To appear in Proceedings of the 16^{th} International Workshop on Principles of Diagnosis, *2005*

Diagnosing Complex Systems with Software-Extended Behavior using Constraint Optimization

Tsoline Mikaelian, Brian C. Williams and Martin Sachenbacher
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar St. Room 32-275, Cambridge, MA 02139
{tsoline, williams, sachenba}@mit.edu

Abstract

Model-based diagnosis has traditionally operated on hardware systems. However, in most complex systems today, hardware is augmented with software functions that influence the system's behavior. In this paper hardware models are extended to include the behavior of associated embedded software, resulting in more comprehensive diagnoses. Capturing the behavior of software is much more complex than that of hardware due to the potentially enormous state space of a program. This complexity is addressed by using probabilistic, hierarchical, constraint-based automata (PHCA) that allow the uniform and compact encoding of both hardware and software behavior. We introduce a novel approach that frames PHCA-based diagnosis as a soft constraint optimization problem over a finite time horizon. The problem is solved using efficient, decomposition-based optimization techniques. The solutions correspond to the most likely evolutions of the software-extended system.

1 Introduction

Model-based diagnosis of devices has traditionally operated on hardware systems [de Kleer and Williams, 1987; Dressler and Struss, 1996]. For instance, given an observation sequence, the Livingstone [Williams and Nayak, 1996] diagnostic engine estimates the state of hardware components based on hidden Markov models that describe each component's behavior in terms of nominal and faulty modes. Researchers at the other end of the spectrum have applied model-based diagnosis to software debugging [Console *et al.*, 1993; Mayer and Stumptner, 2004]. This paper explores the middle ground between the two, in particular the monitoring and diagnosis of systems with combined hardware and software behavior.

Many complex systems today, such as spacecraft, robotic networks, automobiles and medical devices consist of hardware components whose functionality is extended or controlled by embedded software. Examples of devices with software-extended behavior include a communications module with an associated device driver, and an inertial navigation unit with embedded software for trajectory determination. The embedded software in each of these systems interacts with the hardware components and influences their behavior. In order to correctly estimate the state of these devices, it is essential to consider their software-extended behavior.

As an example of a complex system, consider vision-based navigation for an autonomous rover exploring the surface of a planet. The camera used within the navigation system is an instance of a device that has software-extended behavior: the image processing software embedded within the camera module augments the functionality of the camera by processing each image and determining whether it's corrupt. A sensor measuring the camera voltage may be used for estimating the physical state of the camera. A hardware model of the camera describes its physical behavior in terms of inputs, outputs and available sensor measurements. A diagnosis engine such as Livingstone that uses only hardware models will not be able to reason about a corrupt image. The embedded software provides additional information on the quality of the image that is essential for correctly diagnosing the navigation system. To see why this is the case, consider a scenario in which the camera sensor measures a zero voltage. Based solely on hardware models of the camera, the measurement sensor and the battery, the most likely diagnoses will include camera failure, low battery voltage and sensor fault. However, given a software-extended model of the camera that models the process of obtaining a corrupt image, the diagnostic engine may use the information on the quality of the image. Knowing that the processed image is not corrupt, the most likely diagnosis that the measurement sensor is broken may be deduced.

The above scenario demonstrates that a diagnostic system for complex systems with software-extended behavior must: 1) monitor the behavior of both the hardware and its embedded software so that the software state can be used for diagnosing the hardware, and 2) reason about the system state given delayed symptoms. An instance of a delayed symptom is the quality of the image determined by the camera software after it has completed all stages of image processing.

In this paper we introduce a novel model-based monitoring and diagnostic system that operates on software-extended behavior models, to meet requirements 1) and 2) listed above. In contrast to previous work on model-based software debugging [Mayer and Stumptner, 2004; Grosclaude, 2003], the

purpose of this work is to leverage information within the embedded software to refine the diagnoses of physical systems. As such, we are not addressing the problem of diagnosing software bugs. Without loss of generality, we assume that software bugs discovered at runtime are handled by a separate exception handling mechanism.

First, we address modeling issues. Capturing the behavior of software is much more complex than that of hardware due to the hierarchical structure of a program and the potentially large number of its execution paths. We address this complexity by using probabilistic, hierarchical, constraint-based automata (PHCA) [Williams *et al.*, 2001] that can uniformly and compactly encode both hardware and software behavior. Building upon our previous work, we introduce a novel capability for diagnosing systems with software-extended behavior in the presence of delayed symptoms. While Livingstone-2 (L2) [Kurien and Nayak, 2000] handles delayed symptoms for diagnosing hardware systems, our approach generalizes this capability to software-extended behavior by posing the PHCA-based diagnosis problem over a finite time horizon. We frame diagnosis as constraint optimization problem based on soft constraints that encode the structure and semantics of PHCA. The problem is solved using efficient, decomposition-based optimization techniques, resulting in the most likely diagnoses of the software-extended system.

2 Modeling Software-Extended Behavior

Figure 1 shows the software-extended camera module for the vision-based navigation scenario described above. In this example, the failure probabilities for each of the battery, camera and sensor are 10%, 5% and 1% respectively. A typical behavioral model of the camera is shown on the left of Figure 2. The camera can be in one of 3 modes: on, off or broken. The hardware behavior in each of the modes is specified in terms of inputs to the camera such as the power and the behavior of camera components such as the shutter. The broken mode is unconstrained in order to accommodate novel types of failures. Mode transitions can occur probabilistically, or as a result of issued commands. The battery and the sensor components can be modeled in a similar way. For the scenario introduced above, the most likely diagnoses of the module can be generated based on the hardware models alone, as shown on the right of Figure 2. However, the image processing software provides extended functionality that is not described by the model in Figure 2. The specification of the embedded software can offer important evidence that substantially alters the diagnosis. A sample specification of the behavior of the image processing software may take the following form:

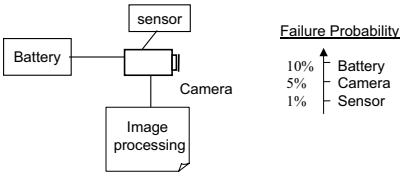


Figure 1: Camera Module for Navigation System

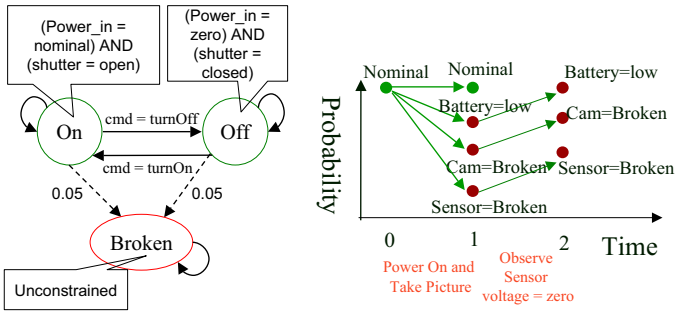


Figure 2: *left:* Behavior Model for the Camera Component. *right:* Most likely diagnoses of the camera module based on hardware component models. Nominal state = no failures.

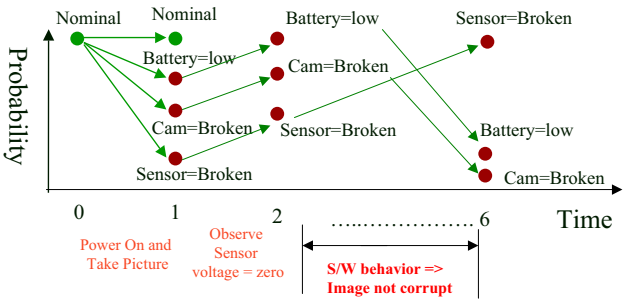


Figure 3: Most likely diagnoses of the camera module based on the software-extended behavior models.

> *If an image is taken by the camera, process it to determine whether it's corrupt. If algorithm X determines that the image is corrupt, discard it and reset the camera; retry until a non-corrupt image is obtained for navigation. Once a high quality image is stored, wait for new image request from navigation unit.*

Such a specification abstracts the behavior of the image processing software implemented in an embedded programming language such as Esterel [Berry and Gonthier, 1992] or RMPL [Williams *et al.*, 2001]. For the above scenario, the behavior of the embedded software provides diagnostic information necessary to correctly estimate the state of the camera module. Given that the image is not corrupt, the possibility that the camera is broken becomes very unlikely. This is illustrated in Figure 3.

Unlike a hardware component that can typically be described by a single mode of behavior, monitoring software behavior necessitates tracking simultaneous hierarchical modes. A modeling formalism that will allow the specification of software behavior must support: 1) full concurrency for modeling sequential and parallel threads of behavior, 2) conditional behavior, 3) iteration, 4) preemption, 5) probabilistic behavior for modeling uncertainty and 6) propositional logic constraints for specifying co-temporal relationships among variables. The following section reviews the modeling framework for handling these requirements.

3 Probabilistic, Hierarchical Constraint-based Automata (PHCA)

Probabilistic, hierarchical, constraint-based automata (PHCA) were introduced in [Williams *et al.*, 2001] as a compact encoding of Hidden Markov Models (HMMs). These automata have the required expressivity to uniformly model both hardware and software behavior.

Definition 1 (PHCA)
A PHCA is a tuple $< \Sigma, P_\Theta, \Pi, O, C, P_T >$, where:

- Σ is a set of locations, partitioned into primitive locations Σ_p and composite locations Σ_c. Each composite location denotes a hierarchical, constraint automaton. A location may be marked or unmarked. A marked location represents an active branch.
- $P_\Theta(\Theta_i)$ denotes the probability that $\Theta_i \subseteq \Sigma$ is the set of start locations (initial state). Each composite location $l_i \subseteq \Sigma_c$ may have a set of start locations that are marked when l_i is marked.
- Π is a set of variables with finite domains. $C[\Pi]$ is the set of all finite domain constraints over Π.
- $O \subseteq \Pi$ is the set of observable variables.
- $C : \Sigma \rightarrow C[\Pi]$ associates with each location $l_i \subseteq \Sigma$ a finite domain constraint $C(l_i)$.
- $P_T(l_i)$, for each $l_i \subseteq \Sigma_p$, is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p^{(t)} \times C[\Pi]^{(t)} \rightarrow 2^{\Sigma^{(t+1)}}$. Each transition function maps a marked location into a set of locations to be marked at the next time step, provided that the transition's guard constraint is entailed.

Definition 2 (PHCA State)
The state of a PHCA at time t is a set of marked locations called a marking $m^{(t)} \subset \Sigma$.

Figure 4 shows a PHCA model of the camera module in Figure 1. The "On" composite location contains three subautomata that correspond to primitive locations "Initializing", "Idle" and "Taking Picture". Each composite or primitive location of the PHCA may have behavioral constraints. The behavioral constraint of a composite location, such as $(power\_in = nominal)$ for the "On" location, is inherited by each of the subautomata within that composite hierarchy. In addition to the physical camera behavior, the model incorporates qualitative software behavior such as processing the quality of an image. Furthermore, based on the image quality, the possible camera configurations may be constrained by the embedded software. For example, if the image is determined to be corrupt, the software attempts to reset the camera. This restricts the camera behavior to transition to the Initializing location.

Recall that Figure 3 shows the most likely state trajectories based on the software-extended PHCA model. At time step 2, as the sensor measurement indicates zero voltage, the most likely diagnosis trajectories are 1) battery = low with 10% probability, 2) camera = broken with 5% probability and 3)

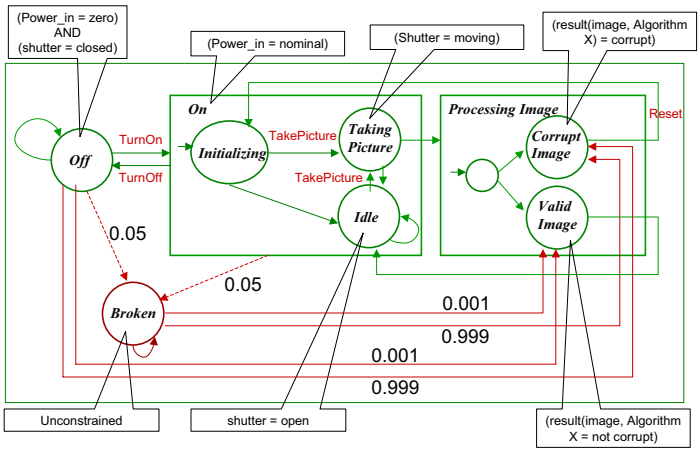


Figure 4: PHCA model for the camera/image processing module. Circles represent primitive locations, boxes represent composite locations and small arrows represent start locations.

sensor is broken with 1% probability. For the first trajectory that indicates that the battery is low, the power to the camera is not nominal, hence the camera will stay in the "Off" location. For the second trajectory, the camera will be in the "Broken" location. For the third trajectory that indicates that the sensor is broken, the power input to the camera will be unconstrained, and hence the PHCA state of the camera may include a marking of the "On" location. Although the evolutions of this third trajectory have an initially low probability of 1%, at time step 6 they become more likely than the others as the embedded software determines that the image is valid. The reason is because the second most likely trajectory at time 2 with camera = "Broken" location marked has a 0.001 probability of generating a valid image, thus making the probability of that trajectory 0.005% at time 6. This latter trajectory is less probable than those trajectories stemming from the sensor being broken with 1% probability. Similarly, the first trajectory with battery = low and camera = Off becomes less likely at time step 6 as there is 0.001% probability of processing a valid image while the camera is "Off".

PHCA models have the following advantages that support their use for diagnosing systems with software-extended behavior. First, since HMMs may be intractable, PHCA encoding is essential to support real-time, model-based deduction. Second, PHCAs provide the expressivity to model the behavior of embedded software by satisfying requirements 1)-6) above [Williams *et al.*, 2001]. Third, the hierarchical nature of the automata enables modeling of complex concurrent and sequential behaviors. As an example of concurrency, the PHCA in Figure 4 allows the simultaneous marking of the "On" location of the camera, as well as the "Initializing", "Idle", or "Taking Picture" locations. This is in contrast to diagnosis based on non-hierarchical models that can estimate each component to be in a single mode of operation. State estimates of components may be required at different levels of granularity. For example, an image-based navigation function may require high level camera state estimates such as "On" or "Off". On the other hand, a function that coordinates imaging

activities may need more detailed camera state estimates such as "Initializing" or "Taking Picture". Simultaneous marking of several camera locations such as "On" and "Initializing", allows their use within functions that require estimates at different levels of granularity.

The following sections introduce a novel diagnostic system based on the PHCA modeling framework. We first introduce our approach for diagnosis over a single time step, and then extend it to handle delayed symptoms. Our approach results in a capability for diagnosing systems with software-extended behavior in the presence of delayed symptoms. Furthermore, our formulation of the diagnosis problem enables the use of decomposition techniques [Dechter, 2003] for efficient solution extraction.

4 Diagnosis as Constraint Optimization based on PHCA Models

We frame diagnosis based on PHCA models as a soft constraint optimization problem (COP) [Schiex *et al.*, 1995]. The COP encodes the PHCA models as probabilistic constraints, such that the optimal solutions correspond to the most likely PHCA state trajectories. The soft constraint formulation allows a separation between probability specification and variables to be solved for. Thus, we can associate probabilities with constraints that encode transitions, while solving for state variables.

Definition 3 (Constraint Optimization Problem)
A constraint optimization problem (COP) is a triple (X, D, F) where $X = \{X_1, ..., X_n\}$ is a set of variables with corresponding set of finite domains $D = \{D_1, ..., D_n\}$, and $F = \{F_1, ..., F_n\}$ is a set of preference functions $F_i : (S_i, R_i) \rightarrow C_i$ where (S_i, R_i) is a constraint and C_i is a set of preference (or cost) values. Each constraint (S_i, R_i) consists of a scope $S_i = \{X_{i1}, ..., X_{ik}\}$ representing a subset of variables X, and a relation $R_i \subseteq D_{i1} \times ... \times D_{ik}$ on S_i that defines all tuples of values for variables in S_i that are compatible with each other. Each preference function F_i maps the tuples of (S_i, R_i) to values in C_i. The solution to variables of interest (solution variables) $Y \subseteq X$ is an assignment to Y that is consistent with all constraints, has a consistent extension to all variables X, and minimizes (or maximizes) a global objective function defined in terms of preference functions F_i.

Given a PHCA state at time t and an assignment to observable and command variables in Π (see Definition 1) at times t and $t+1$, in order to estimate PHCA state at time $t+1$, we encode both the structure and execution semantics of the PHCA as a COP, consisting of:

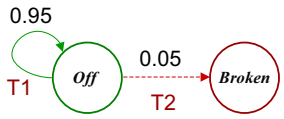
- Set of variables $X_\Sigma \cup \Pi \cup X_{Exec}$, where $X_\Sigma = \{L_1, ..., L_n\}$ is a set of variables that correspond to PHCA locations $l_i \in \Sigma$, Π is the set of PHCA variables, and $X_{Exec} = \{E_1, ..., E_n\}$ is a set of auxiliary variables used for encode the execution semantics of the PHCA.
- Set of finite, discrete-valued domains $D_{X_\Sigma} \cup D_\Pi \cup D_{X_{Exec}}$, where $D_{X_\Sigma} = \{Marked, Unmarked\}$ is the domain for each variable in X_Σ, D_Π is the set of domains for PHCA variables Π, and D_{Exec} is a set of domains for variables X_{Exec}.
- Set of constraints R that include the behavioral constraints associated with locations within the PHCA, as well as encoding of the PHCA execution semantics.
- Preference functions F in the form of probabilities associated with tuples of constraints R. Tuples of hard constraints that are disallowed by the constraint are assigned probability 0.0, while the tuples allowed by the constraint are assigned probability 1.0. Tuples of soft constraints are mapped to a range of probability values based on the PHCA model. These probability values reflect the probability distribution P_Θ of PHCA start states and probabilities associated with PHCA transitions P_T.
- The optimal solution to the COP is an assignment to solution variables X_Σ that represent the state of the PHCA, while maximizing the probability of the transitions that lead to that state from the previous time step. This corresponds to a state assignment that maximizes the product of the probabilities of the enabled constraint tuples.

A key to framing PHCA-based diagnosis as COP is the formulation of the constraints R that capture the execution semantics of the PHCA. PHCA execution involves determining the entailment of behavioral constraints, identifying enabled transitions from a current PHCA state, and taking those transitions to determine the next state. Referring back to the PHCA example in Figure 4, if we assume that at time t the PHCA state is $< On < Idle >>$ and that the transition guard constraint $(command = TakePicture)$ is entailed, and at time t+1 the behavioral constraint $(shutter = moving)$ of the transition's target location is entailed, then the PHCA state at time t+1 will be $< On < TakingPicture >>$. To encode entailment of conditions such as $(command = TakePicture)$, a variable E_T is introduced with domain $\{Entailed, Not - Entailed\}$ to denote whether the transition guard condition is entailed. Entailment of a condition is then formulated as a COP constraint that allows the assignment $E_T = Entailed$ to be associated with tuples that list all possible assignments to the variable $command$ that entail the condition $(command = TakePicture)$. Entailment constraints are generated for all locations that have behavioral constraints and for all transitions that have guard constraints.

The following example on the left of Figure 5 shows a probabilistic choice between two transitions for a section of the PHCA in Figure 4. In order to encode this probabilistic choice, we first introduce a location variable $X^{(t)}_{Off}$ for time t, with domain $\{Marked, Unmarked\}$. Then auxiliary variables $E^{(t)}_{T1}$ and $E^{(t)}_{T2}$ with domain $\{Enabled, Disabled\}$ are introduced for transitions T1 and T2 respectively.

The COP constraint that encodes the probabilistic choice among the two transitions T1 and T2 is formulated logically:

$$X^{(t)}_{Off} = Marked \equiv (\exists\, T \in \{T1, T2\} \mid : E^{(t)}_T = Enabled \wedge (\forall\, T' \in \{\{T1, T2\} - T\} \mid : E^{(t)}_{T'} = Disabled)) \bigwedge X^{(t)}_{Off} = Unmarked \equiv (\forall\, T \in \{T1, T2\} \mid : E^{(t)}_T = Disabled)$$



$X_{Off}^{(t)}$	$E_{T1}^{(t)}$	$E_{T2}^{(t)}$	Prob.
Marked	Enabled	Disabled	0.95
Marked	Disabled	Enabled	0.05
Unmarked	Disabled	Disabled	1.0

Figure 5: *left:* PHCA with two probabilistic transitions. *right:* Probabilistic transition constraint.

This logical formula is compiled into a set of tuples with associated probability values, as shown in Figure 5 (*right*). The tuples are mapped to probability values by the following preference function:

$$F_T = \begin{cases} Prob(T_i) & if\ (\exists T_i^{(t)} : E_{T_i}^{(t)} = Enabled) \\ 1.0 & otherwise \end{cases}$$

The above constraint identifies the enabled transition, but does not encode taking the transition. In general, the following constraint encodes taking enabled transitions, unless the behavior constraint of the transition's target location is not entailed:

$(\forall\ L \in \Sigma \mid : ((\exists\ \tau \in \{T | Target(T) = L\} \mid : E_\tau^{(t-1)} = Enabled) \wedge Behavior_L^{(t)} = Entailed) \Rightarrow X_L^{(t)} = Marked)$

where E_τ represents a transition variable, $Behavior_L$ is an entailment variable for the behavior constraints of location L $\cup$ its composite parent if L is within a hierarchy, and X_L is the location variable of L. The constraint is instantiated for each location of the PHCA, as indicated by $\forall\ L \in \Sigma$.

Some semantic rules apply to PHCA hierarchies. For example, when a composite location becomes marked, all of its start locations become marked. Since "Initializing" is a start location of the composite "On" location, a PHCA in state $< Off >$ may transition to state $< On < Initializing >>$. Furthermore, a composite location should be marked if any of its subautomata are marked. The COP constraints must correctly capture such PHCA semantics and encode mutual exclusions to avoid interference and conflicting effects among the constraints. For brevity, the complete encoding of constraints is not presented.

The formulation of diagnosis as COP is performed offline. Given a PHCA, we have implemented a compiler that automatically generates the corresponding COP. The COP is then used in an online solution phase by dynamically updating it to incorporate constraints on new observations and issued commands. The solutions to the COP can be generated up to a given probability threshold using a constraint optimization solver for soft constraints [Sachenbacher and Williams, 2004]. The solutions incorporate the probability distribution on the initial states as encoded by the COP. The most likely solutions generated at a time step t dynamically update the COP to constrain the set of start states for solving the COP at time step t+1. For example, as Figure 3 shows, state estimates at time 2 may only be reached through those at time 1. Thus limiting the number of state trajectories maintained at each time step has implications for diagnosing faults that manifest delayed symptoms.

5 Diagnosis with Delayed Symptoms

Ideally, diagnosis will maintain a complete probability distribution of all possible system states. However, maintaining all possible state trajectories at each time step is intractable because of exponential growth in state space. Thus at every time step a limited number of trajectories are typically maintained. A potential problem with this approach is that it may miss the best diagnosis if a trajectory through a pruned state that is initially very unlikely becomes very likely after additional evidence. Figure 6 illustrates this situation for the camera module, where the initially unlikely state $(Sensor = Broken)$ is pruned, resulting in the best diagnosis to be unreachable when additional evidence is available at time 6.

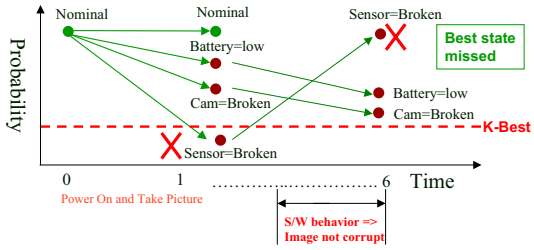


Figure 6: Missed diagnosis as a result of tracking a limited number of trajectories (K-Best)

Dealing with delayed symptoms is particularly important for diagnosing systems with software-extended behavior, due to typically delayed observations associated with software processing. Livingstone-2 (L2) [Kurien and Nayak, 2000] addresses the problem of delayed symptoms for diagnosing hardware systems. We generalize the L2 capability to PHCA-based diagnosis.

We extend our COP formulation of PHCA-based diagnosis to provide flexibility for regenerating the most likely diagnoses over a finite time horizon rather than a single previous step. Thus, we frame the COP over a finite time horizon (N-stages) and leverage the N-stage history of observations and issued commands to generate the most likely diagnosis trajectories over the horizon. This involves augmenting the COP in the previous section to include model variables and constraints for each time step within the N-stage horizon. The solutions to the COP become assignments to location variables $X_\Sigma^{(t)}$, t $\in \{0..N\}$, representing PHCA state trajectories that have maximum probability within the horizon. This probability corresponds to the product of transition probabilities enabled within that trajectory, multiplied by the probability of the initial state of the trajectory. As time progresses during the online solution phase, the N-stage horizon is shifted from $(t \rightarrow t+N)$ to $(t+1 \rightarrow t+N+1)$ and the COP over the new horizon is dynamically updated by constraining its start states at time t+1 to match the solutions from the previous iteration. This reformulation still limits the number of trajectories tracked to a given probability threshold, as described in the previous section. Referring to Figure 6, if we consider a time horizon $(0 \rightarrow 6)$, state trajectories will be regenerated starting from the $(Nominal)$ state at time 0. Therefore, even though the number of trajectories is limited,

the trajectory ending at state ($Sensor = Broken$) at time 6 will have the highest probability based on the delayed observation. Consequently, the state ($Sensor = Broken$) at time 2 will be maintained because it is part of the most likely trajectory at time 6.

Decreasing the probability threshold for the trajectories being tracked solves the delayed-symptom problem by maintaining a larger number of states at each time step. However, for a system with many combinations of similar failure states with high probability, the number of trajectories maintained will have to be very large in order to be able to account for a delayed symptom that supports an initially low probability state. For such systems, considering even a small number of previous time steps gives enough flexibility to regenerate the correct diagnosis.

6 Implementation and Discussion

The PHCA model-based diagnosis capability, described above, has been implemented in C++. Figure 7 shows the offline compilation phase and the online solution phase of the diagnosis process.

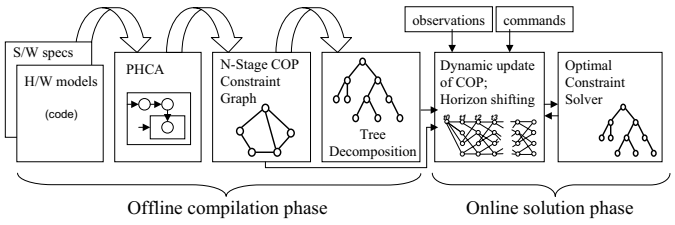


Figure 7: Process diagram for PHCA-based diagnosis

In the offline phase, the N-Stage COP is generated automatically, given a PHCA model and parameter N. To enhance the efficiency of the online solution phase, tree decomposition [Gottlob *et al.*, 2000] is applied to decompose the COP into independent subproblems. This enables backtrack-free solution extraction during the online phase [Dechter, 2003]. In our implementation, the COP is decomposed using a tree decomposition package that implements bucket elimination [Kask *et al.*, 2003].

The online monitoring and diagnosis process uses both the COP and its corresponding tree decomposition. The online phase consists of a loop that shifts the time horizon, updates and solves the COP at each iteration. The COP is updated by incorporating new observations and commands, and constraining the start states to track the trajectories obtained within the previous horizon. At each iteration of the loop, the updated COP is solved using an implementation of the decomposition-based constraint optimization algorithm in [Sachenbacher and Williams, 2004] that can generate diagnoses up to a given probability threshold.

For the camera model with $N = 2$, the COP has ~ 150 variables and ~ 100 constraints and is solved online in ~ 1 sec, resulting in more comprehensive diagnoses than previous hardware models. Future work includes evaluating the efficiency of the COP formulation using several complex scenarios, optimizing the COP formulation by minimizing the number of variables and constraints generated, investigating the optimal size of the diagnosis horizon and its relationship to the number of trajectories tracked.

7 Acknowledgments

This research is sponsored in part by NASA CETDP under contract NNA04CK91A and by the DARPA SRS program under contract FA8750-04-2-0243.

References

[Berry and Gonthier, 1992] G. Berry and G. Gonthier. The *esterel* programming language: design, semantics and implementation. *Science of Computer Programming*, 19(2):87–152, Nov. 1992.

[Console *et al.*, 1993] L. Console, G. Friedrich, and D. Theseider Dupre. Model-based diagnosis meets error diagnosis in logic programs. In *Proc. IJCAI-93*, 1993.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Dressler and Struss, 1996] O. Dressler and P. Struss. The consistency-based approach to automated diagnosis of devices. *Principles of Knowledge Representation*, pages 267–311, 1996.

[Gottlob *et al.*, 2000] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[Grosclaude, 2003] I. Grosclaude. Model-based monitoring of component-based software systems. In *Proc. DX-03*, 2003.

[Kask *et al.*, 2003] K. Kask, R. Dechter, and J. Larrosa. Unifying cluster-tree decompositions for automated reasoning. Technical report, U. of California at Irvine, 2003.

[Kurien and Nayak, 2000] J. Kurien and P. Nayak. Back to the future for consistency-based trajectory tracking. In *Proc. AAAI-00*, 2000.

[Mayer and Stumptner, 2004] W. Mayer and M. Stumptner. Approximate modeling for debugging of program loops. In *Proc. DX-04*, 2004.

[Sachenbacher and Williams, 2004] M. Sachenbacher and B. C. Williams. Diagnosis as semiring-based constraint optimization. In *Proc. ECAI-04*, 2004.

[Schiex *et al.*, 1995] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems:hard and easy problems. In *Proc. IJCAI-95*, 1995.

[Williams and Nayak, 1996] B. C. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proc. AAAI-96*, pages 971–978, 1996.

[Williams *et al.*, 2001] B. C. Williams, S. Chung, and V. Gupta. Mode estimation of model-based programs: monitoring systems with complex behavior. In *Proc. IJCAI-01*, 2001.