendizaje a la diagnosis de Sistemas Dinámicos con Etiquetado Múltiple*. In proceedings of the IX CAEPIA –TTIA. 2001.
[18] A. Panati and D. T. Drupé *Stated based vs simulation-based diagnosis of dynamic system.* ECAI2000. !4Th European Conference on Artificial Intelligent. 2000.
[19] A. Malik and P. Struss. *Diagnosis of Dynamic Systems does not necessarily require simulation.* Workshop Notes of the Seventh International Workshop on Principles of Diagnosis DX-96 Montreal. 1996.
[20] J. W. Forrester *Principles of systems.* Wright-Allen Press 1968.
[21] J. Ross Quinlan. *C45:Program for Machine Learning*. Morgan Kaufman, 1993.

Fault Isolation using Process Algebra Models

Dan Lawesson[1], **Ulf Nilsson**[1], **Inger Klein**[2]
[1]Dept of Computer and Information Science
[2]Dept of Electrical Engineering
Linköping University, 581 83 Linköping, SWEDEN
{danla,ulfni}@ida.liu.se inger@isy.liu.se

Abstract

We investigate the problem of doing post mortem fault isolation for concurrent systems using a behavioral model. The aim is to isolate the action that has caused the failure of the system, the root action. The naive approach would be to say that a certain action is the root action iff it is a logical consequence of the model and observations that the action is the first "bad thing to happen". This, however, is a strong requirement and puts high demand on the model. In this paper we describe the concept of *strong root candidate*, a relaxation of the naive approach. The advantage of determining the strong root candidate directly from model and observations is that the set of traces consistent with model and observations need not be explicitly computed. The property of strong root candidate can instead be determined on-the-fly, thus only computing relevant parts of the reachable state space.

1 Introduction

In this paper we describe a model-based [Hamscher *et al.*, 1992] approach to fault isolation in object oriented control software. The work is motivated by a real industrial robot control system developed by ABB Robotics. The system is large (the order of 10^6 lines of code), concurrent, has an object oriented architecture and is highly configurable, supporting different types of robots and cell configurations. Since the system is time- and safety-critical the first priority, in case of a failure, is to bring the system to a safe state; alarms that go off are logged and can be analyzed when the system comes to a stand-still. The faults considered are primarily hardware faults, and therefore we rely on the assumption that the failing hardware has some software counterpart that is affected by the failure of the hardware. In addition we make the common single fault assumption, i.e. that a system failure is caused by only one fault (but resulting in cascading alarms).

The log thus contains *partial information* about the events that took place at the approximate time of the system failure. However, the order in which messages are logged does not necessarily reflect the way error messages propagate – the system is concurrent and safety critical actions may have to be taken before error reporting takes place. Hence, in what follows we (somewhat conservatively) view the log as a *set* of error messages. In addition a system may contain a number of critical events that are unobservable, but which may explain all observable alarms.

The ultimate aim of our fault isolation method is to single out the error message that explains the actual cause of the failure, or possibly an unobservable critical event explaining the observations. That is, we aim to discard error messages which are definitely effects of other error messages, while trying to isolate error messages (or critical events) which explain all other messages. In contrast to message filtering, we can thus find failing components that have not manifested themselves in the error log, if the failing of the component is a logical consequence of the model and the observations. Given the size of the software it is not possible to use the code directly – we have to rely on a model of the software. In this paper we consider finite state machine models expressed in a process algebra. The process algebra is chosen here because it allows for more straightforward formal reasoning than for example state charts, but the contribution of this work - the fault isolation - relies only on the labeled transition system semantics of the model. In practice, the aim is to use a behavioral model that is an artifact of the software development process, such as state charts. Then there is no extra cost associated with maintaining a correct model when the software evolves, since then so does the model.

In standard AI diagnosis literature, see e.g. [Reiter, 1987], a diagnosis is a (minimal) set of failed components explaining the observations. But for dynamic systems (systems with state) a diagnosis is often defined as the set of all traces, or trajectories, consistent with the observations (see e.g. [Cordier *et al.*, 2001; Console *et al.*, 2000]). However, this definition is generally insufficient to isolate the origin of the fault(s), and requires post-processing to pin-point e.g. the faulty component(s). Our approach is more direct and focuses on finding the alarm that explains (is consistent with) all observables: given the system description, expressed in a simple process algebra, and the observations, we try to infer the origin of the fault using properties of actions involving the temporal order, expressed in a specification language based on a subset of the temporal logic CTL, originally developed for verification [Clarke *et al.*, 1999]. This resembles the process of model checking and as in the case of model-checking there is no need for calculation of the entire state space (obviously

equivalent to the set of traces consistent with model and observations) if the temporal logic formulae are evaluated by constructing the state space on-the-fly.

Our approach also bears some resemblance to that of Sampath et al. [Sampath *et al.*, 1995]. However their work is mainly concerned with diagnosability in discrete event systems; i.e. to detect, within finite delay, whether a certain type of fault has occurred. While our approach is amenable only to post-mortem analysis, the work reported in [Sampath *et al.*, 1995] is mainly intended for monitoring and on-line detection and diagnosis.

The rest of the paper is organized as follows: In Section 2 we describe the behavior language that will be used to define a transition relation, that defines the set of all possible behaviors (i.e. traces). In Section 3 we provide rules for entailment of some predicates of interest from configurations and the traces that can follow from them. Finally, we outline ongoing and future work in Section 4.

2 A behavior language

A behavior model can be expressed in different ways, and we have chosen to use a process algebra. No matter which formalism and notation that is used, the semantics should provide a labeled transition relation that describes the state transitions of the modeled system. In this section we describe a process algebra influenced by CCS [Milner, 1989] and give the necessary semantics.

2.1 Processes

Our process language is constructed from the following syntactic categories

- a finite set $\mathcal{L}$ of *action labels* denoted by a in our meta language. Every action label is equipped with an associated arity $n \geq 0$.
- a set $\mathcal{O}$ of *object id's* denoted by o.
- a finite set $\mathcal{S}$ of *states* A with associated arity $n \geq 0$.

We consider four types of *actions* (denoted by α in our meta language).

- Send actions of the form $o{:}\overline{a}(\mathbf{t})$, where o is the recipient object, $\overline{a}$ an n-ary action label and $\mathbf{t}$ is an n-tuple of object id's or variables.
- Receive actions of the form $a(\mathbf{x})$ where a is an n-ary action label and $\mathbf{x}$ is an n-tuple of variables.
- Internal actions of the form a, where a is a nullary action label.
- New-actions of the form $new(o, P)$ where $o \in \mathcal{O}$ and P is a process expression, defined below.

A process is described by a *process expression*, denoted by P (and occasionally Q), and given by the following abstract syntax

$$\mathcal{P} ::= A(\mathbf{t}) \mid \sum_{i \in I} \alpha_i.\mathcal{P}$$

where I is a finite index set. Sums are usually written simply $\alpha_1.P_1 + \alpha_2.P_2$. We reserve the nullary state $Stop$ for a completed process. We assume that every $A/n \in \mathcal{S}$ ($Stop$ excepted) has a defining equation of the form

$$A(\mathbf{x}) \stackrel{def}{=} P.$$

A *process state* σ is a partial map from $\mathcal{O}$ to $\mathcal{P}$. The object $init \in \mathcal{O}$ is called the *initializing object*, the state $Main \in \mathcal{S}$ is called the *main process* and the state $\sigma_0 := \{init \mapsto Main\}$ is called the *initial process state*.

Let $\sigma{:}\mathcal{O} \to \mathcal{P}$ be a process state, $o \in \mathcal{O}$ and $P \in \mathcal{P}$. By $\sigma[o \mapsto P]$ we denote the process state which is almost identical to σ except possibly at o. That is

$$\sigma[o \mapsto P](x) := \begin{cases} P \text{ if } x = o \\ \sigma(x) \text{ otherwise} \end{cases}$$

The behaviors of our system are described by the labeled transition rules in Figure 1. Our transitions are of the form

$$\sigma \xrightarrow{\alpha} \sigma'$$

where α (the observation) is a set of pairs of the form (o, a) representing action a occurring in object o.

There are four transition rules, *sync*, *internal*, *new* and *def*. The rule *sync* allows two objects to synchronize their state transitions and optionally exchange values. In our limited algebra, the only values that can be transmitted are object identifiers. However, the idea is not to model all system behavior, but to have a system model that reveals synchronization and system structure. The rule *internal* allows a single object to perform a transition by itself. Creation of new objects is handled by the rule *new*, and *def* allows for exchanging a state with its definition.[1]

Example

Typically, a system is described by creating a main process that sets up the system structure. Figure 2 shows an example of such a system. Process $Main$ creates three objects and runs $Setup$ which tells the objects about each other via the $\overline{init}$ call. This is needed since when started, a process does not know anything about its environment. After $init$, each object will act as a peer-to-peer node, as showed in Figures 3 (the system) and 4 (object details). Objects can send requests to each other, and sometimes the answer to a request is a failure, and then the system is brought to a halt by transmission of $down$ messages.

3 Fault Isolation

The available information when doing fault isolation is a system model and an observation (in our case a message log). We use the term *scenario* to refer to that information. In the following we overload the term action in the context of scenarios to mean pairs $(o, a) \in \mathcal{O} \times \mathcal{L}$ where o is an object identifier and a is an action label. Some of the actions in a system are *critical actions*, actions that are associated with system failures.

Thus a scenario is a quadruple $(\longrightarrow, Crit, Log_p, Log_n)$, where $\longrightarrow$ is a process state transition relation, $Crit \subseteq \mathcal{O} \times \mathcal{L}$

[1] Since we rely on a finite state space model, we do not allow unbounded creation of objects via the *new* rule.

$$\text{sync}: \frac{\sigma(o_i) = P_1 + o_j : \overline{a}(\mathbf{t}).P + P_2 \quad \sigma(o_j) = P_3 + a(\mathbf{x}).Q + P_4}{\sigma \xrightarrow{\{(o_i,\overline{a}),(o_j,a)\}} \sigma[o_i \mapsto P][o_j \mapsto Q\{\mathbf{x}/\mathbf{t}\}]}$$

$$\text{internal}: \frac{\sigma(o_i) = P_1 + a.P + P_2}{\sigma \xrightarrow{\{(o_i,a)\}} \sigma[o_i \mapsto P]}$$

$$\text{new}: \frac{\sigma(o_i) = P_1 + new(o, Q).P + P_2 \quad \sigma[o_i \mapsto P][o \mapsto Q] \xrightarrow{\alpha} \sigma'}{\sigma \xrightarrow{\alpha} \sigma'}$$

$$\text{def}: \frac{\sigma(o_i) = A(\mathbf{t}) \quad A(\mathbf{x}) \stackrel{def}{=} P \quad \sigma[o_i \mapsto P\{\mathbf{x}/\mathbf{t}\}] \xrightarrow{\alpha} \sigma'}{\sigma \xrightarrow{\alpha} \sigma'}$$

Figure 1: Process transition rules ($\mathbf{t}$ is a vector of object id's)

$$\begin{array}{lcl}
Servent(this, x, y) & \stackrel{def}{=} & x{:}\overline{req}(this).Wait(this, x, y) + y{:}\overline{req}(this).Wait(this, x, y) + \\
 & & req(o).Compute(this, x, y, o) + down().Down \\
Wait(this, x, y) & \stackrel{def}{=} & ok().Servent(this, x, y) + fail().Fail(x, y) \\
Compute(this, x, y, o) & \stackrel{def}{=} & o{:}\overline{ok}().Servent(this, x, y) + o{:}\overline{fail}().Servent(this, x, y) \\
Fail(x, y) & \stackrel{def}{=} & x{:}\overline{down}().Fail(x, y) + y{:}\overline{down}().Fail(x, y) \\
Down & \stackrel{def}{=} & Stop \\
S & \stackrel{def}{=} & init(this, x, y).Servent(this, x, y) \\
Main & \stackrel{def}{=} & new(s_1, S).new(s_2, S).new(s_3, S).Setup(s_1, s_2, s_3) \\
Setup(x, y, z) & \stackrel{def}{=} & x{:}\overline{init}(x, y, z).y{:}\overline{init}(y, z, x).z{:}\overline{init}(z, x, y).Stop
\end{array}$$

Figure 2: A process algebra example

is the set of critical actions, $Log_p \subseteq \mathcal{O} \times \mathcal{L}$ is the set of actions that have been observed (i.e. the message log), and $Log_n \subseteq \mathcal{O} \times \mathcal{L}$ is the set of actions known not to have occurred (i.e. the observable actions not contained in the message log). Thus, we assume that a synchronized action is logged as two separate actions – one from the sending object and one from the receiving. This allows modeling of message sending with unknown receiver and is no severe limitation since it is possible to express receiver information by having a model where the desired action labels are unique and receiver object id thus becomes unambiguous.

A *configuration*, denoted C, is the symbol $\perp$ or a pair (σ, l) where σ is a process state and $l \subseteq \mathcal{O} \times \mathcal{L}$ is a set of actions. The following rules defines the *configuration transition relation* $\Rightarrow$ for a given $\longrightarrow$ and Log_n.

$$\frac{\sigma \xrightarrow{\alpha} \sigma' \quad \alpha \cap Log_n = \emptyset}{(\sigma, l) \Rightarrow (\sigma', l \cup \alpha)}$$

$$\frac{\sigma \xrightarrow{\alpha} \sigma' \quad \alpha \cap Log_n \neq \emptyset}{(\sigma, l) \Rightarrow \perp}$$

The configuration $(\{init \mapsto Main\}, \emptyset)$ is called the *initial configuration*. The configuration $\perp$ is called *a forbidden configuration* and represent configurations that are allowed by the behavioral model, but inconsistent with the observations at hand. We see configurations as snapshots of the system state of a given scenario, and the configuration transition relation describes the behavior of the system. Fault isolation is the process of finding the first critical action that has occurred in a given scenario, the *root action*. Given the single fault assumption and a system model that is properly designed, the first critical action to occur in the system is the cause of the failure.

An action α is *present* in a scenario if the system model and the observation entails the occurrence of α. An action α is an *enabled root* if the assumption that α is root action is consistent with the observations and the system model. We introduce the concept of *strong root candidate*, and say that a strong root candidate is an action that is both present and an enabled root.

3.1 Predicate rules

Given a certain scenario $(\rightarrow, Crit, Log_p, Log_n)$, we wish to reason about properties of reachable configurations. Therefore we define predicates, that correspond to the interesting properties, by determining for which configurations they hold true. Since we are interested in strong root candidates, we need to formally define present actions and enabled root actions. Thus we define the predicate $present(\alpha)$ that holds in configurations where action α must occur sometime in the future and the predicate $enabledroot(\alpha)$ that holds for configurations where it is consistent to assume that α may be the first critical action to occur. In defining these two predicates, we will need some helper predicates. We will use $okend$ that holds in configurations that correspond to consistent ending states of the system. An ending state is a state where no more observable actions occur, i.e. when the system has reached a final state. In a configuration where α has occurred, $seen(\alpha)$ holds, while $nocrit$ holds in configurations where no critical action has occurred. The predicate end holds in configurations where there is no next configuration.

We define entailment of logical formulae from the following syntax:

$$\mathcal{F} ::= \mathcal{F} \vee \mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \neg\mathcal{F} \mid EF(\mathcal{F}) \mid EX(\mathcal{F}) \mid AG(\mathcal{F}) \mid end \mid okend \mid nocrit \mid seen(\alpha) \mid present(\alpha) \mid enabledroot(\alpha)$$

In order to be able to define entailment for the desired predicates, we will need the following. We use $\stackrel{*}{\Rightarrow}$ for the reflexive transitive closure of $\Rightarrow$. First we define entailment for basic connectives.

$$\frac{C \models F_1 \quad C \models F_2}{C \models F_1 \wedge F_2} \qquad \frac{C \models F_2}{C \models F_1 \vee F_2}$$

$$\frac{C \models F_1}{C \models F_1 \vee F_2} \qquad \frac{C \not\models F}{C \models \neg F}$$

We will be reasoning about temporal order, so we need to define temporal logic operators.

$$\frac{C \stackrel{*}{\Rightarrow} C' \quad C' \models F}{C \models EF(F)}$$

$$\frac{C \Rightarrow C' \quad C' \models F}{C \models EX(F)}$$

$$\frac{C' \models F \text{ whenever } C \stackrel{*}{\Rightarrow} C'}{C \models AG(F)}$$

We also need entailment for a few helper predicates. The predicate end determines if a configuration lacks successor (i.e. $end \equiv \neg EX(true)$ where $true$ is entailed by every configuration), $seen(\alpha)$ is true when an action α has occurred and $nocrit$ holds when no critical actions have yet occurred.

$$\frac{\neg\exists C', C \Rightarrow C'}{C \models end} \qquad \frac{\alpha \in l}{(\sigma, l) \models seen(\alpha)} \qquad \frac{\forall \alpha \in l, \alpha \notin Crit}{(\sigma, l) \models nocrit}$$

Now we have the tools needed to define the desired predicates. If we have reached a configuration from which the system cannot continue to execute and all actions in Log_p are seen, then the configuration is an $okend$, unless the configuration is a forbidden configuration. It is thus one of the possible halting configurations, given the scenario at hand.

$$\frac{\forall \alpha \in Log_p, C \models seen(\alpha) \quad C \models end \quad C \neq \perp}{C \models okend}$$

If it is true for all reachable configurations that whenever we have reached an $okend$, we have seen action α, we conclude that the presence of α is entailed from observations and system model.

$$\frac{C \models AG(\neg okend \vee seen(\alpha))}{C \models present(\alpha)}$$

If there is a reachable configuration C_1 such that no critical actions has taken place, and there is a configuration step that takes us from C_1 to C_2 where the critical action α has occurred, we conclude that α is an enabled root if it is possible to reach an $okend$ from C_2.

$$\frac{\alpha \in Crit \quad C \models EF(nocrit \wedge EX(seen(\alpha) \wedge EF(okend)))}{C \models enabledroot(\alpha)}$$

3.2 Reasoning about behavior

Given a scenario, the strong root candidates are the actions α for which

$$(\{init \mapsto Main\}, \emptyset) \models present(\alpha) \wedge enabledroot(\alpha)$$

If we have no strong root candidates or more than one strong root candidate, the system model is not strong enough for efficient fault isolation. If, on the other hand, we have exactly one strong root candidate, we assume that we have pinpointed the true cause of the fault. This is reasonable to assume, since the action found is the only one that is known to have occurred (its presence is entailed by the scenario) and it is consistent with the given scenario to assume that the action is a root event.

Of course there is still a possibility that there are other enabled root events whose presence are consistent with the scenario, but assuming one of them to be root would demand an explanation to why the strong root candidate (proven to be present!) is not the root.

3.3 Prototype implementation

We have designed a prototype XSB [Sagonas *et al.*, 1994] program that takes a system model and observations as input and enumerates the strong root candidates. XSB is a Prolog dialects using tabulation (memoization) to improve termination. Given the system model in Figure 2 and facts stating that any sending of $fail$ or $down$ indicates system failure, i.e. those actions are critical actions, and the observations that $(o_2, \overline{fail})$ *has not* occurred and $(o_3, fail)$ *has* occurred, the XSB Prolog program computed $(o_1, \overline{fail})$ to be the single strong root candidate.

The system consists of three objects that all execute the same process. See Figure 4 for an automata representation of a similar process (parameters are not explicit in the automata). Consider the critical actions. Obviously, no $\overline{down}$ message can be root action since it will always be preceded by a $\overline{fail}$ action, and neither can $(o_2, \overline{fail})$ be root action since it is known to not have occurred at all. This leaves us with $(o_1, \overline{fail})$ and $(o_3, \overline{fail})$. It is consistent with the system model and the observations to assume that $(o_3, \overline{fail})$ is the root action, since if o_2 receives the $fail$ from o_3, then o_1 can send $fail$ to o_3 afterwards. We cannot prove that $(o_3, \overline{fail})$ has happened, however. This can be done for $(o_1, \overline{fail})$, and therefore it is the only action that is both enabled root and present.

Thus, having some intuition of the system makes the fault isolation described above almost trivial, but the key motivation of this work is to formalize and automate this intuition.

4 Future Work

In previous work with Larsson [Larsson *et al.*, 2000; Larsson, 1999] we studied the fault isolation problem using a structural model. A key feature of that approach is the use of software engineering models, in particular UML [Object Management Group, 1999] class diagrams. Such a model can be developed and maintained at a relatively low cost being an integrated part of the software development process. The work presented here and in our previous work [Lawesson, 2000; Lawesson *et al.*, 2001] aims to strengthen the diagnostic capability while still using standard and state-of-the-art modeling notations. Behavior in UML is often expressed using statecharts, and process algebras provide a textual representation of state machines. Of course, enforcing the software developer to construct complete statecharts for all classes is not realistic in large software systems; hence, reasoning must be able to cope with incomplete or missing behavioral descriptions. Our approach should also be extended to deal with the special features characteristic of object oriented software systems such as classes and inheritance. Below we sketch some partial solutions to such issues, which will be addressed in our future work.

4.1 Classes behaviors and inheritance

Our process algebra expresses a system model as a flat set of the process defining equations without any hierarchy. In an object oriented design, the system behavior is partitioned into classes. Furthermore, inheritance allows for a hierarchy of classes. We implement simple schemas called classes in order to achieve the partitioning and (inheritance) hierarchy.

Thus, in the following a *class* is a scheme that can be compiled to a set of process defining equations. A class C may inherit parts of its characteristics (e.g. its behavior) from a *superclass*, and in that context C is referred to as the *subclass*. A state inheritance sequence

$$S \mapsto [A_1, A_2, ..., A_n]$$

is a declaration saying that state S in the superclass is refined by states $A_1, A_2, ..., A_n$ in the subclass where A_1 is the default state (i.e. the substate entered when entering the superstate S). When compiling the class to process equations, the inheritance sequence describes how the defining equations from the superclass should be used. Thus, we implement a simple form of inheritance as refinement. The syntax used for defining classes below is

$$N = (S, I), D$$

where N is the name of the class, S is the name of the superclass (if any), I is the set of state inheritance sequences and D is a set of process defining equations. If there is no superclass we write $N = (), D$.

Example

Lacking formal tools, we outline the approach by an example. In the following we define two classes C_1 and C_2, where C_2 refines the state A in C_1 with states C and D. We say that states C and D refine state A.

$$\begin{aligned}
C_1 = (), \{ & \\
& A \stackrel{def}{=} b.B \\
& B \stackrel{def}{=} a.A \\
\} & \\
C_2 = (C_1, \{A \mapsto [C, D]\}), \{ & \\
& C \stackrel{def}{=} d.D \\
& D \stackrel{def}{=} c.C \\
& B \stackrel{def}{=} e.D \\
\} &
\end{aligned}$$

Now, C_2 may be compiled to the following process equations.

$$C_2{:}C \stackrel{def}{=} b.C_2{:}B + d.C_2{:}D$$
$$C_2{:}B \stackrel{def}{=} a.C_2{:}C + e.C_2{:}D$$
$$C_2{:}D \stackrel{def}{=} b.C_2{:}B + c.C_2{:}C$$

The outgoing transitions from A become outgoing transitions from all refining states, while the incoming transitions are moved from the refined state to the first of the refining states. If there are transitions from the same state in both super- and subclass, they are joined as indeterministic choice, as with state B and transitions $a.A$ and $e.D$. The states are prefixed with the class name to avoid name space clashes.

4.2 Statecharts

Since both processes and statecharts have a transition system semantics, the mapping is straightforward once the semantics of the statecharts is fixed. We use a handshaking semantics of the statecharts, because of expressivity and domain properties as described in [Lawesson, 2000]. We define the semantics via our process language by providing a mapping from statecharts to processes. The mapping is rather straightforward since we restrict ourselves to statecharts without history states – essentially making the state chart equivalent to an automata without hierarchy, see for example [Lilius and Porres, 1999]. The process algebra example in Figure 2 could represent a slightly improved version[2] of the automata in Figure 4 with structure information (i.e. the states S, $Main$ and $Setup$) added.

4.3 Default behaviors of class diagrams

Since a class diagram in general does not contain behavioral information in terms of statecharts, we may introduce a superclass called *Propagator* that encapsulates the behavior of being able to propagate errors as well as reporting errors to the log, and a subclass *Breakable* that is a propagator that can introduce errors by the transition *crit*. The idea is to let all classes inherit from *Propagator*, and then refine with behavioral models when available, and use *Breakable* for classes that may give rise to critical actions but where a behavioral model is missing. The definition of *Propagator* and *Breakable* are given in Figure 5.

The paths of error propagation between classes is computed by using information about dependencies between classes in the class diagrams (as in [Larsson, 1999; Larsson *et al.*, 2000]), and then reflected in the $Failed(\mathbf{x})$ state that models error propagation.

Acknowledgments

This work has been financially supported by VINNOVA's Center of Excellence ISIS – Information Systems for Industrial Control and Supervision. We are also grateful for the cooperation with ABB Robotics, and in particular Magnus Larsson.

[2] In the process algebra version of this example a $\overline{req}$ also provides a reference to the caller, so that the called object know where to address the answer. This makes the example more similar to a real system, but in this completely symmetric case when the objects run the same code without parameters other than references to each others, the semantics remains the same.

References

[Clarke *et al.*, 1999] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Console *et al.*, 2000] L. Console, C. Picardi, and M. Ribaudo. Diagnosis and Diagnosability Analysis using PEPA. In *Proc. 14th European Conference on Artificial Intelligence*, pages 131–136. IOS Press, 2000.

[Cordier *et al.*, 2001] Marie-Odile Cordier, Laurence Rozé, and Yannick Pencolé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *Proc. 12th Intl Workshop on Principles of Diagnosis, DX01*, 2001.

[Hamscher *et al.*, 1992] W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers, 1992.

[Larsson *et al.*, 2000] M. Larsson, I. Klein, D. Lawesson, and U. Nilsson. Fault isolation in object oriented control systems. In *4th IFAC Symposium On Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS 2000)*, June 2000.

[Larsson, 1999] M. Larsson. *Behavioral and Structural Model Based Approaches to Discrete Diagnosis*. PhD thesis no 608, Department of Electrical Engineering, Linköping University, 1999.

[Lawesson *et al.*, 2001] Dan Lawesson, Ulf Nilsson, and Inger Klein. Model-checking based fault isolation in uml. In *Proc. 12th Intl Workshop on Principles of Diagnosis, DX01*, pages 103–110, 2001.

[Lawesson, 2000] Dan Lawesson. *Towards Behavioral Model Fault Isolation for Object Oriented Control Systems*. Licentiate thesis no 863, Dept of Computer and Information Science, Linköping University, 2000.

[Lilius and Porres, 1999] J. Lilius and I. Porres. The semantics of UML state machines. Technical Report 293, Turku Centre for Computer Science, 1999.

[Milner, 1989] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Object Management Group, 1999] Object Management Group. OMG Unified Modeling Language Specification, version 1.3, June 1999.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[Sagonas *et al.*, 1994] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94)*, pages 442–453, 1994.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete-Event Systems. *IEEE trans. Automatic Control*, 40(9):1555–1575, 1995.

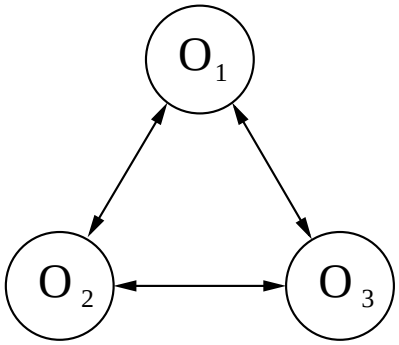


Figure 3: A global picture of the example system consisting of the objects o_1, o_2 and o_3. Each object has behavior as described in Figure 4.

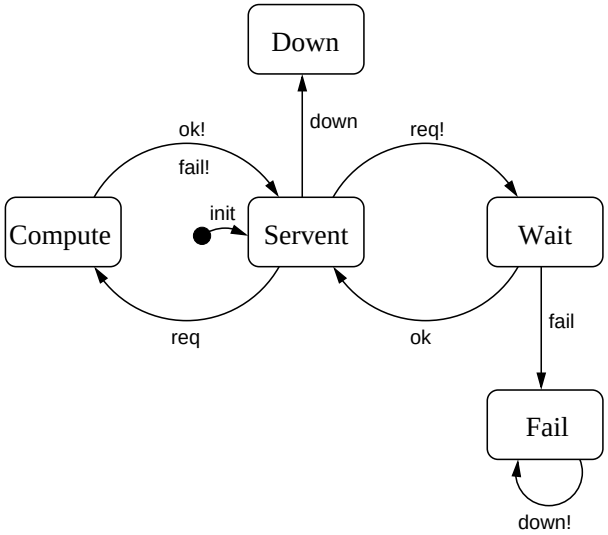


Figure 4: An automata describing a peer-to-peer system. Sending actions are suffixed with ! and the rest of the actions are receiving actions. There are no internal actions in this automata.

$$
\begin{array}{l}
Propagator_n = (), \{ \\
\quad Main \quad \stackrel{def}{=} \quad init(\mathbf{x}).OK(\mathbf{x}) \\
\quad OK(\mathbf{x}) \quad \stackrel{def}{=} \quad fail().Failing(\mathbf{x}) \\
\quad Failing(\mathbf{x}) \quad \stackrel{def}{=} \quad log.Failed(\mathbf{x}) + nolog.Failed(\mathbf{x}) \\
\quad Failed(x_1, x_2, ..., x_n) \quad \stackrel{def}{=} \quad x_1{:}fail().Failed(\mathbf{x}) + ... + x_n{:}fail().Failed(\mathbf{x}) \\
\}
\end{array}
$$

$$
\begin{array}{l}
Breakable = (Propagator, \{\}), \{ \\
\quad OK(\mathbf{x}) \quad \stackrel{def}{=} \quad crit.Failing(\mathbf{x}) \\
\}
\end{array}
$$

Figure 5: Definitions of the classes *Propagator* and *Breakable*

Distributed Diagnosis of Networked, Embedded Systems

James Kurien **Xenofon Koutsoukos** **Feng Zhao**

Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 93404
jkurien,koutsouk,zhao@parc.com
Phone: 1-650-812-4340
Fax: 1-650-812-4334

Abstract

Networked embedded systems are composed of a large number of physically distributed nodes that interact with the physical world via a set of sensors and actuators, have their own computational capabilities, and communicate with each other via a wired or wireless network. Monitoring and diagnosis for such systems must address several challenges caused by the distribution of resources, communication limitations, and node and link failures. This paper presents a distributed diagnosis framework that exploits the topology of a physical system to be diagnosed to limit inter-diagnoser communication and compute diagnoses in an anytime and any information manner, making it robust to communication and processor failures. The framework adopts the consistency-based diagnosis formalism and develops a distributed constraint satisfaction realization of the diagnosis algorithm. Each local diagnoser first computes locally consistent diagnoses, taking into account local sensing information only. The local diagnosis sets are reduced to globally consistent diagnoses through pairwise communications between local diagnosers. The algorithm has been successfully demonstrated for the diagnosis of paper path faults for the Xerox DC265 printer.

Introduction

Our diagnostic research is motivated by existing and emerging applications of networked, embedded systems. In such systems the physical plant is composed of a large number of distributed nodes, each of which performs a moderate amount of computation, collaborates with other nodes via a wired or wireless network, and is embedded in the physical world via a set of sensors and actuators. Examples include distributed sensor networks (Chu, Haussecker, & Zhao 2001), complex electromechanical systems with embedded controllers (Zhao *et al.* 2001), data networks, smart matter systems (Jackson *et al.* 2001), and ad-hoc wireless networks of consumer devices. Such systems present a number of interesting new challenges for diagnostic systems. A moderate amount of computation is potentially available, but it is partitioned into embedded chunks that range in size from tiny, in the case of smart dust sensor motes (Kahn, Katz, & Pister 1999) to moderate in the case of consumer devices. Communication between nodes is available, but may involve unreliable delivery, power-constrained wireless networks, or large, complex topologies requiring multiple hops to connect two arbitrary nodes. Finally, nodes might leave the network dynamically and nodes of a previously unseen type might join in their place.

In this paper, we consider how we might apply techniques from model-based diagnosis to these types of problems. In general, traditional model-based techniques are centralized. They assume that the diagnostic algorithm is run on a single processing unit that has access to observations from all sensors in the physical plant. In the next two sections of the paper, we briefly discuss centralized, model-based techniques and discuss how they cause scalability, robustness and reconfigurability problems if employed directly on networked, embedded systems. We then present a set of useful properties for diagnostic algorithms for such systems. In the fourth section, we present a simple formulation for diagnosis of discrete, distributed systems in order to motivate discussion and map the formulation onto distributed constraint satisfaction and distributed constraint optimization. We next propose an algorithmic framework for distributed diagnosis that operates in an anytime manner and is robust to communication and processor failures. We discuss the communications requirements for the framework and compare performance results for one instantiation of the distributed diagnosis framework against a centralized diagnoser. In the related work section, we discuss why existing distributed constraint satisfaction and optimization algorithms are not well suited for distributed diagnosis of networked, embedded systems. We finally discuss two open areas for future work. The contributions of this paper are that it illustrates the interesting features of networked, embedded systems that make them challenging for traditional model-based diagnosis techniques, it presents a simple formulation of the distributed diagnosis problem for these type of systems and relates it to distributed constraint satisfaction and optimization, it presents a class of robust, anytime algorithms for performing diagnosis, and it illustrates preliminary diagnostic results on a model of a real physical system with comparisons to an existing centralized diagnoser.

Model-based Diagnosis

The objective of diagnosis is to determine the state of a physical plant such as a printer, aircraft or network, based upon the current sensor readings from the plant and prior knowledge about the plant's structure and behavior. In order for the diagnosis to be useful for on-line control of the plant,

accurate diagnoses must be generated in a time-critical manner using the available computational resources. In most model-based diagnostic techniques, prior knowledge about the physical plant consists of a description of the behavior of each component of the plant, including normal and faulty behaviors, and the interconnections between components (Hamscher, Console, & de Kleer 1992). Partial observability presents the main challenge of diagnosis. Faults in a component may not be directly observable, and instead may cause changes in the behavior of the plant that propagate through several components before becoming observable at a sensor. To perform diagnosis, the component models are combined into a global store, observations are obtained from the physical plant, and a centralized algorithm is applied to find a system-wide diagnosis. We believe this very abstract description captures many diagnostic formalisms, including logic-based formalisms such as those based upon (de Kleer & Williams 1989) or (Reiter 1987), bond graphs (Mosterman & Biswas 1997) and many others. Throughout this paper we will use a formalism and examples consistent with GDE (de Kleer & Williams 1987) and its descendants, keeping in mind the general properties of centralized, model-based diagnosis that are at issue.

Figure 1 on the next page schematically illustrates a small model for the kind of traditional problem we might attack with a model-based diagnoser. The 24 boxes represent rollers, gears, motors, sensors and other devices in a printer paper path. For example, the acRoll acquires a sheet of paper from the paper tray and transports it to the feedRoll, driven by the acBelt. We have developed a simple diagnostic application for this paper path system using L2 (Kurien & Nayak 2000), a centralized, GDE-style diagnoser developed by NASA. Each component is modeled by finite state machine augmented with finite domain variables that describe its behavior. Arcs between components in Figure 1 represent interactions between components, for example conveying that the acRoll receives an angular velocity from the acBelt. This is represented by a constraint between the corresponding variables. There are five sensors that report the time of arrival of a sheet of paper at various points in the paper path.

To perform diagnosis with L2 and this model, observations as to when or if the paper arrived at various points in the path would first be obtained from the printer's sensors via its internal data bus and sent to an external processor running L2. The values would be discretized and assigned to the corresponding variables in the constraint system. A constraint optimization algorithm would be applied to the updated constraint system to find assignments to the variables that are consistent with the observations. Such an assignment might represent that the paper was late at the first sensor because the feedMotor is slow, slowing down both the acRoll and the feedRoll. This information could then be used to perform maintenance, or in systems with redundancy, to reconfigure the system for robust control. In addition to this small demonstration, we have applied similar diagnostic techniques to spacecraft (Bernard *et al.* 1998), chemical processing plants (Goodrich & Kurien 2001), scientific instruments, and other electromechanical systems to assist in robust control.

Given a set of component models and a centralized diagnoser C:

1. C combines the component models in a central store
2. Observations are collected from the physical system
3. C computes the system-wide diagnoses

Figure 2: Centralized Diagnosis of a Centralized System

Given a set S of currently connected components and a centralized diagnoser C:

1. $\forall S$, S forwards its component model to C
2. C combines the component models in a central store
3. $\forall S$, S forwards its observations to C
4. C computes the system-wide diagnoses
5. $\forall S$, C projects the variables of interest to S from the diagnoses and forwards them to S

Figure 3: Centralized Diagnosis of a Networked System

Challenges of Monitoring and Diagnosing Networked, Embedded Systems

Suppose we would like to perform diagnosis for a reconfigurable, networked, embedded system. Such systems are constructed such that each component is locally controlled by a small, embedded processor which coordinates with other processors via a potentially unreliable network. In addition, components and their processors might be unplugged and replaced with upgraded versions from time to time. Examples of such systems are ad-hoc wireless networks, modular robots, and more conventional systems such as intranets. Even traditional electro-mechanical systems such as printers and automobiles now contain on-board networks, embedded sensing and tens or hundreds of local controllers.

We can provide diagnostic information to the local controllers of such a system using centralized diagnosis via the process outlined in Figure 3. First, a centralized, global diagnosis problem is created by assembling a global model of the components within a centralized diagnoser. The observations are centrally collected and a diagnosis or set of diagnoses are computed by the centralized diagnoser. Aspects of the centralized, global diagnosis are then be distributed back to the local controllers.

This approach makes several assumptions. First, there must exist a processor large enough to store the global diagnostic model and run the centralized diagnostic algorithm. If this processor fails, it must be acceptable for no further diagnoses to be generated. Second, there must exist a central bus or buses with sufficient capacity to forward all data needed for diagnosis to the central processor. If a bus fails, the data needed to diagnose and recover for the failure must be located on the near side of the bus with respect to the diagnostic processor, or it must be acceptable for no further diagnoses to be generated for the bus and the far side compo-

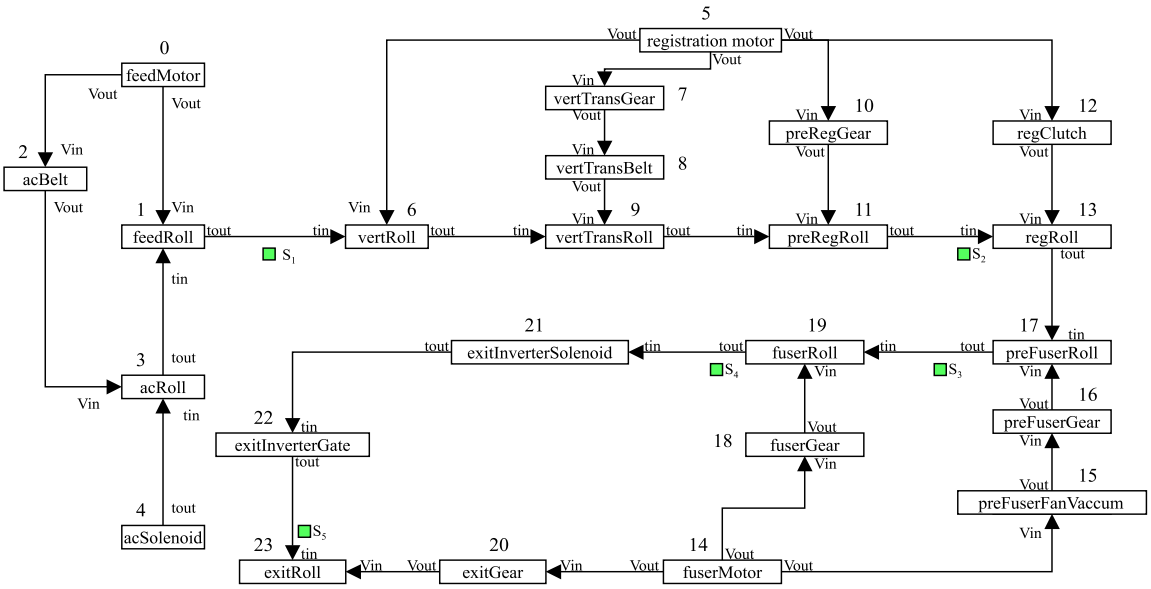


Figure 1: Paper Path Model in Xerox DC265ST Printer

nents. Finally, the set of components to be diagnosed must be represented using the same formalism, and in most applications must be known a priori.

With networked, embedded systems, all of these assumptions may be false. Each processor in the plant may be quite small. If a processor fails, we may require the components attached to remaining processors to continue operating in a full diagnosis and control cycle. If the network is bifurcated, we may require that each half of the plant continues operations to the extent possible and works to resolve the failure with the locally available information. New components might join into the network at any time by publishing their capabilities such as described by JINI (Sun Microsystems Inc 1999).

These issues suggest an approach wherein we do not artificially centralize the problem but allow a local diagnoser to be associated with each system processor. Each local diagnoser finds a partial diagnostic solution using a model of the locally controlled portion of the plant and the locally available observations. Communication is then required to refine the partial diagnostic solution into a diagnosis, in effect making use of observations and models local to other diagnosers. We next suggest themes for dividing and coordinating the diagnostic process to maximize scalability, robustness and reconfigurability, based upon our experience with both diagnosis and networked, embedded systems.

- **Scalability**

  Dividing the diagnostic problem among local diagnosers allows us to apply multiple processors and potentially address computational scalability problems caused by the small processors we may encounter in some embedded systems. To address communication scalability issues, we seek to exploit the topology of the physical plant. We would like to arrange that two local diagnosers need communicate only if the subsystems of the physical plant they correspond to are physically interconnected or share data. Thus the structure of our diagnostic architecture will mimic the physical topology of the plant being diagnosed. For the type of engineered systems that are typically amenable to diagnosis, physical scalability is accomplished by modularizing subsystems and connecting them through fairly narrow physical interfaces (power, data, physical support). By respecting these interfaces, we expect our communication needs for moving diagnostic data to scale as well as the underlying physical plant.

- **Robustness**

  A diagnostic architecture must be extremely robust to failure and able to operate in an anytime and any information manner. This can be accomplished with refinement. We would like to arrange that each diagnoser locally produce a superset of the diagnoses that a global diagnoser would produce for the local components. Communication with other diagnosers is then used only to prune the local diagnosis set. This yields several important properties. First, the diagnostic process can be interrupted at any time and each diagnoser will contain the true diagnosis plus possible imposters. This is an important safety feature in domains where taking action based upon a false negative can cause serious harm. Second, if diagnosers fail, then the remaining diagnosers will simply produce coarser (more conservative) estimates of the possible states of their components. Third, if the system is bifurcated due to a communication failure, then each half will produce all diagnoses consistent with the reachable diagnosers and any state of the other half of the system.

- **Reconfigurability**

  A side effect of employing local diagnosers that commu-

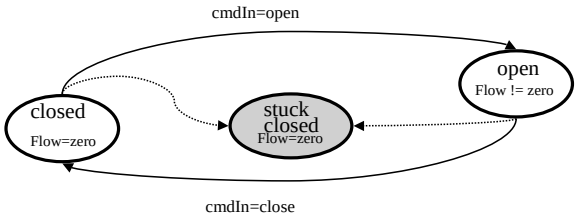


Figure 4: Automaton Representing A Single Valve

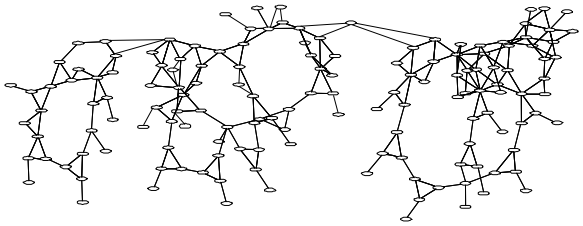


Figure 5: Variable Connectivity In a Global Model

nicate via opaque interfaces defined by the physical plant is natural support for modular or reconfigurable plants. Intuitively, a connected subset of the components of Figure 1 may be disconnected from the plant and replaced by new hardware with a different model, so long as the physical and diagnostic interface at the point of disconnection is maintained. In addition, this opens the possibility of participation by different implementations of the same diagnostic algorithm or even different algorithms participating in a diagnosis. The latter would of course require an interface that is semantically meaningful for all participating diagnosers. However, even the former capability might be useful in allowing vendors of components that are likely to be connected (e.g. data network components or power distribution components) to create diagnosers that can collaborate.

We believe these properties will be of interest as we begin to investigate applications involving very large numbers of embedded processors communicating via networks. In the next section we introduce a simple formalization that will allow us to discuss algorithmic directions for type of problem.

Centralized Formulation

Our approach to distributed qualitative diagnosis follows the centralized diagnostic formalism developed in (de Kleer & Williams 1989) and extended in (Williams & Nayak 1996) and (Kurien & Nayak 2000). To motivate our distributed algorithms, we begin with a brief overview of the centralized technique, summarized from (Kurien & Nayak 2000). Suppose we would like to diagnose the state of a single component, a valve, which is qualitatively modeled via the finite state machine illustrated in Figure 4. We refer to each possible discrete state of a component as a $mode$. A valve v has three modes, $open$, $closed$, and $stuckClosed$. The behavior of the flow of the valve within each mode, which has the discrete domain $\{zero, nonzero\}$, can be captured with the following propositional formulae.

$$\begin{array}{lcl} v = open & \Rightarrow & flow_v = nonzero \\ v = closed & \Rightarrow & flow_v = zero \\ v = stuckClosed & \Rightarrow & flow_v = zero \end{array}$$

If $flow_v$ is observable from the physical plant, we will refer to this variable as an *observation*. In order to represent the non-determinism of the automaton within a propositional framework, the encoding introduces an *assumption* variable a. Intuitively, a_v represents the choice that Nature makes as to whether valve v will behave normally or experience a failure when it is commanded. The transition portion of the automaton can thus be captured by the following formulae.

$$\begin{array}{l} a_v = normal \Rightarrow \\ \quad \begin{array}{lcl} v_t = closed \wedge cmd_t = open & \Rightarrow & v_{t+1} = open \\ v_t = closed \wedge cmd_t \neq open & \Rightarrow & v_{t+1} = closed \\ v_t = open \wedge cmd_t = close & \Rightarrow & v_{t+1} = closed \\ v_t = open \wedge cmd_t \neq close & \Rightarrow & v_{t+1} = open \\ v_t = stuckClosed & \Rightarrow & v_{t+1} = stuckClosed \end{array} \\ a_v = stick \Rightarrow v_{,t+1} = stuckClosed \end{array}$$

Intuitively, the diagnostic task is to find a set of assignments to the assumptions, here $\{a_v\}$, such that the model is consistent with the observations, here $\{flow_v\}$. For example, suppose $v_t = closed$, we command the valve open, represented by $cmd_t = open$. The plant assigns O as $flow_v = zero$. The only consistent assignment to a_v is $a_v = stick$ and we diagnose valve is stuck closed. If we wish to model multiple automata, we introduce a mode and assumption for each automaton and compile all automata into a set of formulae that may share variables. For example, two valves in series share the same flow. Figure 5 visualizes the compilation of the device constraints into a global constraint system model. Each node represents a finite domain variable. Two nodes are connected by an edge if the two variables appear in a constraint together, denoting that the possible values of the variables are related by interacting together in some physical process or the transmission of data. Note that a realistic model such as that of Figure 5 contains many observations and assumptions, and many assignments may be consistent. More formally, let A denote the set of assumptions, O denote the set of observations, and F denote the formulae describing the plant. Given an assignment Ω to O created by observing the plant, a diagnosis D is an assignment to A such that the following propositional formula is consistent:

$$\wedge_{a_i \in A}(a_i = d_i) \ \wedge_{o_j \in O} (o_j = \omega_j) \wedge F.$$

To perform diagnosis over multiple components, we must find an assignment to each a that renders the set of formulae consistent with all observations. Intuitively, we assign the observations reported by the physical plant, Ω to the variables of the graph corresponding to observations, O, then reassign the assumption variables, A until the constraint system illustrated in Figure 5 becomes consistent. Thus in this diagnosis framework, diagnosis can be viewed a constraint satisfaction problem.

A second diagnostic task is to find the most likely diagnoses. For each assumption assignment we can associate the prior probability of the even the assumption represents.

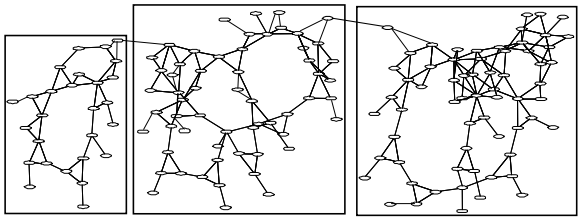


Figure 6: Partition Among Three Diagnosers

Thus, P(a_v=stick) denotes the prior probability of the valve sticking. Assuming conditional independence, the probability of a diagnosis is defined as follows.

$$P(D) = \Pi_{a_i \in D} P(a_i = d_i)$$

Given multiple components, we must find the assignment to each a that renders the set of formulae consistent with all observations such that the probability of the assignment is maximal. Intuitively, we assign the observations reported by the physical plant, Ω to the variables of the graph corresponding to observations, O, then choose among the possible reassignments of assumption values to assumption variables, A, until the constraint system illustrated in Figure 5 becomes consistent. The choice of which assumption to reassign and to which value to assign it is based upon the probability of the possible assignments. In this case, diagnosis can be viewed as a constraint optimization problem.

Distributed Diagnosis

In this paper, we propose splitting the global diagnostic process into a number of cooperating local diagnostic processes. In order to distribute the problem, we divide the global diagnoser which produces assignments to A into a set of local diagnosers which make assignments to subset of A. Intuitively, we partition the edges of Figure 5. If a node is connected to edges in more than one partition, it is replicated and the partitions must reach consensus on its value. More formally, a local diagnoser L is described by (F_L, V_L, A_L, O_L, R_L) where F_L is the subset of F assigned to L, V_L denotes the set of variables that appear in F_L, A_L denotes $A \cap V_L$, O_L denotes $O \cap V_L$ and R_L denotes the union of $V_L \cap V_M$ over all other diagnosers M. Figure 6 illustrates a possible partitioning of the constraint graph of Figure 5. The slightly darker nodes indicate the members of R_L, shared variables that have been replicated. Given a fixed number of diagnosers or the maximum number of constraints a diagnostic processor can accommodate, we can use a graph partitioning algorithm (Sanchis 1989) to find a partitioning of the graph that attempts to minimize R_L for each diagnoser.

Our approach to finding consistent diagnoses in a distributed fashion is refinement based. Intuitively, each local diagnoser finds the diagnoses for the locally modeled component that are consistent with the constraints of the local model and the local observations. This is a superset of the diagnoses for the local components that are consistent will all constraints and observations. Each local diagnoser then communicates with directly other diagnosers to further reduce the set of consistent diagnoses for the local components. We would like that the diagnoses start with a superset of the globally consistent diagnoses and move toward only the globally consistent diagnoses. We define the relationships $conservative$ and $feasible$ between the diagnoses produced by a global diagnoser and the diagnoses produced by a local diagnoser. A local diagnosis set D_L is conservative with respect to the global diagnosis set D_G if

$$\forall \delta_G \in D_G \; \Pi_{A_L}(\delta_G) \in D_L$$

where Π is the projection operator. That is, the assignments made to the assumptions local to L by a global diagnosis must also be made by a local diagnosis. A local diagnosis set D_L is feasible if the assignments made to the local assumptions are contained in a consistent global diagnosis. More formally,

$$\forall \delta_L \in D_L \; \exists \delta_G \in D_G : \Pi_{A_L}(\delta_G) = \delta_L.$$

1. Given observation set Ω, if $o_j \in O_L$, assign $o_j = \omega_j$ in L.
2. $\forall L$, if $O_L \neq \emptyset$, compute all assignments to $A_L \cup R_L$ s.t. $\wedge_{o_j \in O_L} (o_j = \omega_j) \wedge_{a_i \in A_L} (a_i = d_i) \wedge_{r_i \in R_L} (r_i = \rho_i) \models F_L$
3. For each $r \in R_L$, for each other diagnoser M, if $r \in V_M$ send all R_L assignments to M.
4. In each such M, compute all assignments such that $\wedge_{r_i \in R_L} (r_i = \rho_i) \wedge_{a_k \in A_M} (a_k = d_k) \wedge_{r_k \in R_M} (r_k = \rho_k) \models F_M$
5. If the consistent R_M assignments decreased in step 4, return to step 3, substituting M for L.

Figure 7: Consistency-based, Anytime Diagnosis

Incremental Consistency

We next discuss an algorithmic framework for incrementally revising a set of conservative diagnoses into a set feasible diagnoses in a robust, anytime, distributed manner, followed by results from one particular instantiation of this framework. The approach of the algorithmic framework is similar in spirit to Waltz's algorithm (Waltz 1975). Each set of diagnoses is monotonically reduced toward a feasible set as a side effect of spreading consensus on the value of variables shared between diagnosers. The algorithm is illustrated in Figure 7.

The algorithm operates by incrementally reducing the possible assignments to A_L for all L, first by introduction of observations and second by communication between diagnosers. Each local diagnoser begins with a conservative local diagnosis set in A_L. Typically this would be all possible diagnoses, which can be implicitly captured by an appropriate encoding of the constraint set F_L. In Step 1, observations are assigned in every diagnoser which has constraints involving an observation. In Step 2, the observation assignments are used to compute all assignments to $A_L \cup R_L$ that are consistent with F_L and the observations received by L. Note that the projection of A_L from these assignments is a

conservative diagnosis set. Intuitively, suppose an assignment to A_L appears in a global diagnosis but is not computed by L. If it is not computed, it must be inconsistent with F_L and the assignments to O_L. It is therefore inconsistent with F and the assignments to O, and could not appear in a global diagnosis. In Step 3, the assignments to R_L are projected out of the consistent assignments of L and forwarded to each other diagnoser M that references these variables. In Step 4, M eliminates a subset of its assignments that are not feasible. Intuitively, an assignment α to A_M is not feasible if there is no assignment to A containing α that is consistent with F and O. If α constrains a variable in R_L to have a value that was not received from L, then α is inconsistent with all consistent assignments to A_L. Thus, each time Step 4 is performed, infeasible assignments to A_M are eliminated. Each diagnoser begins with a conservative set of assignments to A_L, and as rounds of communication are performed, the local diagnoses are moved toward feasibility in an anytime manner. Per Step 5, the algorithm continues as long as consistent assignments are eliminated. In the worst case, each loop would eliminate one of an exponential number of possible assignments.

Note that we have described the algorithm to propagate sets of assignments that remain consistent in one local diagnoser to to other diagnosers in which the assigned variables appear. More generally, we may propagate any information that allows remote diagnosers to restrict the domain of a variable based upon inference performed in the local diagnoser. Examples include assignments that cannot be made because of constraints within one diagnoser (nogoods), assignments that must be made, or sets of possible assignments to a variable that remain consistent. Note also that this algorithm is not complete with respect to distributed constraint satisfaction. Intuitively, suppose we have two local diagnosers, one containing only the constraint $A \vee B$ and the other containing only the constraint $\bar{A} \vee B$. Neither can constrain and propagate the value of B, though B must be true. This same restriction applies to the centralized constraint satisfaction technique used in L2, so we do not believe it presents a significant drawback. The related work section contains further details on the relationship between distributed diagnosis and distributed constraint satisfaction and why we believe an incomplete algorithm is sufficient.

Communication Requirements

When presented with a networked, embedded system, we may perform centralized diagnosis of the distributed system by transmission of observations or distributed diagnosis of the distributed system by transmission of intermediate results. Choosing distributed diagnosis allows us to trade communication bandwidth for reduced processor requirements, increased robustness and greater reconfigurability. In this section, we examine how the communication requirements of the distributed, incremental diagnosis algorithm compare to a centralized approach. We first consider the communication requirements of the centralized procedure shown in Figure 3. Let n be the number of components and s be the number of components with sensors. In Step 3 of the procedure, each of s components forwards its observations to C. In Step 5, C forwards the diagnostic results to each of n components. Assuming all observations from a single component can be sent in a single message, Figure 3 requires s point to point messages to C and one broadcast message from C to all n components

We now consider the communication requirements for the distributed algorithm of Figure 7. This algorithm performs distributed diagnosis by exchanging messages that refine the value of shared variables across local diagnosers. Let v be the number of variables that are shared, and r be the average number of diagnosers that share each variable, and m be the average number of messages exchanged that involve a given variable. For example, if each local diagnoser uses unit propagation, it can send messages specifying that a variable must have a certain value or cannot have a certain value, but no messages specifying disjunctions between assignments. Thus m is bounded by the size of the largest domain of a shared variable. The increase in messages created by moving to the distributed diagnoses technique is given by the ratio

$$\alpha_1 = \frac{vrm}{s+1}.$$

Note that counting the number of messages exchanged is not sufficient to determine the cost of communication. In many applications, such as wireless networks with limited energy or bandwith, the number of packets transmitted is a critical cost measure. Network topology will determine the number of packet transmissions or hops required to deliver a message. In many applications, each node in a network is connected to a small number of neighbors. Point to point communication is implemented by multiple hops between neighbors, and a broadcast is implemented by flooding the network. Let h_c be the average distance in hops between a node with a sensor and the centralized diagnoser. Let h_v be the average number of hops between nodes that share a variable. In general, the change in the total number of packet transmissions required by decentralizing the problem is determined by

$$\alpha_2 = \frac{vrmh_v}{sh_c + n}$$

Intuitively, packet transmission for the centralized diagnoser scales with the size and width of the network, while the decentralized approach scales with the number of constraints that cross network components. Note that if the network topology reflects the physical interactions of the components, it is likely the case that $h_v < h_c$. Thus we can construct wide networks with very localized interactions for which centralized diagnosis requires more packet transmissions than decentralized diagnosis, though we do not expect this to be the case in practice. In addition to total packet transmission, we may further refine our cost measure to include the maximum number of packets transmitted by any link in the network. This determines the minimum bandwidth or power storage a network node must support. The ratio α_2 does not capture that in the centralized case, all messages must pass through network links connected to the central diagnoser. This drives up the minimum capabilities of a network node in relation to distributed diagnosis where message sources and destinations are more evenly distributed

Independent Faults In	L2		Distributed		
	Diag	Time	Spread	Diag	Time
First module	6	0.02	9	21	0
Two modules	12	0.18	14	49	0
Three modules	84	13.28	20	343	0.05
All modules	108	27.08	24	637	0.22

Table 1: Comparison of distributed diagnoser and L2

through the system. We are currently defining a diagnostic model for a distributed sensor network in addition to available models of more traditional electro-mechanical systems in order to better characterize the communication requirements of both distributed and centralized algorithms

Results

To implement the distributed diagnosis algorithm described above, each local diagnoser could represent its conservative diagnosis set as a partial assignment in a GDE-style diagnoser, a relational table, a binary decision diagram and so on, so long as the representation can be efficiently pruned when an observation or neighboring diagnoser decreases the range of a variable. Ideally, we would like to test a centralized diagnoser against a set of local diagnosers that compute and represent diagnoses in the same manner. For these preliminary results, we present the performance of the centralized L2 diagnoser against a distributed diagnoser that takes advantage of the small local model size enabled by distributing the problem. PARC intern Rong Su implemented the distributed algorithm using finite-state automata to prune inconsistent assignments to V_L (Steps 2 and 4 of Figure 7) and a distributed consensus algorithm (Steps 3 and 5) shown to converge to feasible diagnoses (Su *et al.* 2002). Table 1 compares performance with L2 on the paper path model. The first three columns are the name of the diagnostic scenario, the diagnoses found by L2, and the time required. Since the physical plant has few sensors, the number of consistent diagnoses grows with the complexity of the scenario. The fourth column is the number of local diagnosers reached via Step 3 of the algorithm, out of 24. The fifth column lists the number of diagnoses found by the distributed algorithm. Note that the FSA-based algorithm finds more diagnoses than L2. L2 is conflict based, and thus postulates only those failures that can eliminate a discrepancy between an expected observation and the observation received from the plant. The FSA-based algorithm finds all consistent failures, including those that would be indistinguishable from proper operation of the plant. The sixth column is the time to compute the diagnoses, demonstrating the dramatic speed advantage, on this model, of computing feasible local diagnoses via a pre-compiled FSA representation then determining consistent combinations versus global, on-line inference. The current implementation runs each local diagnoser serially on a single processor, and we believe a parallel implementation will provide a greater speed advantage.

Related Work

A diagnoser for a networked, embedded system may be centralized, decentralized or distributed. Work in centralized diagnosis may be applied by collecting models and observations from the networked components of the physical plant and appling a centralized algorithm. As described in the third section of this paper, this raises robustness and scalability issues that must be addressed. Rish, Brodie and Ma, for example, attempt to increase the efficiency of a centralized diagnostic procedure for a distributed network of computers using an approximate representation and carefully designed active probing of the distributed system (Rish, Brodie, & Ma 2002). In decentralized diagnosis, e.g. (Debouk, Lafortune, & Teneketzis 2000), local diagnosers communicate with a coordination process that assembles a global diagnosis. The coordination process of decentralized approaches are still subject to robustness and scalability issues. We are therefore pursuing an approach of distributed diagnosis, similar to (Baroni *et al.* 1999), where there is no centralized control structure or coordination process. Each local diagnoser communicates directly with other diagnosers.

We have formulated the the distributed diagnostic process as a distributed constraint satisfaction problem (DCSP). Since many problems in scheduling, resource allocation, and hardware design can be formulated as constraint satisfaction problems, the distributed constraint satisfaction problem has received a large amount of attention. Yokoo and Hirayama provide an excellent overview (Yokoo & Hirayama 2000) of algorithms for solving DCSP's. These existing algorithms do not meet our needs for two reasons. First, the great majority of the algorithms are formulated assuming the computational nodes and network connecting the nodes are reliable, and that all messages sent between nodes arrive in the order sent. For diagnosis of networked, embedded systems, we seek specific guarantees of behavior in response to the loss of computing nodes or bifurcation of the network. Second, the majority of DCSP algorithms are designed to solve general discrete constraint satisfaction problems, such as the graph coloring problem. The ability to solve general CSP problems requires features that complicate distribution, such as backtracking on choices for variable assignments. In practice, centralized diagnosers are able to find consistent diagnoses using incomplete, backtrack-free procedures such as unit propagation. This difference arises because the constraints we generate from finite state models such as illustrated in Figure 4 tend to be closer to Horn clauses in structure than general discrete constraints and diagnosis may use observation values asserted by the physical plant to drive constraint processing. We therefore expect a distributed diagnoser acting upon the same models should be able to use less powerful inference methods than full constraint satisfaction. While we have encountered full DCSP algorithms that allow some fault tolerance, such as the Mozart system (Roy 1999), and some simpler constraint processing methods that assume reliable, fully connected networks, such as distributed arc consistency (Nguyen & Deville 1998), we have not yet encountered an algorithm that is sufficiently narrow in scope and robust to failures.

Future Work

A number of issues remain for future work. The issue of how to use knowledge of the prior probability of failures to avoid computing all consistent diagnoses has been explored but not solved. The algorithm of Figure 7 also does not take into account any information about the likelihood of failures. We may of course find the set of globally consistent diagnoses and compute the probability of each by assuming conditional independence of the failures, as described above. However, rather than computing the probabilities of all consistent diagnoses, we might wish to avoid generating unlikely diagnoses given we have generated a sufficient number of consistent, likely diagnoses. Conflict-directed, best-first search (de Kleer & Williams 1989) is a centralized, discrete constraint optimization algorithm that is specialized for diagnosis. It efficiently enumerates consistent assignments to a set of propositional variables in order of their cost, or in this case enumerates diagnoses in order of their prior probability. Intuitively, it operates by starting with the highest probability assignment to the assumptions, the case where no failures have occurred. It substitutes a minimal cost assignment to an assumption with a non-minimal cost assignment only when a conflict between an observation value assigned by the plant and the value predicted by the current assumption assignments occurs. Our current direction in developing a distributed analog is to begins with a maximum likelihood (*e.g.*,no failure) assignment to A_L within each diagnoser L, which in turn constrains the shared variables. When diagnosers L and M disagree on the value of a shared variable r, each performs a local diagnosis to conservatively approximate the maximum probability assignment to the assumptions that would admit a different value for r. This information can then be used to limit propagation of variable changes throughout the system. We have implemented a preliminary version of this system using copies of L2 as the local diagnosers for the purposes of exploration, but we are currently limited to very simple network topologies. Formalizing a reasonably general algorithm for generating a conservative estimate of the most likely diagnoses in a robust, distributed, anytime manner remains future work.

As framed here, the distributed diagnoser never computes complete global diagnoses. Rather, at each local diagnoser it computes feasible local diagnoses. These are projections of the global diagnoses that are relevant to that diagnoser. In the case that control of the plant is distributed, we believe this is appropriate. Each processing node uses the possible states of its components, as determined by the feasible local diagnoses, to inform its control. However, even when performing distributed diagnosis of a distributed system, computation of the global diagnoses may be of interest for purposes such as centralized, supervisory control or display to a user. We note that simply taking the cross-product of the feasible diagnoses produced by each local diagnoser will result in a superset of the global diagnoses. Some combinations of the cross-product may not appear in any consistent global diagnosis. If the consistent global diagnoses are needed, we may compute them by checking combinations of local feasible diagnoses from multiple diagnosers against a combined model using a linear-time technique such as unit propagation. This can be done hierarchically and in parallel, allowing us to rule out inconsistent partial combinations of local diagnoses in order to avoid explicitly checking all combinations. Intuitively and from initial experiments, we suspect for many problems this technique would be a competitive method for producing all consistent global diagnoses. In fact, the performance numbers for the FSA-based distributed algorithm shown in Table 1 are for both computing the conservative and feasible local diagnoses for each local diagnoser and then computing the globally consistent combinations of these local diagnoses. Formalizing this technique and more thoroughly investigating its effectiveness remain future work.

Conclusion

We have developed a distributed diagnosis framework that leverages the topology of the physical plant to limit inter-diagnoser communication and compute consistent diagnoses in an anytime and any information manner, making it robust to communication and processor failures. The framework is conservative, in that it avoids false negatives in favor of false positives in the case where computation cannot be completed due to limited time or communication failure. This property can be vital in applications where safety is critical. In addition to being anytime and conservative, our approach allows a very small granularity for the local diagnosers. We can potentially create a diagnoser per physical component if desired. This flexibility allows us to consider time/space/communication tradeoffs that implement each local diagnoser as an exponentially large (in the small local model size) structure that enables diagnosis to be performed collaboratively on very weak networked processors. One implementation of the distributed algorithm for finding consistent diagnoses has been implemented using a discrete-event formulation and tested on one model. Our future work includes implementations of the algorithm using binary decision diagrams and the unit propagation implementation of L2 to compute locally consistent assignments. The latter will allow direct comparison of centralized and distributed implementations of the same diagnostic technique on a variety of problems modeled for L2. We are also continuing to extend the formulation to include optimization-based distributed diagnosis.

Acknowledgment This work is supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract F33615-99-C3611. Rong Su implemented the distributed diagnoser as a PARC intern. NASA Ames Research Center provided the L2 diagnosis engine.

References

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

Bernard, D. E.; Dorais, G. A.; Fry, C.; Jr., E. B. G.; Kanefsky, B.; Kurien, J.; Millar, W.; Muscettola, N.; Nayak, P. P.; Pell, B.; Rajan, K.; Rouquette, N.; Smith, B.; and Williams, B. C. 1998. Design of the remote agent experiment for spacecraft autonomy. In *Procs. IEEE Aerospace*.

Chu, M.; Haussecker, H.; and Zhao, F. 2001. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *Int'l J. High Performance Computing Applications*. To appear. Also, Xerox Palo Alto Research Center Technical Report P2001-10113, May 2001.

Collin, Z.; Dechter, R.; and Katz, S. 1999. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130. Reprinted in (Hamscher, Console, & de Kleer 1992).

de Kleer, J., and Williams, B. C. 1989. Diagnosis with behavioral modes. In *Proceedings of IJCAI-89*, 1324–1330. Reprinted in (Hamscher, Console, & de Kleer 1992).

Debouk, R.; Lafortune, S.; and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic System: Theory and Applications* 10(1/2):33–86.

Goodrich, C., and Kurien, J. 2001. Continous measurements and quantitative constraints - challenge problems for discrete modeling techniques. In *Proceedings of iSAIRAS-2001*.

Hamscher, W.; Console, L.; and de Kleer, J. 1992. *Readings in Model-Based Diagnosis*. San Mateo, CA: Morgan Kaufmann.

Jackson, W.; Fromherz, M.; Biegelsen, D.; Reich, J.; and Goldberg, D. 2001. Constrained optimization based control of real time large scale systems: Airject movement object system. In *Proceedings of the 40th IEEE Conference on Decision and Control*, 4717–4720.

Kahn, J. M.; Katz, R. H.; and Pister, K. S. J. 1999. Mobile networking for smart dust. In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*.

Kurien, J., and Nayak, P. P. 2000. Back to the future with consistency based trajectory tracking. In *Proceedings of AAAI-00*.

Mosterman, P., and Biswas, G. 1997. Monitoring, prediction and fault isolation in dynamic physical systems. In *Proceedings of AAAI-97*, 100–105.

Nguyen, T., and Deville, Y. 1998. A distributed arc-consistency algorithm. *Science of Computer Programming* 30(1-2):227–250.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–96. Reprinted in (Hamscher, Console, & de Kleer 1992).

Rish, I.; Brodie, M.; and Ma, S. 2002. Efficient fault diagnosis using probing. In *Proceedings of the AAAI Spring Symposium on Information Refinement and Revision for Decision Making: Modeling for Diagnostics, Prognostics and Prediction*.

Roy, P. V. 1999. The separation of concerns in distributed programming: Application to distribution structure and fault tolerance in mozart.

Sanchis, L. A. 1989. Multiple-way network partitioning. *IEEE Transactions on Computers* 38(1):62–81.

Su, R.; Wonham, W. M.; Kurien, J.; and Koutsoukos, X. 2002. Distributed diagnosis for qualitative systems. Technical Report SPL-01-071, Palo Alto Research Center. Submitted to WODES 2002.

Sun Microsystems Inc. 1999. Jini architectural overview.

Waltz, D. L. 1975. Understanding line drawings of scenes with shadows. In Winston, P. H., ed., *The Psychology of Computer Vision*. McGraw-Hill. 19–91.

Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Procs. AAAI-96*, 971–978.

Yokoo, M., and Hirayama, K. 2000. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* 3(2):185–207.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, 614–621.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering* 10(5):673–685.

Zhang, Y., and Mackworth, A. K. 1992. Parallel and distributed finite constraint satisfaction: Complexity, algorithms and experiments. Technical Report TR-92-30, Department of Computer Science, The University of British Columbia.

Zhao, F.; Koutsoukos, X.; Haussecker, H.; Reich, J.; Cheung, P.; and Picardi, C. 2001. Distributed monitoring of hybrid systems: A model-directed approach. In *Proc. IJCAI'2001*, 557–564.

Author Index