An Efficient Model-based Diagnosis Engine for Hybrid Systems using Structural Model Decomposition

Anibal Bregon[1], Sriram Narasimhan[2], Indranil Roychoudhury[3], Matthew Daigle[4], and Belarmino Pulido[5]

[1,5] *Dept. of Computer Science, University of Valladolid, Spain*
{*anibal,belar*}*@infor.uva.es*

[2] *Univ. of California Santa Cruz, NASA Ames Research Center. Moffett Field, CA 94035, USA*
sriram.narasimhan-1@nasa.gov

[3] *SGT Inc., NASA Ames Research Center. Moffett Field, CA 94035, USA*
indranil.roychoudhury@nasa.gov

[4] *NASA Ames Research Center. Moffett Field, CA 94035, USA*
matthew.j.daigle@nasa.gov

Abstract

Complex hybrid systems are present in a large range of engineering applications, like mechanical systems, electrical circuits, and embedded computation systems. The behavior of these systems is made up of continuous and discrete event dynamics that increase the difficulties for accurate and timely online fault diagnosis. The Hybrid Diagnosis Engine (HyDE) architecture offers flexibility to the diagnosis application designer to choose the modeling paradigm and the reasoning algorithms. The HyDE architecture supports the use of multiple modeling paradigms at the component and system level. However, HyDE faces some problems regarding performance in terms of time and space complexity. This paper focuses on developing efficient model-based methodologies for online fault diagnosis in complex hybrid systems. To do this, we propose a diagnosis framework where structural model decomposition is integrated within the HyDE diagnosis framework to reduce the computational complexity associated with the fault diagnosis of hybrid systems. As a case study, we apply our approach to a diagnostic benchmark problem, the Advanced Diagnostics and Prognostics Testbed (ADAPT), using real data.

1. Introduction

Nowadays, complex hybrid systems are present in many engineering applications, from electrical circuits to embedded computation systems. Their behavior is made up of continuous and discrete event dynamics, making accurate and timely online fault diagnosis more difficult. This paper focuses on developing efficient model-based methodologies for online fault diagnosis in complex hybrid systems. Hybrid systems modeling and diagnosis have been approached by the DX community, and several proposals have been made based on hybrid modeling (Mosterman & Biswas, 1999), hybrid state estimation (Hofbaur & Williams, 2004), or a combination of on-line state tracking and residual evaluation (Benazera & Travé-Massuyès, 2009; Bayoudh et al., 2008). In all cases, the solution requires to somehow model and eventually fully or approximately estimate the set of possible states, and to diagnose the current set of consistent modes. A major restriction, however, is that each technique uses its own modeling paradigm and the reasoning algorithms implement a single strategy. This does not facilitate the generation of flexible, integrated, reasoning solutions by the inclusion of additional diagnosis strategies, thus restricting the diagnostic capabilities of the hybrid diagnoser.

In (Narasimhan & Brownston, 2007), the authors proposed a general framework for stochastic and hybrid model-based diagnosis called Hybrid Diagnosis Engine (HyDE). HyDE offers flexibility to the diagnosis application designer to choose the modeling paradigm and the

reasoning algorithms. The HyDE architecture supports the use of multiple modeling paradigms at the component and system level. Several alternative algorithms are available for the various steps in diagnostic reasoning. This approach is extensible, with support for the addition of new modeling paradigms as well as diagnostic reasoning algorithms for existing or new modeling paradigms. However, HyDE faces some problems regarding performance in terms of space and time complexity.

Recently, we have proposed to use structural model decomposition for efficient fault diagnosis and prognosis in continuous systems (Bregon, Biswas, & Pulido, 2012; Daigle et al., 2011a, 2012). In (Roychoudhury et al., 2013), we generalized those ideas and proposed a common model decomposition framework, where we solve the model decomposition problems for three separate system health management tasks, namely, estimation (used for residual generation that is usually required for fault detection and fault identification), fault isolation, and prediction (used for fault prognostics). The basic idea of the approach is to partition the global system model into submodels based on the set of measurements. This way, we will have submodels that are smaller than the global system model, leading to efficiency improvements and potential for concurrent computation.

In this paper, we integrate structural model decomposition as in (Roychoudhury et al., 2013) within the HyDE diagnosis framework. Structural model decomposition is used to decompose the HyDE models, thus reducing the computational complexity associated with the fault diagnosis of hybrid systems. This work contributes in two different aspects. First, we propose an online diagnosis approach for hybrid systems where the system model is partitioned into submodels, which are implemented using the HyDE modeling framework. Then, the global diagnosis result is provided by the combination of the local diagnosis results corresponding to the submodels. Second, we apply our approach to a real system, the Advanced Diagnostics and Prognostics Testbed (ADAPT) with satisfactory results.

The rest of the paper is organized as follows. Section 2 presents the HyDE diagnosis framework. Section 3 discusses the basic ideas of structural model decomposition. Section 4 proposes an integrated framework where structural model decomposition is used to reduce HyDE's computational burden. Section 5 shows results for the case study. Section 6 reviews the related work and current approaches for hybrid systems fault diagnosis and structural model decomposition. Finally, Section 7 concludes the paper.

2. HyDE

HyDE (Hybrid Diagnosis Engine) (Narasimhan & Brownston, 2007) combines ideas from consistency-based, control-theory-based and stochastic diagnosis approaches to provide a general, flexible and extensible architecture for stochastic and hybrid diagnosis. HyDE supports the use of multiple modeling paradigms and is extensible to support new paradigms. HyDE also offers a library of algorithms to be used in the various steps of the diagnostic reasoning process. The key features of HyDE are:

- Diagnosis of multiple discrete faults.
- Support for hybrid models, including autonomous and commanded discrete switching.
- Support for stochastic models and stochastic reasoning.
- Capability for handling time delay in the propagation of fault effects.

Next we present the HyDE modeling approach and reasoning procedure.

2.1. HyDE Models

HyDE models have two parts, the transition model and the behavior model. The transition model describes the components that make up the system, the various operating modes of the system (including faulty ones), and the conditions for transitions between the operating modes. The behavior model specifies the behavior evolution and has three parts: the propagation model, integration model, and dependency model. The information in the propagation model allows the estimation of unknown variable values from known variable values. The dependency model captures information about the dependencies between variables, models, and components. The integration model describes how the variables' values are propagated across time steps. HyDE supports the representation of each of the behavior models in more than one paradigm.

2.2. HyDE Reasoning

HyDE reasoning is the maintenance of a set K of weighted candidates (k_i, w_i). A candidate represents the hypothesized trajectory of the system inferred from the transition and behavior models, knowledge of the initial operating modes of all components and initial values of all variables, and the sensor observations reported to HyDE. The candidates' weights are a way of ranking them and depend on several factors, including prior probabilities of transitions and the degree of fit between model predictions and observations. Although weights

are in the range [0, 1], weight is not a probability measure.

Each candidate contains a possible trajectory of system behavior evolution represented in the form of a hybrid state history and transition history. The hybrid state is a snapshot of the entire system state at any single instant. It associates all components with their current operating modes and all variables with their current values. Applications run HyDE at discrete time steps, typically but not necessarily when observations are available. Time steps need not be periodic. For each time step that HyDE reasons about, a candidate contains two hybrid states, one at the beginning of the time step and one at the end, as well as the set of transitions taken by the system between the previous and current time steps.

At time step 0, the candidate set is initialized with candidate(s) derived from the initial hybrid state of the system. Once the initial candidate set has been created, HyDE's reasoning process uses the same sequence of operations for each time step. The reasoning process can be divided into three categories of operations (Narasimhan & Brownston, 2007):

1. *Candidate Set Management* maintains the candidate set. The operations include updating the weights of all candidates, pruning candidates that do not satisfy minimum weight requirements, adding new candidates (the next best ones from the candidate generator) when necessary, and optionally re-sampling or normalizing the distribution of weights.
2. *Candidate Testing* deals with operations on a single candidate. The operations include determining the occurrence of any transitions, estimating the hybrid states at the beginning and end of a time step, comparing against observations to update weight of the candidate as well as reporting inconsistencies.
3. *Candidate Generation* creates candidate generators from inconsistencies reported by Candidate Testing and supplies the next-best potential (untested) candidate to Candidate Set Management when requested. This is achieved using a conflict directed search. First reported inconsistencies are used to generate conflicts, i.e., the subset of operating modes that cannot all be true at the same time. The conflicts are then used to guide a search for new candidates by optimizing some candidate property (typically weight or size).

As we have mentioned, the size of the system model (HyDE uses the global model of the complete system) directly affects the computational complexity for each one of the steps in the reasoning process. Our proposal on this work is to use structural model decomposition to divide the global system model into minimal submodels such that the complexity in the reasoning process is reduced. The next section describes our structural model decomposition approach to compute minimal submodels. Then, in Section 4 we will show in detail how these minimal submodels are integrated within the HyDE framework.

3. Structural Model Decomposition

In this section, we briefly present our structural model decomposition framework (Roychoudhury et al., 2013). We begin with the definition of a *model*.

Definition 1 (Model)**.** A *model* $\mathcal{M}$ is a tuple $\mathcal{M} = (V, C)$, where V is a set of variables, and C is set of constraints. V consists of five disjoint sets, namely, the set of state variables, X; the set of parameters, Θ; the set of inputs, U; the set of outputs, Y; and the set of auxiliary variables, A. Each constraint $c = (\varepsilon_c, V_c) \in C$ consists of an equation ε_c involving variables $V_c \in V$.

Input variables $u \in U$ are known/measured; and the output variables $y \in Y$ correspond to (measured) sensor signals. Parameters $\theta \in \Theta$ include explicit model parameters that are used in the model constraints. Θ does not need to include all parameters in the equations, only those that must be included explicitly (e.g., for joint state-parameter estimation or fault isolation). These parameters, by definition, are not computed in terms of any other variables, and, in this way, appear as inputs. Since the state variables X are, by definition, enough to describe the future behavior of the system, the auxiliary variables $a \in A$ are not strictly needed, however, they make the model easier to parse, develop, and implement.

As shown in Defn. 1, a constraint $c = (\varepsilon_c, V_c)$ includes an equation ε_c over the set of variables V_c. Note that c does not impose any computational causality on the variables V_c, i.e., although ε_c captures the information about how to compute a variable $v \in V_c$ in terms of all other variables in V_c, the constraint does not specify which $v \in V_c$ is the dependent variable in equation ε_c. We write a constraint $c_1 = (\varepsilon_{c_1}, V_{c_1})$ by its equation, e.g., as follows:

$$a + b = c + d \qquad (c_1)$$

where $V_{c_1} = \{a, b, c, d\}$.

In order to define for a constraint c which variable $v \in V_c$ is the dependent variable that is computed by the others using the constraint, we require the notion of a *causal assignment*.

Definition 2 (Causal Assignment)**.** A *causal assignment* α to a constraint $c = (\varepsilon_c, V_c)$ is a tuple $\alpha =$

(c, v_c^{out}), where $v_c^{out} \in V_c$ is assigned as the dependent variable in equation ε_c.

Unlike a constraint, a causal assignment defines a computational causality (or computational direction) to a particular variable $v_c^{out} \in V_c$ in the constraint in which it can be computed in terms of all other variables in V_c. We write a causal assignment of a constraint using the constraint's equation in a causal form. For example, for constraint c_1 above choosing $v_{c_1}^{out} = d$:

$$d := a + b - c \qquad (\alpha_1)$$

where Constraint c_1 is rewritten with a := symbol to explicitly denote that the direction of computation is from variables a, b, and c to d.

We say that a set of causal assignments $\mathcal{A}$, for a model $\mathcal{M}$ is *valid* if

- For all $v \in U \cup \Theta$, $\mathcal{A}$ does not contain any α such that $\alpha = (c, v)$, i.e., U and Θ are not computed in terms of any other variables.
- For all $v \in Y$, $\mathcal{A}$ does not contain any $\alpha = (c, v_c^{out})$ where $v \in V_c - \{v_c^{out}\}$, i.e., no variable is computed in terms of any $y \in Y$.
- For all $v \in V - U - \Theta$, $\mathcal{A}$ contains exactly one $\alpha = (c, v)$, i.e., other than the variables in U and Θ, every variable must have exactly one constraint to compute it.

A *causal model* is a model extended with a valid set of causal assignments.

Definition 3 (Causal Model)**.** Given a model $\mathcal{M}^* = (V, C)$, a *causal model* for $\mathcal{M}^*$ is a tuple $\mathcal{M} = (V, C, \mathcal{A})$, where $\mathcal{A}$ is a set of valid causal assignments.

Given a model, we generate submodels that allow for the computation of a given set of variables using only local inputs. Given a definition of the local inputs (in general, selected from V) and the set of variables we wish to be computed by the submodel (selected from $V - U - \Theta$), we create from a causal model $\mathcal{M}$ a causal submodel $\mathcal{M}_i$. We obtain a submodel in which only a subset of the variables in V are computed using only a subset of the constraints in C. In this way, each submodel computes its variable values independently from all other submodels. A submodel can be defined as follows.

Definition 4 (Causal Submodel)**.** A *causal submodel* $\mathcal{M}_i$ of a causal model $\mathcal{M} = (V, C, \mathcal{A})$ is a tuple $\mathcal{M}_i = (V_i, C_i, \mathcal{A}_i)$, where $V_i \subseteq V$, $C_i \subseteq C$, and $\mathcal{A}_i$ is a set of (valid) causal assignments for $\mathcal{M}_i$.

Note that, in general, $\mathcal{A}_i$ is not a subset of $\mathcal{A}$, because since we allow to select local inputs from Y, these variables become local inputs, i.e., appear in U_i, and the causal assignment in $\mathcal{A}$ that computes these variables is changed to a form where some other variable in the corresponding constraint is selected as the dependent variable. As a result, these causal assignments will be different, but the rest of the causal assignments in $\mathcal{A}_i$ will still be found in $\mathcal{A}$.

The procedure for generating a submodel from a causal model is given as Algorithm 1 (Roychoudhury et al., 2013). Given a causal model $\mathcal{M}$, a set of variables $U^* \supseteq U$ that includes the input variables in $\mathcal{M}$ as well as some other variables previously not in U that are considered as local inputs, and a set of variables to be computed V^*, and a preferences list, P (explained below), the `GenerateSubmodel` algorithm derives a causal submodel $\mathcal{M}_i$ that computes V^* using a subset of U^*.

In the following we briefly describe the algorithm, see (Roychoudhury et al., 2013) for additional details. In Algorithm 1, the queue, *variables*, represents the set of variables that have been added to the submodel but have not yet been resolved, i.e., they cannot yet be computed by the submodel. This queue is initialized to V^*, the set of variables that must be computed by the submodel. The algorithm then loops until this queue has been emptied, i.e., the submodel can compute all variables in V^* using only variables in U^*. Within the loop, the next variable v is popped off the queue. We then determine the best constraint to use to resolve this variable with the `GetBestConstraint` subroutine (Subroutine 2). We add the constraint to the submodel and the causal assignment for the constraint in the form that computes v. We then need to resolve all the variables being used to compute v, i.e., all its predecessors in the causal graph. Each of these variables that have not already been visited (not already in V_i), are not parameters (not in Θ), and are not local inputs (not in U^*) must be resolved and so are added to the queue. Then the variables are added to the submodel and the loop continues until the queue is emptied.

The goal of the `GetBestConstraint` subroutine is to find the best constraint to resolve v. The subroutine constructs a set C that is the set of constraints that can completely resolve the variable, i.e., resolves v without further backward propagation (all other variables involved in the constraint are in $V_i \cup \Theta \cup U^*$), and then chooses the best according to a preferences list P. If no such constraint exists, then the constraint that computes v in the current causal assignment is chosen, and further backward propagation will be necessary. Here, we are preferring minimal resolutions of v, i.e., those that do not require backward propagation, because then the submodel will be minimal in the number of variables and constraints needed to compute V^*.

Algorithm 1 $\mathcal{M}_i = \texttt{GenerateSubmodel}(\mathcal{M}, U^*, V^*, P)$

$V_i \leftarrow V^*$
$C_i \leftarrow \varnothing$
$\mathcal{A}_i \leftarrow \varnothing$
$variables \leftarrow V^*$
while $variables \neq \varnothing$ **do**
  $v \leftarrow \texttt{pop}(variables)$
  $c \leftarrow \texttt{GetBestConstraint}(v, V_i, U^*, \mathcal{A}, P)$
  $C_i \leftarrow C_i \cup \{c\}$
  $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{(c, v)\}$
  for all $v' \in V_c$ **do**
    if $v' \notin V_i$ **and** $v' \notin \Theta$ **and** $v' \notin U^*$ **then**
      $variables \leftarrow variables \cup \{v'\}$
    end if
    $V_i \leftarrow V_i \cup \{v'\}$
  end for
end while
$\mathcal{M}_i \leftarrow (V_i, C_i, \mathcal{A}_i)$

Subroutine 2 $c = \texttt{GetBestConstraint}(v, V_i, U^*, \mathcal{A}, P)$

$C \leftarrow \varnothing$
$c_v \leftarrow$ **find** c **where** $(c, v) \in \mathcal{A}$
if $(V_{c_v} - v) \subseteq V_i \cup U^*$ **then**
  $C \leftarrow C \cup \{c_v\}$
end if
for all $y \in Y \cap U^*$ **do**
  $c_y \leftarrow$ **find** c **where** $(c, y) \in \mathcal{A}$
  if $v \in V_{c_y}$ **and** $(V_{c_y} - v) \subseteq V_i \cup U^*$ **then**
    $C \leftarrow C \cup \{c_y\}$
  end if
end for
if $C = \varnothing$ **then**
  $c \leftarrow c_v$
else if $c_v \in C$ **then**
  $c \leftarrow c_v$
else
  $C' \leftarrow C$
  for all $c_1, c_2 \in C$ **where** $c_1 \neq c_2$ **do**
    $y_1 \leftarrow$ **find** y **where** $(c_1, y_1) \in \mathcal{A}$
    $y_2 \leftarrow$ **find** y **where** $(c_2, y_2) \in \mathcal{A}$
    if $(y_1 \triangleleft y_2) \in P$ **then**
      $C' \leftarrow C' - \{c_1\}$
    end if
  end for
  $c \leftarrow \texttt{first}(C')$
end if

In general, a variable v is involved in many constraints, however, exactly one of these constraints, in the given causal assignment, computes v. If this constraint does not completely resolve v, we find the constraints in which v is used to compute some output variable $y \in Y \cap U^*$. We consider modifying the causal assignment so that such a y (used now as an input) is used to compute v, instead of v being used to compute y. This can only be performed if, for the causal assignment in which y is being used to compute v, all other variables involved in the constraint are in $V_i \cup \Theta \cup U^*$, in which case this constraint in this new causal assignment can completely resolve v. If no constraint can be found that completely resolves v, then the constraint that in the current causal assignment computes v will have to be used, and backward propagation will be necessary. Otherwise, we select the most preferable constraint that completely resolves v. Preference among constraints (in which an output would be transformed to an input) is computed using a preferences list P, that contains a partial ordering of all the outputs in the model of the form $y_i \triangleleft y_j$, meaning that y_j is preferred over y_i. The subroutine goes through every pair of constraints and removes from the list of most preferable constraints, C', any constraint that uses a measured variable that is less preferable to one involved in another constraint. Of those remaining, an arbitrary choice is made. The preferences list can be used to prefer measured variables with less noise over those with more noise.

In the following sections, we show how this model decomposition approach can be integrated within the HyDE diagnosis framework to reduce the computational complexity associated with the diagnosis of faults in hybrid systems.

4. Integration Proposal

The three main steps in the reasoning process of HyDE are simulation, comparison and candidate generation. These steps are performed for each currently consistent candidate in the candidate set. In this section, we show how the inclusion of structural model decomposition affects each one of these steps, thus proposing a framework where decomposed models can be implemented within HyDE.

In the simulation step, the behavior of the system is simulated using the global model of the system. The goal of the simulation step is to predict expected values of variables in the model that correspond to sensed observations. The main problem regarding this simulation step in HyDE is related to the time and memory performance of HyDE. Our proposal here is to use structural model decomposition so several smaller simulation tasks can be run. The advantage of using minimal submodels for simulation is its smaller size when compared to the size of the global model. However, as we will explain later, computing HyDE models from minimal submodels will affect the comparison and the candidate generation steps in the reasoning process of HyDE as well.

In order to implement minimal submodels in HyDE, we have to look at the models used by HyDE, which are similar to simulation models. They describe the expected behavior of the system under nominal and fault conditions. The model can be constructed in modular and hierarchical fashion by building component subsystem models (which may themselves contain component sub-

system models) and linking them through shared variables/parameters. The component model is expressed as operating modes of the component and conditions for transitions between these various modes. Faults are modeled as transitions whose conditions for transitions are unknown (and have to be inferred through the reasoning process). Finally, the behavior of the components is expressed as a set of variables/parameters and relations governing the interaction among them (for example, equations). The relation between HyDE components and our structural decomposition framework is summarized as follows:

- HyDE model variables are related to variables V in our model.
- The propagation model is specified as constraint predicates over model variables. Constraints may be Boolean expressions if the variables are Boolean; algebraic and ordinary differential equations for interval- and real-valued variables, and equality or inequality for all variables. These are related to the constraints, C, and causal assignments, $\mathcal{A}$, in our model description.
- Candidates k_i in HyDE are related to parameters θ_i in our model.
- The integration model in HyDE is related to variables X in our model.

The comparison step then takes the predictions from the simulation step and the sensed observations and determines if they are consistent with each other or not. This step is performed only for those variables specified to be output variables (some sensed variables are designated inputs and will not be involved in the comparison step). Typically the percentage difference is compared to a threshold defined in the noise characteristics for each sensor specified when building the HyDE model. When HyDE is run without model decomposition only a subset of the sensed variables (those designated as output) are used in comparisons, while with minimal submodels all sensed variables will be used in comparisons. However this overhead is quite insignificant when compared to computational complexity of the simulation and candidate generation steps.

The third and final step is candidate generation, which is typically the most computationally intensive step. When the comparison step results in inconsistencies, a best first search is performed over the unknown transition space to identify potential candidates. When predicted values and sensed observations for a set of variables do not match, then all unknown transitions that could have influenced those inconsistent variables are considered suspects. There are two such flavors of dependencies. A component may have behavioral constraints in the current mode that affect the inconsistent variables and unknown transitions take the component to a different mode that influences the inconsistent variables in a different way. For this a dependency graph that maps dependencies between variables of the system through currently active behavioral constraints is generated. Back propagation through this graph starting from the inconsistent variable, identifies all suspected components. For each suspected component, all unknown transitions from the current mode of that component are selected as potential candidates. Among these transitions those that lead to component modes that influence the inconsistent variable(s) in the same way as the current component mode are eliminated.

The second flavor of influences are from components that do not affect the inconsistent variables in the current mode but have unknown transitions to modes that do influence the inconsistent variables. To identify such components a global dependency graph is generated that maps all dependencies in all modes of all components. Back propagation through this graph would then identify additional potential candidates that could possibly fix the inconsistencies.

When HyDE is used without model decomposition, the dependency graphs and candidate generation represent the entire model, which results in complexity that is exponential in the total number of unknown transitions that influence in the model. After model decomposition the HyDE model is decomposed into independent submodels each of which has its own dependency graph that is not connected to the other submodels. As a result, the candidate state space is significantly reduced. While this approach works for nonsensor faults, sensor faults pose a problem when using a decomposed model. Since a sensed observation can be used as input in other submodels a sensor fault would result in inconsistent variables in all of the submodels involving the sensor as an input or an output. In such cases we need a mechanism to report a single sensor fault instead of a fault from each submodel.

Such a mechanism is implemented in HyDE by representing the sensor as a single component. However inside the component there will be a variable for each submodel that the sensor appears in. When the sensor is used as an observation then its corresponding variable in the HyDE model is marked as an output variable, whereas if the observation is used as an input in the decomposition the corresponding variable is marked as an input variable in the HyDE model. The modes of the sensor component (that include nominal faulty modes) are shared by all of these variables. In other words these variables are connected to the rest of the variables in their submodels through independent behavioral con-

straints in the sensor component's modes. This would result in nonconnected dependency graphs but referring to shared component modes. As a result the back propagation would identify the shared component as a suspect.

Example 1. Consider a sensor component $S1$ with an associated variable $v1$ that appears in two submodels $M1$ and $M2$. In $M1$ it appears as an output variable $v1_o$ and in $M2$ it appears as input variable $v1_i$. Let the output variable associated with $M2$ be $v2$. When $S1$ is faulty then we will notice an inconsistency in the output $M1$ (the predicted value for $v1_o$ would be nominal, but because of the sensor fault, the observed value for $v1_0$ will not be consistent) as well as $M2$ (since we will simulate a faulty $v1$ value through $M2$, the predicted value for $v2$ will not match the observed value). The dependency graph associated with $M1$ will have edges going back from $v1_o$ to other variables represented in relations in $M1$. The edge to $v1_0$ (going back from $v1$) will be labeled as depending on $S1$ being in the nominal node (which is the current operating mode of $S1$). The dependency graph for $M2$ will go backwards from $v2$ and will ultimately reach $v1_i$ through relations represented in $M2$. In this case the edge out of $v1_i$ (going back into $v1_i$) would be labeled as depending on $S1$ too. In this case when we see $v1_o$ and $v2$ inconsistent, $S1$ will be selected as the most likely common explanation (unless there is another double fault with one component fault in $M1$ and another component fault in $M2$ that is more probable as defined by prior probabilities in the model). This example sensor component is illustrated in Fig. 1. The model inside sensor $v1$ is displayed below the $v1$ component for convenience. In the nominal and faulty modes of operation, there will be independent constraints relating $v1_{predictedo}$ with $v1_0$ and $v1_i$ with $v1_{predictedi}$. This will break the propagation path from $M1$ at $v1_o$ and start an independent propagation path from $v1_i$ to $M2$.

This approach allows us to gain the benefits of reduced computational complexity of the model decomposition without adding an additional diagnostic fusion step that might have been necessary if each submodel was completely independent.

5. Case Study

In this section we present our case study, a subset of the Advanced Diagnostics and Prognostics Testbed (ADAPT) (Poll et al., 2007), called ADAPT-Lite, which is an electrical power distribution system. We first briefly present the ADAPT-Lite system and then we show results that we obtained by using our integration approach.

5.1. ADAPT-Lite

A schematic of ADAPT-Lite is given in Fig. 2. Sensors prefixed with an "E" are voltage sensors, those with an "IT" are current sensors, and those with "ISH" or "ESH" are for states of circuit breakers and relays, respectively. TE228 is the battery temperature sensor, and ST516 is the fan speed sensor. Note that the inverter converts DC power to AC, and E265 and IT267 provide rms values of the AC waveforms. Here, v_B and i_B are the battery voltage and current, v_0 is the voltage across C_0, v_s is the voltage across C_s, e is the inverter efficiency, v_{inv} is the inverter voltage on the DC side, R_{inv} is the DC resistance of the inverter, R_{dc} is the DC load resistance, J_{fan} is the fan inertia, and B_{fan} is a damping parameter. Additional details on ADAPT-Lite may be found in (Daigle & Roychoudhury, 2010).

5.2. Diagnosis Results

For the case study we used test scenarios generated for the Diagnostic Competition 2011 (DXC 2011) (Poll et al., 2011). Specifically we used all of the 30 nominal scenarios and picked 66 fault scenarios that considered only discrete, abrupt and persistent faults. For these scenarios we ran the full HyDE model (we will call it HyDE) and the decomposed HyDE model (we will call it HyDE+SMD).[1]. Equations for the ADAPT model and its submodels can be found in (Daigle et al., 2011b). We then compared the diagnosis as well as the number of candidates that were tested before arriving at the diagnosis. For the nominal scenarios both models performed about the same with HyDE+SMD using less computational time. However this time saving was very insignificant (order of milliseconds). One of the reasons for this is that the full ADAPT model is relatively small and behavioral constraints were mostly algebraic.

Both models were tuned to not generate any false positives when run on the nominal scenarios. The results of running the faulty scenarios are presented in Table 1. Each row in the table represents a fault in ADAPT. Regarding the columns, the first column identifies the faulty component and the kind of fault; the second and third columns indicate the time of fault injection and its magnitude; the fourth (resp. seventh) column shows the HyDE (resp. HyDE+SMD) diagnosis result; the fifth (resp. eighth) column indicates the number of candidates that HyDE (resp. HyDE+SMD) needs to explore immediately after the fault detection; the sixth (resp. ninth) column shows the HyDE (resp. HyDE+SMD) classification errors (either a false positive or a false negative); finally, the tenth column shows the difference in the number of fault candidates considered for each one

[1] SMD stands for Structural Model Decomposition.

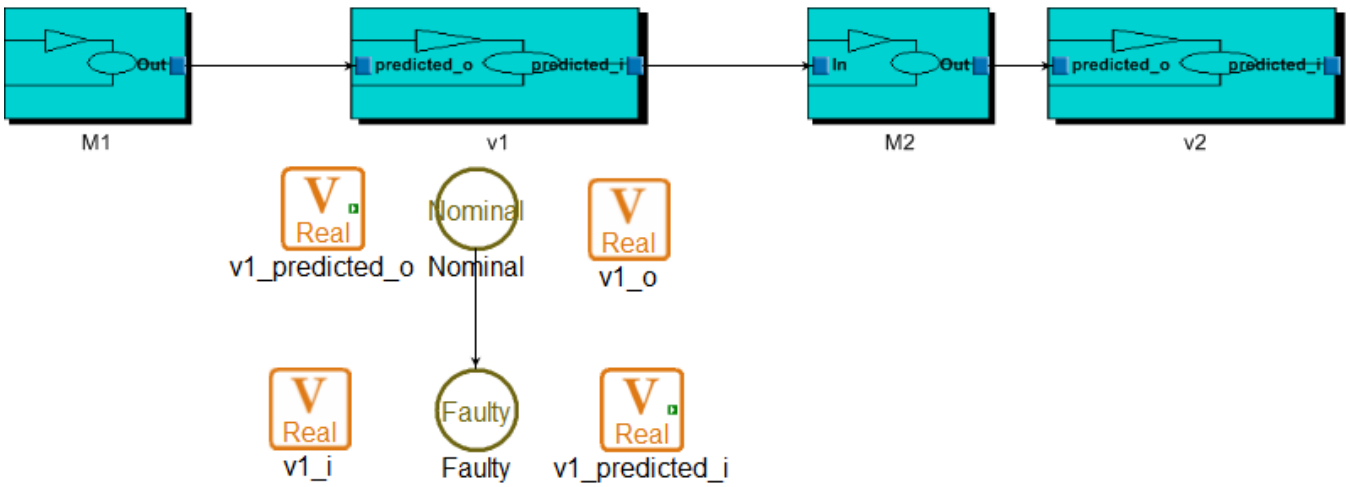

Figure 1. HyDE PC Sensor Model.

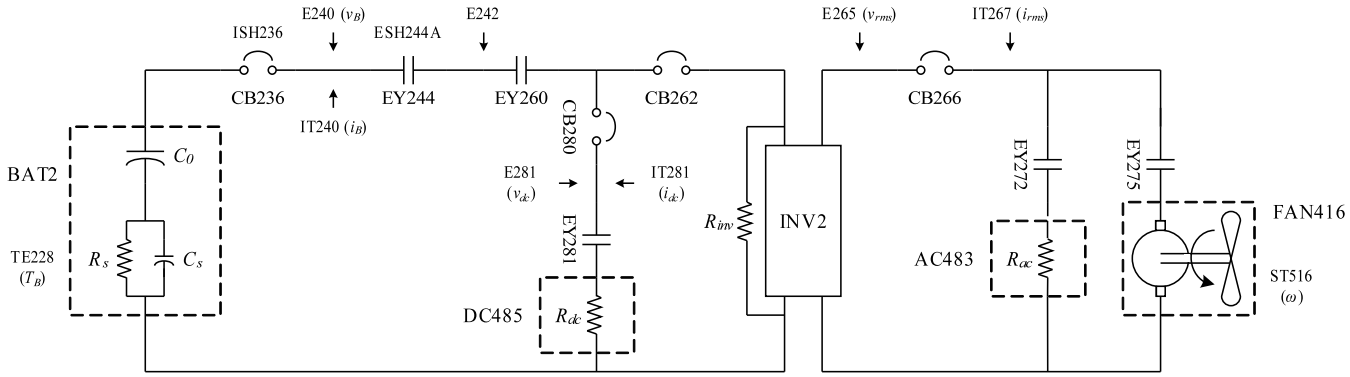

Figure 2. ADAPT-Lite schematic.

of the approaches. For an easier evaluation of the results obtained, Table 2 summarizes these results by giving the total number of candidates tried and classification errors for both of the approaches. Table 2 distinguishes between sensor and nonsensor faults.

Since the candidate generation takes a significant amount of time (order of seconds), the computational time can be considered to be directly proportional to the number of candidates tested. From the results we can see that there are two main advantages from combining HyDE with structural model decomposition.

First we see that the number of errors is reduced from 19 to 11. The reason for this will be apparent when we see how the simulation step is performed in the two cases. When only HyDE is used, the full model is simulated and any errors introduced because of model approximations (parameters in the model are estimated from data and are based on the best fit available and hence are approximate) get propagated through the model and accumulate. As a result at the comparison step some variables are incorrectly determined to be inconsistent when they are not (false positives). This problem can be addressed by increasing the threshold used for comparison but that would lead to some valid inconsistencies to not be detected at all (false negatives). When using HyDE+SMD this problem is substantially mitigated by the fact that simulation results (and any associated errors) do not get propagated to other submodels (instead the actual sensed input values are used). This results in more accurate predictions (assuming sensor values used as inputs are not too noisy) which leads to better diagnostic accuracy.

The second advantage is that fewer candidates are tested in the candidate generation step. As shown in the results, a total of 277 candidates for sensor faults and 44 candidates for nonsensor faults are tested when using HyDE. On the other hand, when HyDE+SMD is used, a total number of 54 candidates are tested for sensor faults and 20 for candidate faults. The reason for this is that the candidate generation step does not have to

Table 1. Diagnosis Results

Fault	At Time	Magnitude	HyDE Diagnosis	HyDE Candidates Tried	HyDE errors	HyDE+SMD Diagnosis	HyDE+SMD Candidates Tried	HyDE+SMD errors	Difference in Candidates Tried
IT240.Offset	72.00	5.40	IT240.Offset	14	0	IT240.Offset	1	0	13
IT240.Offset	101.00	0.30	NONE	0	1	IT240.Offset	1	0	0
E242.Offset	158.00	−2.00	NONE	0	1	E242.Offset	1	0	0
IT240.Stuck	83.00	16.88	IT240.Stuck	15	0	IT240.Stuck	2	0	13
IT267.Offset	192.00	−0.20	NONE	0	1	NONE	0	1	0
IT281.Offset	101.00	1.80	IT281.Offset	7	0	IT281.Offset	1	0	6
ESH244A.Stuck	49.00	0.00	ESH244A.Stuck	2	0	ESH244A.Stuck	2	0	0
IT267.Offset	104.00	0.70	IT267.Offset	11	0	IT267.Offset	1	0	10
IT281.Offset	47.00	0.20	NONE	0	1	NONE	0	1	0
ST516.Offset	168.00	90.00	ST516.Offset	9	0	ST516.Offset	1	0	8
ST516.Offset	121.00	−30.00	NONE	0	1	NONE	0	1	0
ST516.Stuck	58.00	0.00	ST516.Stuck	10	0	ST516.Stuck	2	0	8
ISH236.Stuck	41.00	0.00	ISH236.Stuck	2	0	ISH236.Stuck	2	0	0
ST516.Offset	203.00	−300.00	ST516.Offset	9	0	ST516.Offset	1	0	8
E240.Stuck	102.00	23.90	NONE	0	1	E240.Offset	1	1	0
E242.Stuck	173.00	0.00	E242.Stuck	4	0	E242.Stuck	2	0	2
E265.Stuck	41.00	0.00	E265.Stuck	7	0	E265.Stuck	2	0	5
IT281.Offset	101.00	−0.70	NONE	0	1	IT281.Offset	1	0	0
ST516.Offset	112.00	240.00	ST516.Offset	9	0	ST516.Offset	1	0	8
IT267.Offset	174.00	0.10	NONE	0	1	NONE	0	1	0
E240.Offset	138.00	−5.10	E240.Offset	2	0	E240.Offset	1	0	1
IT267.Offset	187.00	−1.40	IT267.Offset	11	0	ERROR	3	1	8
IT267.Stuck	49.00	2.38	IT267.Stuck	12	0	IT267.Stuck	2	0	10
IT240.Offset	199.00	−1.70	IT240.Offset	14	0	IT240.Offset	1	0	13
IT281.Offset	132.00	−0.05	NONE	0	1	NONE	0	1	0
E281.Stuck	80.00	21.38	ERROR	6	1	E281.Stuck	2	0	4
IT240.Offset	69.00	−4.20	IT240.Offset	14	0	IT240.Offset	1	0	13
IT281.Stuck	152.00	0.00	ERROR	8	1	IT281.Stuck	1	0	7
TE228.Offset	175.00	5.00	TE228.Offset	1	0	TE228.Offset	1	0	0
E265.Offset	39.00	8.00	E265.Offset	6	0	E265.Offset	1	0	5
AC483.FailedOff	79.88	*N/A*	EY272.StuckOpen	1	1	EY272.StuckOpen	1	1	0
DC485.FailedOff	51.73	*N/A*	EY284.StuckOpen	1	1	EY284.StuckOpen	1	1	0
FAN416.FailedOff	87.92	*N/A*	FAN416.FailedOff	1	0	FAN416.FailedOff	1	0	0
INV2.FailedOff	167.99	*N/A*	INV2.FailedOff	1	0	INV2.FailedOff	1	0	0
CB236.FailedOpen	170.97	*N/A*	CB236.FailedOpen	1	0	CB236.FailedOpen	1	0	0
CB262.FailedOpen	188.72	*N/A*	CB262.FailedOpen	1	0	CB262.FailedOpen	1	0	0
CB266.FailedOpen	129.80	*N/A*	ERROR	13	1	CB266.FailedOpen	1	0	12
CB280.FailedOpen	135.03	*N/A*	CB280.FailedOpen	1	0	CB280.FailedOpen	1	0	0
EY244.StuckOpen	35.35	*N/A*	EY244.StuckOpen	1	0	EY244.StuckOpen	1	0	0
EY260.StuckOpen	176.83	*N/A*	EY260.StuckOpen	1	0	EY260.StuckOpen	1	0	0
EY272.StuckOpen	62.87	*N/A*	EY272.StuckOpen	1	0	EY272.StuckOpen	1	0	0
EY275.StuckOpen	141.90	*N/A*	EY275.StuckOpen	1	0	EY275.StuckOpen	1	0	0
EY284.StuckOpen	83.83	*N/A*	EY284.StuckOpen	1	0	EY284.StuckOpen	1	0	0
DC485.FailedOff	59.08	*N/A*	EY284.StuckOpen	1	1	EY284.StuckOpen	1	1	0
FAN416.FailedOff	105.22	*N/A*	FAN416.FailedOff	1	0	FAN416.FailedOff	1	0	0
INV2.FailedOff	120.70	*N/A*	INV2.FailedOff	1	0	INV2.FailedOff	1	0	0
CB236.FailedOpen	35.66	*N/A*	CB236.FailedOpen	1	0	CB236.FailedOpen	1	0	0
CB266.FailedOpen	60.89	*N/A*	ERROR	13	1	CB266.FailedOpen	1	0	12
EY260.StuckOpen	80.06	*N/A*	EY260.StuckOpen	1	0	EY260.StuckOpen	1	0	0
EY272.StuckOpen	39.27	*N/A*	EY272.StuckOpen	1	0	EY272.StuckOpen	1	0	0
ISH236.Stuck	46.00	0.00	ISH236.Stuck	2	0	ISH236.Stuck	2	0	0
ST516.Offset	187.00	−243.00	ST516.Offset	9	0	ST516.Offset	1	0	8
TE228.Offset	101.00	21.00	TE228.Offset	1	0	TE228.Offset	1	0	0
IT240.Offset	203.00	−2.30	IT240.Offset	14	0	IT240.Offset	1	0	13
ST516.Offset	188.00	420.00	ST516.Offset	9	0	ST516.Offset	1	0	8
IT281.Offset	99.00	1.70	IT281.Offset	7	0	IT281.Offset	1	0	6
IT267.Offset	163.00	0.20	NONE	0	1	IT267.Offset	1	0	0
IT267.Offset	146.00	−0.30	IT267.Offset	11	0	IT267.Offset	1	0	10
IT281.Stuck	140.00	0.00	ERROR	8	1	ERROR	2	1	6
IT240.Stuck	95.00	18.26	IT240.Stuck	15	0	IT240.Stuck	2	0	13
E242.Offset	138.00	−3.00	E242.Offset	3	0	E242.Offset	1	0	2
E281.Stuck	83.00	23.42	E281.Stuck	5	0	E281.Stuck	2	0	3
IT240.Offset	178.00	1.50	NONE	0	1	IT240.Offset	1	0	0
IT267.Offset	172.00	−2.00	IT267.Offset	11	0	IT267.Offset	1	0	10
ST516.Offset	131.00	−300.00	ST516.Offset	9	0	ST516.Offset	1	0	8

Table 2. Summary of Diagnosis Results

Kind of Fault	Sum of HyDE Candidates Tried	Sum of HyDE errors	Sum of HyDE+SMD Candidates Tried	Sum of HyDE+SMD errors
Nonsensor faults	44	5	20	3
Sensor faults	277	14	54	8

back propagate past submodel boundaries when using HyDE+SMD. To understand this further first we look at how the unknown transition probabilities are set up. All nonsensor faults are considered to have the same probability and have higher probabilities than sensor faults. Among sensor faults (we consider only offset and stuck) the offset fault is considered more probable than stuck fault. In the full HyDE model when we see some inconsistent variables all components upstream of the sensors have to be considered suspect. However in the case of HyDE+SMD all components upstream of the sensor only in that submodel have to be considered suspect. For sensor faults we see an even more marked improvement in performance because of the special mechanism used to represent sensors in HyDE+SMD. In this case when we see two submodels to have inconsistent variables, the first explanation is the sensor that appears as output in one and input in the other. In the HyDE case all nonsensor faults upstream have to be considered before the sensor fault is considered, resulting in more candidates being tested. For HyDE+SMD we notice that we always test 1 (if actual fault is offset) or 2 (if actual fault is stuck then offset is tested first and then stuck is selected) candidates only.

As examples we will consider one nonsensor fault (DC485 Failed) and one sensor fault (IT281 Offset). The HyDE and HyDE+SMD model fragments containing these two components are illustrated in Fig. 3 and Fig. 4. For the DC485 Failed scenario using only HyDE we see that IT281 and IT240 are inconsistent and HyDE identifies EY284, DC485, CB280, EY260, EY244 and CB236 as possible suspects (based on the intersection of what is upstream of IT240 and IT281). When EY284 is tested it is consistent (EY284 and DC485 failures cannot be distinguished because they do not have any sensors in between them). When using HyDE+SMD only IT281 is detected to be inconsistent and now only EY284 and DC485 are picked as suspects since only those 2 components are present in the submodel that contains IT281 as output. In this case also EY284 is tested first and found to be consistent (resulting in the same diagnostic error due to lack of diagnosability).

When we consider the IT281 Offset scenario, HyDE generates EY284, DC485, CB280, EY260, EY244, CB236 and IT281 as suspects. Since nonsensor faults have higher probability, it considers the 6 nonsensor faults first, but they do not provide consistent predictions. Finally IT281 Offset is selected as a candidate which results in consistency. When HyDE+SMD model is used, IT281 and IT240 are found to be inconsistent. In this case the only intersection when searching for suspects is the IT281 component. Testing the IT281 Offset (because it has higher probability than IT281 Stuck) results in consistency.

6. Related work

Hybrid systems diagnosis has been tackled in different ways. Approaches based in a pure DES following the proposition by (Sampath et al., 1995) model the system as a set of automata, one for each working mode, that tries to track the discrete state, while performing diagnosis as a state-estimation process (Hofbaur & Williams, 2004; Benazera & Travé-Massuyès, 2009). The obvious difference and advantage with HyDE is that it does not need to pre-enumerate modes because they are generated on the fly. Moreover it is not required to generate, track and confirm any potential new discrete state given the ability to track continuous behavior.

Decompositional approaches for continuous systems diagnosis, such as PCs (Pulido & Alonso-González, 2004), ARRs (Staroswiecki & Declerck, 1989), and MSOs (Krysander et al., 2008), have been extended for hybrid systems following somewhat the proposal by (Cocquempot et al., 2004), and their concept of parameterized ARRs (Bayoudh et al., 2009; Moya et al., 2012). The set of ARRs or PCs for any mode must be generated off-line, and the active PCs or ARRs must be derived on-line. The obvious disadvantage is the need to model every potential transition in terms of known or estimated system variables.

There is also the option to combine ARRs and hybrid mode tracking as in (Rienmuller et al., 2013). This work combines hybrid estate estimation which is based on activated or non-activated residuals derived from ARRs for the current system. As in previous approaches, the set of potential states must be taken into account and two different diagnosis processes must be done at the same time to avoid tracking multiple discrete modes.

To avoid enumeration of potential modes, approaches based on Hybrid Bond Graphs, HBGs, adapt the model of the current continuous state by activating/deactivating switching junctions in a Bond-Graph model, and quickly providing a valid causal assignment (Narasimhan & Biswas, 2007). That approach can be combined with system model decomposition such as PCs, in the Hybrid PCs approach, providing a set of subsystems that can track the continuous behavior, while adapting to mode changes thanks to the underlying hybrid bond-graph modeling (Bregon, Alonso, et al., 2012). These HBG based approaches avoid enumeration of modes, but are still linked to one kind of diagnosis algorithm.

Summarizing a main difference between HyDE and the mentioned approaches is that all of them are linked to

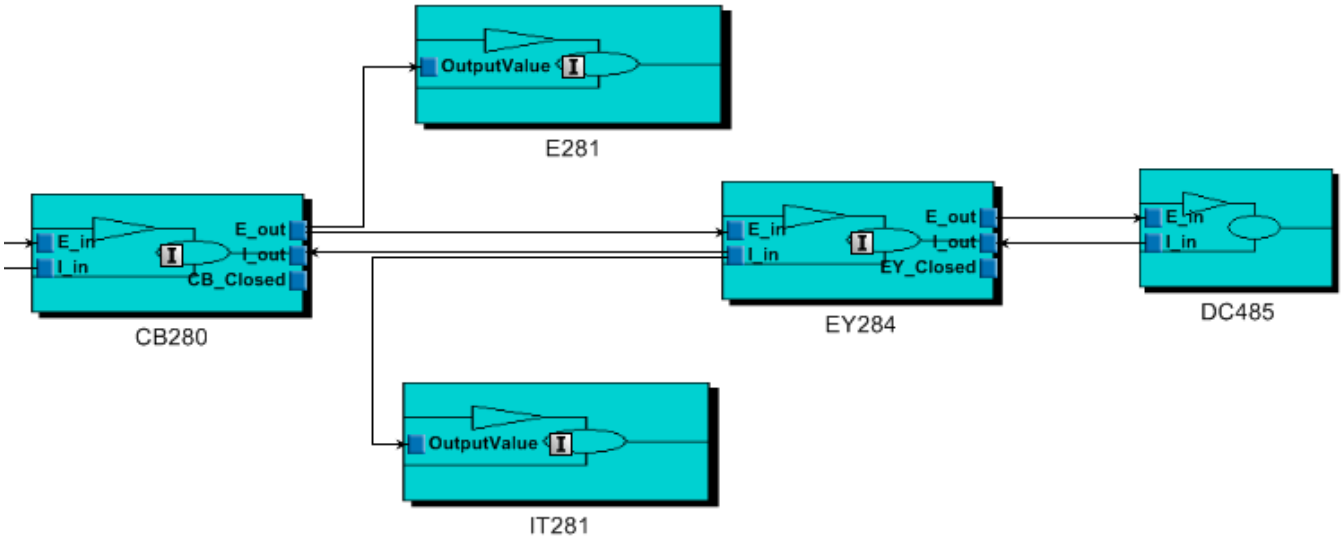

Figure 3. HyDE SMD Sensor Model.

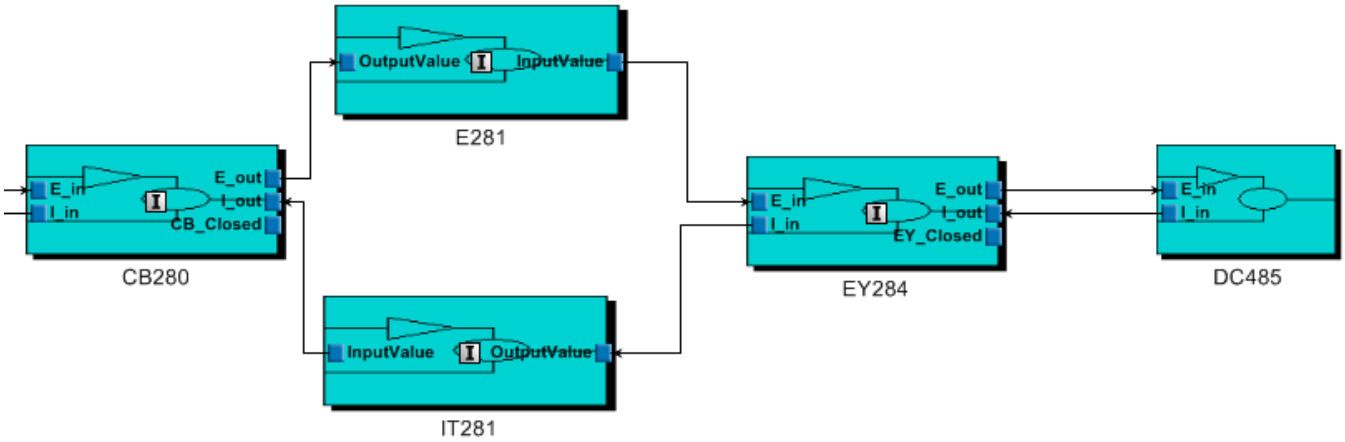

Figure 4. HyDE SMD Sensor Model.

one (or at most two) modeling paradigms, and integrates one diagnosis algorithm.

An implicit assumption in the integration of HyDE and structural model decomposition, due to the potential presence of output sensors as input in the submodels is that sensor noise should not be too high. This is an issue with all model decomposition approaches, because the additional introduction of noisy sensor values as input. This fact provokes sometimes a delay in the detection time, needing a longer period to be sure that the difference in the residual is not related to noise. But this is a common problem in almost any approach to model-based diagnosis, including those without model decomposition.

7. Conclusions

In this paper we presented a method of combining HyDE and structural model decomposition that lets us improve the performance of HyDE under assumptions that sensor noise is not too high. The combined approach results it better diagnosis accuracy as well as reduced computational complexity. We demonstrated this on an electrical testbed at NASA Ames Research Center that has published nominal and faulty data sets as part of the Diagnostic Competition series. In future work we would like to apply this method to other systems, more datasets, and further characterize the improvement in performance. Of particular interest would be multiple fault and increased sensor noise scenarios.

Acknowledgment

A. Bregon and B. Pulido's funding for this work was provided by the Spanish MCI TIN2009-11326 grant. S. Narasimhan, I. Roychoudhury and M. Daigle's funding for this work was provided by the NASA System-wide Safety and Assurance Technologies (SSAT) Project.

References

Bayoudh, M., Travé-Massuyès, L., & Olive, X. (2008). Coupling Continuous and Discrete Event System Techniques for Hybrid System Diagnosability Analysis. In *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence* (pp. 219–223). Amsterdam, The Netherlands, The Netherlands: IOS Press.

Bayoudh, M., Trave-Massuyes, L., & Olive, X. (2009). On-line analytic redundancy relations instantiation guided by component discrete-dynamics for a class of non-linear hybrid systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on* (p. 6970-6975).

Benazera, E., & Travé-Massuyès, L. (2009, October). Set-theoretic estimation of hybrid system configurations. *Trans. Sys. Man Cyber. Part B*, *39*, 1277–1291.

Bregon, A., Alonso, C., Biswas, G., Pulido, B., & Moya, N. (2012). Fault Diagnosis in Hybrid Systems using Possible Conficts. In *Proc. of Safeprocess'2012.* Mexico City, Mexico.

Bregon, A., Biswas, G., & Pulido, B. (2012, May). A Decomposition Method for Nonlinear Parameter Estimation in TRANSCEND. *IEEE Trans. on Systems, Man, and Cybernetics, Part A: Systems and Humans*, *42*(3), 751-763.

Cocquempot, V., El Mezyani, T., & Staroswiecki, M. (2004, july). Fault detection and isolation for hybrid systems using structured parity residuals. In *Control Conference, 2004. 5th Asian* (Vol. 2, p. 1204 - 1212 Vol.2).

Daigle, M., Bregon, A., & Roychoudhury, I. (2011a, September). Distributed Damage Estimation for Prognostics Based on Structural Model Decomposition. In *Proceedings of the Annual Conference of the Prognostics and Health Management Society 2011* (p. 198-208).

Daigle, M., Bregon, A., & Roychoudhury, I. (2011b, Oct). Qualitative Event-based Diagnosis with Possible Conflicts: Case Study on the Third International Diagnostic Competition. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis* (p. 285-292). Murnau, Germany.

Daigle, M., Bregon, A., & Roychoudhury, I. (2012, September). A Distributed Approach to System-Level Prognostics. In *Annual Conference of the Prognostics and Health Management Society 2012* (p. 71-82).

Daigle, M., & Roychoudhury, I. (2010, October). Qualitative Event-based Diagnosis: Case Study on the Second International Diagnostic Competition. In *Proceedings of the 21st International Workshop on Principles of Diagnosis* (pp. 371–378).

Hofbaur, M., & Williams, B. (2004, oct.). Hybrid estimation of complex systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *34*(5), 2178 -2191.

Krysander, M., Åslund, J., & Nyberg, M. (2008). An Efficient Algorithm for Finding Minimal Overconstrained Sub-systems for Model-based Diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics, Part A*, *38*(1).

Mosterman, P. J., & Biswas, G. (1999). Diagnosis of continuous valued systems in transient operating regions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, *29*(6), 554-565.

Moya, N., Pulido, B., Alonso-González, C., Bregon, A., & Rubio, D. (2012). Automatic generation of Dynamic Bayesian Networks for hybrid systems fault diagnosis. In *Proceeding of Intl. Workshop on principles of Diagnosis, DX, 2012.* Great Malvern, U.K..

Narasimhan, S., & Biswas, G. (2007, may). Model-Based Diagnosis of Hybrid Systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *37*(3), 348 -361.

Narasimhan, S., & Brownston, L. (2007, May 29-31). HyDE - A General Framework for Stochastic and Hybrid Model-based Diagnosis. In *Proceedings of the 18th International Workshop on Principles of Diagnosis, DX07* (p. 186-193). Nashville, TN, USA.

Poll, S., de Kleer, J., Abreau, R., Daigle, M., Feldman, A., Garcia, D., et al. (2011, October). Third International Diagnostics Competition – DXC'11. In *Proc. of the 22nd International Workshop on Principles of Diagnosis* (pp. 267–278).

Poll, S., et al. (2007, May). Evaluation, Selection, and Application of Model-Based Diagnosis Tools and Approaches. In *AIAA Infotech@Aerospace 2007 Conference and Exhibit.*

Pulido, B., & Alonso-González, C. (2004). Possible Conflicts: a compilation technique for consistency-based diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics, Part B, Special Issue on Diagno-*

sis of Complex Systems, *34*(5), 2192-2206.

Rienmuller, T., Hofbaur, M., Trave-Massuyes, L., & Bayoudh, M. (2013, March). Mode set focused hybrid estimation. *International Journal of Applied Mathematics and Computer Science*, *23*(1), 131-144.

Roychoudhury, I., Daigle, M., Bregon, A., & Pulido, B. (2013, March). A Structural Model Decomposition Framework for Systems Health Management. In *Proceedings of the 2013 IEEE Aerospace Conference.*

Sampath, M., Sengputa, R., Lafortune, S., Sinnamohideen, K., & Teneketsis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*.

Staroswiecki, M., & Declerck, P. (1989, July). Analytical redundancy in nonlinear interconnected systems by means of structural analysis. In *IFAC Symp. on Advanced Information Processing in Automatic Control.*

Biographies

Anibal Bregon received his B.Sc., M.Sc., and Ph.D. degrees in Computer Science from the University of Valladolid, Spain, in 2005, 2007, and 2010, respectively. From September 2005 to June 2010, he was Graduate Research Assistant with the Intelligent Systems Group at the University of Valladolid, Spain. He has been visiting researcher at the Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA; the Dept. of Electrical Engineering, Linkoping University, Linkoping, Sweden; and the Diagnostics and Prognostics Group, NASA Ames Research Center, Mountain View, CA, USA. Since September 2010, he has been Assistant Professor and Research Scientist at the Department of Computer Science from the University of Valladolid. Dr. Bregon is a member of the Prognostics and Health Management Society and the IEEE. His current research interests include model-based reasoning for diagnosis, prognostics, health-management, and distributed diagnosis and prognostics of complex physical systems.

Sriram Narasimhan is a Project Scientist with University of California, Santa Cruz working as a contractor at NASA Ames Research Center in the Discovery and Systems Health area. His research interests are in model-based diagnosis with a focus on hybrid and stochastic systems. He is the technical lead for the Hybrid Diagnosis Engine (HyDE) project. He received his M.S and Ph.D. in Electrical Engineering and Computer Science from Vanderbilt University. He also has a M.S in Economics from Birla Institute of Technology and Science.

Indranil Roychoudhury received the B.E. (Hons.) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, Rajasthan, India in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, Tennessee, USA, in 2006 and 2009, respectively. Since August 2009, he has been with SGT, Inc., at NASA Ames Research Center as a Computer Scientist. Dr. Roychoudhury is a member of the Prognostics and Health Management Society and the IEEE. His research interests include hybrid systems modeling, model-based diagnostics and prognostics, distributed diagnostics and prognostics, and Bayesian diagnostics of complex physical systems.

Matthew Daigle received the B.S. degree in Computer Science and Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, TN, in 2006 and 2008, respectively. From September 2004 to May 2008, he was a Graduate Research Assistant with the Institute for Software Integrated Systems and Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN. During the summers of 2006 and 2007, he was an intern with Mission Critical Technologies, Inc., at NASA Ames Research Center. From June 2008 to December 2011, he was an Associate Scientist with the University of California, Santa Cruz, at NASA Ames Research Center. Since January 2012, he has been with NASA Ames Research Center as a Research Computer Scientist. His current research interests include physics-based modeling, model-based diagnosis and prognosis, simulation, and hybrid systems. Dr. Daigle is a member of the Prognostics and Health Management Society and the IEEE.

Belarmino Pulido Belarmino Pulido received his Degree, MsC degree, and PhD degree in Computer Science from the University of Valladolid, Valladolid, Spain, in 1992, 1995, and 2001 respectively. In 1994 he joined the Departamento de Informatica in the University of Valladolid, where he is Associate Professor since 2002. He has been working on Model-based reasoning and Knowledge-based reasoning, and their application to Supervision and Diagnosis. He has worked in different national and European funded projects related to Supervision and Diagnosis. He is a member of the IEEE (M'2000), ACM (M'2003), and CAEPIA (1997, part of ECCAI) professional associations. He is also the coordinator of the Spanish Network on Supervision and Diagnosis of Complex Systems since 2005. Currently he is working on model-based diagnosis and prognosis of continuous and hybrid systems, using both centralized and distributed approaches.