HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis

Sriram Narasimhan[1] and Lee Brownston[2]

[1]University of California, Santa Cruz and [2]Perot Systems Government Services, Inc.
M/S 269-3, NASA Ames Research Center, Moffett Field, CA-94035
[1]sriram,[2]lbrownston@email.arc.nasa.gov

Abstract

In recent years several approaches have been proposed for model-based diagnosis of hybrid systems. These approaches deal with discrete or parametric faults, and perform consistency-based, stochastic or mixed reasoning. The major restriction that a diagnosis application designer faces is that each technique uses its own modeling paradigm and the reasoning algorithms implement a single strategy. Diagnosis application designers would like to have the flexibility of building models that are suitable for the task (*e.g.*, appropriate abstraction level, appropriate details, type of modeling paradigm used). They would also like to have the flexibility in choosing the strategies used in the diagnostic reasoning process.

We propose a general framework for stochastic and hybrid model-based diagnosis called Hybrid Diagnosis Engine (HyDE) that offers this flexibility to the diagnosis application designer. We list the key steps in stochastic and hybrid diagnosis and identify the kinds of models that are needed in those steps. 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. We discuss the current status of HyDE and its application in diagnosis of real and conceptual systems.

Introduction

The traditional approach to building a model-based diagnosis system is to select a diagnosis technology, build models targeted for the selected technology, and test the reasoning algorithms on the models using real or simulated data to refine the models and fine tune the performance of the algorithms. This approach works well when the diagnosis application designer is familiar with the diagnosis technology and the modeling paradigm it uses and has a good idea of how to design the models to achieve a specific goal, such as the diagnosis of a specific set of pre-selected faults. In many cases, however, the diagnosis problem is not so well defined and the diagnosis designer would like to have the flexibility to experiment with different kinds of models (*e.g.*, multiple paradigms and abstraction levels) as well as different strategies for diagnosis reasoning (*e.g.*, pure consistency-based, purely stochastic and different kinds of search algorithms).

In this paper we present the Hybrid Diagnostic Engine (HyDE), a general framework for stochastic and hybrid model-based diagnosis of discrete faults, that is, spontaneous changes in operating modes of components. HyDE combines ideas from consistency-based [Hamscher, *et al*., 1992; De Kleer and Williams, 1987] and stochastic [Hofbaur and Williams, 2002; Dearden and Clancy, 2002] approaches to model-based diagnosis using discrete [Williams and Nayak, 1996; Kurien and Nayak, 2000], continuous [Gertler, 1988; Mosterman and Biswas, 1999] and hybrid [Narasimhan and Biswas, 2003; Hofbaur and Williams, 2002; Dearden and Clancy, 2002] models to create a flexible and extensible architecture for stochastic and hybrid diagnosis. HyDE supports the use of multiple paradigms and is extensible to support new paradigms. HyDE offers the application designer a wide variety of options in selecting the diagnosis reasoning strategy. 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.

Some preliminary work related to HyDE has been presented in [Narasimhan, *et al*., 2003; Narasimhan, *et al*., 2004].

The following sections first present the kinds of models used in HyDE, each of which includes a generic transition model and a modeling-paradigm-specific behavior model. Then we describe the kinds of faults HyDE can diagnose and how they are represented in the models. Next we discuss the HyDE reasoning architecture and how it supports the use of diverse algorithms, enabling various diagnosis strategies. We then identify the features currently implemented and finally describe some applications that use HyDE for diagnosis.

HyDE Models

HyDE models have two parts. The first, which is common to all supported modeling paradigms, describes the transition behavior of the system. The second is the behavior model, which is specific to each modeling paradigm.

The transition model describes the following elements of the system:

1. The set L of operating modes l_i of the system, $1 \leq i \leq |L|$, called *Locations*.
2. The set T of allowed *transitions* t_i, $1 \leq i \leq m$, between the locations of the system, where $t_i = l_j \rightarrow l_k$ indicates a transition from location l_j to l_k.

The behavior model specifies the behavior evolution and has three parts: propagation model, integration model and dependency model. Each kind of model part is used for a different step in the diagnosis reasoning process. 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. Depending on the modeling paradigm used, the same model may serve all three roles or it may be necessary to specify each model independently. The integration model is necessary only if there are variables that have state, *i.e.*, if values at one time step depend on values at previous time steps. The dependency model is optional, since a fully-connected graph may be used as a trivial dependency model, but Candidate Generation can be optimized if a dependency model is specified. The behavior model is expressed as:

1. The set V of *variables* v_i in the system, $1 \leq i \leq |V|$.
2. The set of D *domains* d_i, $1 \leq i \leq |V|$, specifying the allowed values (data types) for the variables, where d_i represents the domain for variable v_i.
3. The set G of transition *guards* g_i, $1 \leq i \leq |T|$, representing the conditions for system transitions, where g_i is a condition predicate for transition t_i.
4. The *propagation model* PM specifies the behavior of the system within a time step[1] as relations over variables. This includes
    a. Global model $PM_g = R_g(V)$, where R_g is the set of relations constraining values of variables expressed in one of the supported modeling paradigms.
    b. Local models $PM(l_i) = R_{li}(V)$ for each $l_i \in L$, where R_{li} is the set of relations constraining values of variables expressed in one of the supported modeling paradigms.
5. The *integration model* IM specifies the evolution of values each variable across time steps. $IM(v_i) = R_t(v_i(t_k), \delta v_i(t_k), v_i(t_{k-1}), \delta v_i(t_{k-1}), \ldots, v_i(t_{k-n}), \delta v_i(t_{k-n}))$, where R_t is a relation, $v_i(t_k)$ is the instance of variable v_i at time t_k, $\delta v_i(t_k)$ is the instance of the derivative of v_i at time t_k.
6. The *dependency model* DM specifies dependencies among variables in V, relations R_g and R_{li} associated with the propagation model and relations R_t associated with the integration model.

For the sake of modularity, composability and hierarchy, HyDE supports the use of component models as well as system models. Component models specify the system model in terms of the transition and behavior models of the individual components and the interconnections among components. Diagnostic reasoning may be performed directly on component models in some cases, but in most cases the component models have to be composed to a higher-level system model before use. Users can choose to build system models directly. The system model is composed from the component model as follows:

- The transitional model for the system is a synchronous composition of the transitional models of the components and is derived as follows:
    - $SL = L_1 \times L_2 \ldots \times L_n$ where 1, 2 … n are the indices of the components making up the system. A system location sl_1 would be $(l_{1i}, l_{2j}, \ldots, l_{nk})$, where the first subscript indexes the component and the second subscript indexes a location in that component.
    - The system transitions T_s are derived as follows: For each transition for component c $l_{ca} \rightarrow l_{cb}$, corresponding transitions are created from every location in the system that includes l_{ca} in the label to the location that contains the label l_{cb}, with the remaining label being the same.
- The composition of the behavior model is a specialized algorithm that depends on the type of modeling paradigm used for the system model. The system behavior model is derived as follows:
    - $V_s = V_g \cup V_1 \cup V_2 \ldots \cup V_n$, where 1, 2 … n are the indices corresponding to the components making up the system and V_g represents the set of variables that do not belong to any component.
    - The transition guards remain the same for corresponding transitions.
    - For each location $l_s = \{l_{1i}\, l_{2j} \ldots l_{nk}\}$. $M_s = M_g \cup M_{connection} \cup M_{1i} \cup M_{2j} \ldots \cup M_{nk}$ where the union operation U depends on the modeling paradigm used for component and system models. $M_{connection}$ is the component connection model that specifies how components interact with each other. M represents propagation, integration and dependency models.

[1] Variables are assumed to take one or more values from their domains at each time step in the reasoning process described later.

HyDE also supports multiple behavior model paradigms at the component and system level. Component models may be specified in one paradigm and then transformed to another form before composition to system model form; or the component models may be transformed to a system model in one paradigm, which can then be transformed to a system model in another paradigm. Figure 1 illustrates the interaction between component and system models. The following convention is used in Figures 1 through 5.

Both the transformation and composition may be performed in either order when needed, rather than pre-compiling system models in all possible system locations.

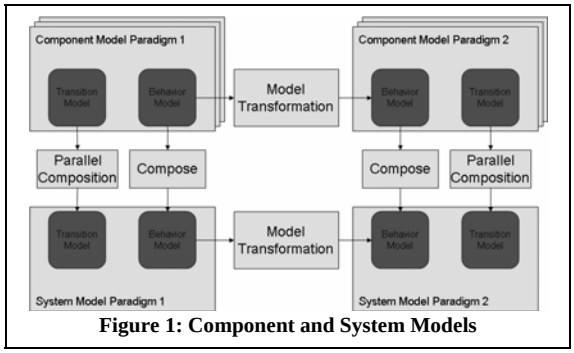

Figure 1: Component and System Models

HyDE Faults

HyDE can deal with discrete faults, that is, those that correspond to an abrupt change in the configuration of the system. HyDE models represent discrete faults as transitions without guards (called unguarded transitions). The absence of guards signifies that the occurrence of these transitions cannot be directly observed, and hence must be determined by reasoning about symptom observations. Examples of such faults are valve stuck open, motor stalled and circuit breaker tripped. Unexpected behavior not explicitly modeled can be handled as an unguarded transition to a unique "unknown" location that has no model associated with it, assuring that it will be consistent with all observations. It is also possible to model parametric faults (abrupt changes in the values of system parameters) if the new values for the parameters are known. For example, we can model resistive faults as an n% change in resistance for several pre-specified values of n. It is not possible to model the general case in which n is not known and has to be inferred by reasoning.

The use of unguarded transitions allows HyDE to infer the occurrence of any unobserved event, not just faults. For example, a transition that is based on a supervisory controller command may be represented as an unguarded transition if the issuance of the command is not observed.

HyDE Reasoning

HyDE reasoning is the maintenance of a set C of weighted candidates (c_i, w_i), $1 \leq i \leq k$. A candidate represents the hypothesized alternative trajectories of the system inferred from the transition and behavior models of the system, knowledge of the initial locations 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, specifically posterior probability.

Each candidate contains a possible trajectory of system behavior evolution represented in the form of a *hybrid state* (HS) 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 locations and all variables with their current values. HS = (SL, VV) where $SL = \{(comp_i \rightarrow l_i) \mid 1 \leq i \leq n\}$, and $VV = \{(v_i \rightarrow value_i) \mid 1 \leq i \leq m\}$. 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.

If HyDE has been run for a set of time steps $\{t_i \mid 1 \leq i \leq end\}$, a candidate will be of the form $\{(Trans_{\rightarrow ti}, hs_{ti-}, hs_{ti+}) \mid 1 \leq i \leq end\}$, where $Trans_{\rightarrow ti} = \{(comp_i, trans_i) \mid 1 \leq i \leq end\}$ is a set of ordered transitions (with associated component) believed to have been taken by the system between time steps t_{i-1} and t_i, hs_{ti-} indicates the hybrid state at the beginning of time step t_i, and hs_{ti+} indicates the hybrid state at the end of time step t_i. All components that do not have an explicit transition in $Trans_{\rightarrow t}$ are assumed to have taken an implicit special guarded transition called self-transition $trans_{li}$. This transition is from location l_i to location l_i. As a result, $Trans_{\rightarrow t}$ will include an entry for all components but may have more than one entry for some components if it is believed that more than one transition occurred for those components.

At time step 0 (or some user-specified start time step t_i) the candidate set is initialized with candidate(s) derived from the initial hybrid state of the system. If there is some uncertainty about the initial hybrid state, it is possible to sample from this uncertainty to create a set of initial candidates. In the worst case, when the initial hybrid state is not known, a set of candidates may be created by randomly sampling the locations of the components. $Trans_{\rightarrow 0}$ is set to contain the self transition for all

components. After initialization a candidate c_i would look like $\{(Trans_{\rightarrow 0}, hs_{0-}, hs_{?})\}$, where $hs_{?}$ indicates an unknown or not yet estimated hybrid state. The weights of the candidates may be set to 1; if prior probabilities are available for initial hybrid states, those values can be used as initial weights.

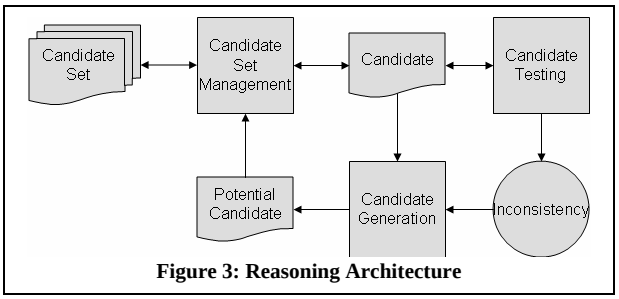

Figure 3: Reasoning Architecture

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, as illustrated in Figure 3:

1. Candidate Set Management maintains the candidate set, including pruning unlikely candidates and adding new candidates when necessary.
2. Candidate Testing deals with operations on a single candidate, including estimation of the hybrid state, updating the weight of candidate and 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.

In the next three sections we will discuss each of these categories in more detail.

Candidate Set Management

There are four Candidate Set Management operations, shown in sequential order in Figure 4.

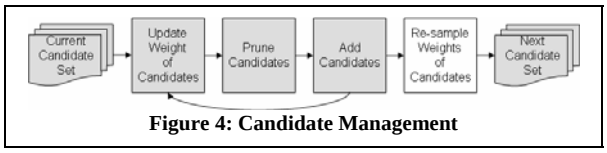

Figure 4: Candidate Management

Updating weight of all candidates. The Candidate Testing operations are called to update the weight of each candidate in the candidate set. When candidates are tested, additional candidates may be spawned (described in the candidate-testing section). Based on user preferences, these candidates may be added to the candidate set for testing; incorporated into a candidate generator to be re-generated when requested (for later testing); or completely ignored.

Prune candidates. After the candidate weights have been updated, candidates with weights below a user-specified threshold t_w are pruned from the candidate set: $C \leftarrow \{ c_i \mid c_i \in C \ \& \ w_i >= t_w \}$.

Add candidates. Additional candidates may be needed to fill the candidate set to a user-specified minimum count. To re-fill the candidate set, a new potential candidate is requested from the bank of candidate generators created as a by-product of the candidate-testing operations. The candidate generators are responsible for selecting the next best candidate based a user-specified ranking function. The potential candidate then goes through Candidate Testing and may be pruned if its updated weight is not high enough. More potential candidates are requested until the candidate set has the requisite number of candidates.

Re-sample weights. As an optional step, candidate weights may be normalized by re-sampling from the distribution of weights. For the candidate set $C = \{(c_i, w_i) \mid 1 \le i \le k\}$, the sum $W = \sum w_i$, $1 \le i \le k$ is determined and the candidate set is cleared $C = \{\}$. A random real number x is sampled between 0 and W. If the value of x is between $\sum w_i$, $1 \le i \le j-1$ and $\sum w_i$, $1 \le i \le j$, candidate $(c_j, 1)$ is added to the candidate set. This process is repeated until the candidate set has the requisite number of candidates (usually k).

Candidate Testing

At each time step t_i, each candidate c_j is tested to find the enabled transitions $Trans_{\rightarrow ti}$ taken between the previous time step t_{i-1} and the current time step t_i; to estimate the hybrid state at the beginning of the time step hs_{t1-}; to compose the system propagation model PM_{ti}; to estimate the hybrid state at end of time step hs_{t1+}; update the weight w_j of the candidate; and to report any inconsistencies I_{ij}. These steps are illustrated in Figure 2.

Find enabled transitions. First collect the set of all possible transitions out of the system location associated with hs_{ti-1}. The user has the option of including unguarded transitions in this list. The guards on the selected transitions are evaluated to compute a weight indicating the

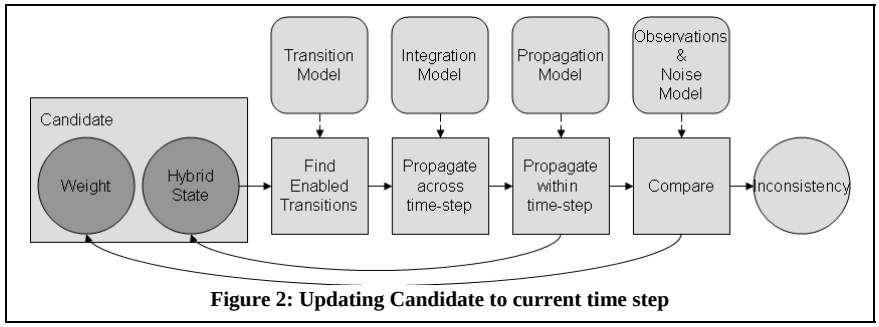

Figure 2: Updating Candidate to current time step

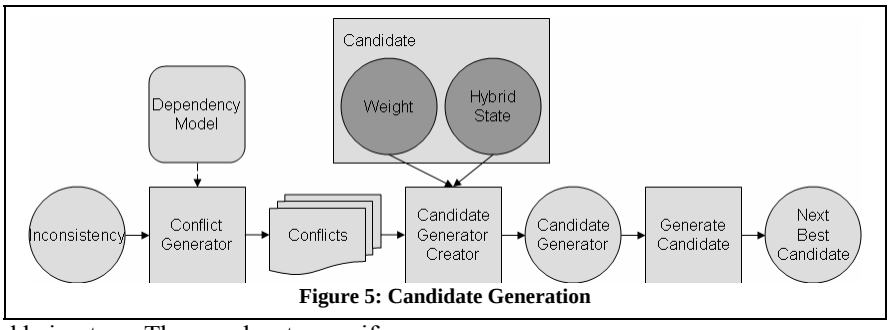

Figure 5: Candidate Generation

likelihood of the guard being true. The user has to specify a policy for handling enabled transitions. The options are to consider only the most likely transition with weight > 0.5, choosing the self transition if no such transition exists; sampling from the distribution of enabled transitions; selecting all transitions with weight > 0.5; or selecting all enabled transitions irrespective of weight.

Estimate beginning hybrid state. The estimation of the hybrid state at the beginning of the time step t_i involves two operations.

1. Estimate the new system location: First a new $Trans_{\rightarrow ti}$ is created for each enabled transition in the previous step. $Trans_{\rightarrow ti}$ is applied to $SL(hs_{ti-1})$ to determine $SL(hs_{ti-})$. There will be as many new $SL(hs_{ti-})$ as there are enabled transitions. As mentioned earlier, all but the most likely one are reported to Candidate Set Management, which decides what will be done with them.
2. Estimate the new variable values: The integration model IM_{ti} is applied to $VV(hs_{ti-1})$ to get $VV(hs_{ti-})$. Additionally, values for any input variables are set to be their sensed values, if reported, at t_i.

Load the propagation model. The possibly-new propagation model of the system corresponding to $SL(hs_{ti-})$ has to be loaded. If $SL(hs_{ti-})$ has been reached earlier in the reasoning process, its propagation model PM_{ti} has already been cached and can be loaded directly. If $SL(hs_{ti-})$ has not been reached, PM_{ti} has to be composed from the model associated with $SL(hs_{ti-})$, as described in the modeling section. The composed model is cached for later re-use.

Estimate end hybrid state. The next step is to estimate hs_{ti+} using PM_{ti}. Assuming that all transitions occur between successive time steps, we set $SL(hs_{ti+}) = SL(hs_{ti-})$. $VV(hs_{ti+})$ is estimated by initializing PM_{ti} with $VV(hs_{ti-})$ and executing PM_{ti} by running a simulation or propagation algorithm. If the execution fails, an Inconsistency I_{ti} is reported to the Candidate Generation system. This inconsistency identifies the relation $r_{inconsistent}$ from PM_{ti} that did not hold.

Compare against observations. If output variables have been reported, HyDE compares these observed values Y_n to their predicted values $\hat{Y}_n$ in hs_{ti+}. The comparison step for n output variables is expressed as: $R_n = CF(\hat{Y}_n, Y_n, \varepsilon_n)$, where CF is a comparison function, R_n is a vector of n residual values in [0, 1] indicating the degree of fit and ε_n are the user-defined noise models for the n sensors associated with the output variables.

Update weight of candidate. An overall degree of fit $r_{overall}$ is also computed from R_n. The user may select this to be either the minimum weight in R_n, the arithmetic mean of R_n or some custom function over $(\hat{Y}_n, Y_n, \varepsilon_n)$. The weight of the candidate is updated by multiplying by $r_{overall}$.

Reporting inconsistency. If $r_{overall}$ is less than a user-specified threshold, an Inconsistency I_{ti} is reported that contains all output variables whose degree of fit $r_i \in R_n$ is less than the same user-specified threshold.

Candidate Generation

The Candidate Generation system is responsible for creating a new potential candidate when requested by the Candidate Set Management system. Figure 5 illustrates the steps in Candidate Generation. The Candidate Generation system consists of a set of candidate generators. When a candidate is requested, each candidate generator reports its next best candidate, where "best" is a user-customizable criterion, and the best among these best candidates is selected and returned to the Candidate Set Management system.

Candidate generators come in two flavors. The first kind, based on a candidate, always reports that candidate until the candidate is selected for use. These kinds of candidate generators handle the situation in which Candidate Testing results in the spawning of new candidates and the user specifies that these candidates go into the set of candidate generators. In such a case each newly-spawned candidate becomes the basis for a candidate generator.

The second kind of candidate generator is based on *conflicts*. A conflict is a list of timed transitions which, if taken jointly by the system, result in an inconsistency. When the propagation or comparison step generates inconsistencies, conflicts are generated using the dependency model and become part of a new candidate generator. The next section describes how a conflict is created from inconsistencies and the section following describes how the conflict-based candidate generator works.

Conflict generation. When an inconsistency I_{ti} is reported at time t_i from the propagation or comparison step, HyDE creates (or retrieves if already created) the dependency model DM_{ti} corresponding to $SL(hs_{ti})$. For each component encountered, the transitions for that component from

$Trans_{\rightarrow ti}$ are added to the conflict set that has time stamp t_i. The same process is repeated in the dependency model DM_{ti-1} for the previous time step t_{i-1}. However, the starting points for the back traversals in the dependency model are all the variables that depend on I_{ti}. This adds transitions with time stamp t_{i-1} to the conflict set. The same procedure is repeated for n previous time steps, where n is a user-supplied parameter. The user may explicitly specify the amount of time to backtrack and the maximum number of guarded transitions to consider. When the inconsistency contains two or more variables, a conflict is created for each variable. A conflict-based candidate generator is then created containing an initial hybrid state hs_{ti-n} at time step t_{i-n}, an initial weight equal to the weight of the candidate that generated the inconsistency, and the sets of conflicts generated from the inconsistencies.

Generating a candidate from a conflict-based candidate generator. When requested for a candidate, the candidate generator generates a set $Trans_{unguarded}$ of timed unguarded transitions $\{trans_{ti} \mid 1 \leq i \leq k\}$ that resolve all the conflicts in the candidate generator. A conflict is resolved by a transition that is a sibling (same source location, different destination location) of at least one transition in the conflict. A set of transitions resolves a set of conflicts if, for each conflict, there is a transition in the set of transitions that resolves the conflict. The Candidate Generator saves its state so that the next time it is asked it can generate the next optimum candidate. The transitions are chosen so as to optimize some user-defined criteria. For example, the user may be interested in transitions with the maximum prior probability or the set of transitions with minimum size. Conflict resolution may use a two-step hitting-set-based solution (generate a hitting set and resolve the hitting set) or conflict-directed search [Williams and Ragno, 2003] to generate the transitions directly. A candidate is then created with initial hybrid state hs_{ti-1}, and pre-loaded $Trans_{\rightarrow t}$ for each t in $\{t_{i-n}, \ldots, t_i\}$. Each entry $trans_{tk}$ in $Trans_{unguarded}$ is added to $Trans_{\rightarrow tk}$.

HyDE Implementation Status

The HyDE reasoning engine is implemented in C++. Complete diagnosis reasoning can be performed although not all of the earlier-discussed capabilities have been implemented yet. It passes an extensive and demanding test suite on Windows, Solaris, Linux and VxWorks platforms. A graphical modeling environment is available using the GME open-source tool [Ledeczi, *et al*., 2001]. The same environment can also be used to set initial state and configuration parameters. Observations can be reported to HyDE either through streams (file or otherwise) or an API allowing integration with a real-time system.

HyDE model variables can have Boolean, Enumeration, Interval or Real domains. For Enumeration domains, the domain's values must be specified. The within-component transition model is specified as a finite-state automaton in which locations are the states and the transition guards may be specified either as commands, some predicate over model variables, or a combination of the two. 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. If all variables are interval or real, the user can configure HyDE to solve the constraints symbolically to transform them into state space equations or a Kalman Filter when used for propagation. Currently the Euler integration technique is the only integration model. The dependency model can either be real-time justification (truth maintenance) mechanism or a dependency graph that is derived from the constraints.

HyDE currently supports only consistency-based Candidate Set Management. Candidates are added to the candidate set only if consistent and deleted only if inconsistent. The maximum number of candidates in the candidate set is configurable. In Candidate Testing, only identification of the first enabled transition is supported. Comparison may use an equality check or thresholding to make a binary determination of consistency. Candidate Generation uses best-first search with a configurable preference for what is considered best. The user can set several parameters, including maximum candidate size, minimum candidate weight, and time of a candidate's last-hypothesized fault transition to guide the search.

HyDE Applications

HyDE has been and is being used on several projects. These projects exercise very different capabilities of HyDE, affirming the need for such a general framework. The following three projects have successfully used HyDE in demonstrations using real-time telemetry streaming from the system being diagnosed.

1. **The Drilling Automation for Mars Environment (DAME)** project, led by NASA Ames Research Center, is aimed at developing a lightweight, low-power drill prototype that can be mounted on a Mars Lander and drill several meters below the Mars surface for conducting geology and astrobiology research. Three kinds of diagnosis technologies were used on this project, HyDE for

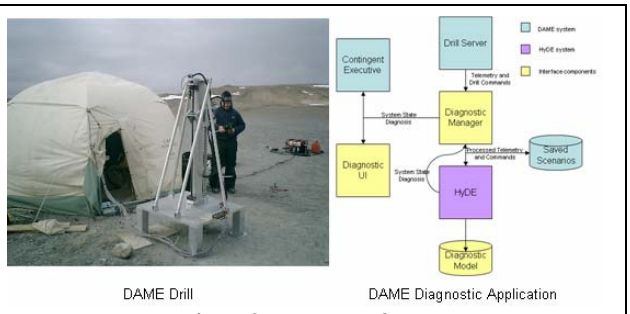

Figure 6: HyDE on DAME

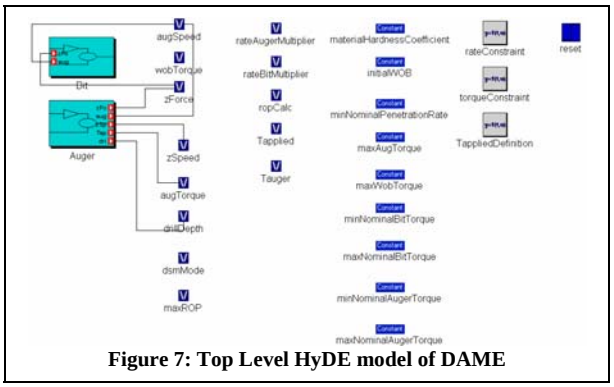

Figure 7: Top Level HyDE model of DAME

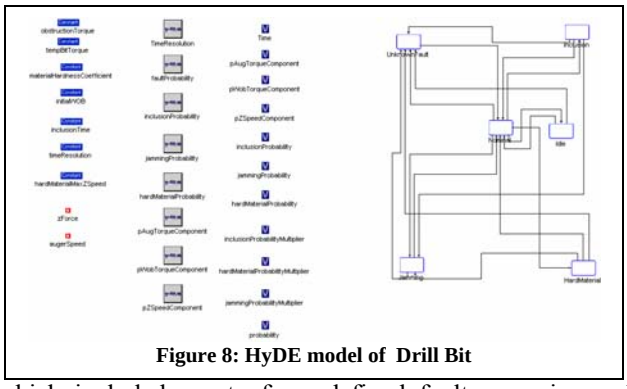

Figure 8: HyDE model of Drill Bit

model-based diagnosis, a rule-based diagnosis system, and a neural-network diagnosis system. The drill and the diagnostic application using HyDE are illustrated in Figure 6. HyDE was to model only the two major components of the drill, the bit and the auger (top level model shown in Figure 7). Nevertheless the model was complex, requiring 73 variables and 162 differential and algebraic constraints. The model had nominal (nominal, idle), faulty (jamming, inclusion unknownFault), and unobserved (HardMaterial) locations. The bit model is shown in Figure 8.

There were four rounds of testing over a period of two years. In 2005, laboratory experiments were run at Honeybee Robotics, the company that constructed the drill. Later in 2005, field tests were performed at Houghton Crater on Devon Island, in the Canadian arctic, chosen to approximate the Martian terrain and climate. Based on the results, laboratory tests were performed at NASA Ames Research Center in 2006 to improve the models for better diagnosis. Finally, there was a second field test at the Houghton Crater. [Balaban, *et al.*, 2007] summarize HyDE's final performance:

> All of the modeled fault modes were encountered in the field; some, such as choking, binding, and hard material, numerous times. The model-based diagnostic system was able to successfully identify the faults in roughly 85% of the cases. The rate of false positive diagnoses was approximately 5%.

2. **The Advanced Diagnostic and Prognostic Test-bed (ADAPT)** was developed at NASA Ames Research Center with the goal to test, measure, evaluate, and mature diagnostic and prognostic health-management technologies. The test-bed hardware for generates, stores, distributes, and monitors electrical power. The initial test-bed configuration is functionally representative of an exploration vehicle's Electrical Power System. HyDE was used to build an 86-component enumeration constraint model of the test-bed components. The ADAPT is in stark contrast to the DAME model in that it contains a lot of components but very few variables and constraints in the components. Diagnosis was purely consistency-based. In addition, modeling used a special feature of HyDE that allows observations to be made input variables, enabling the modelers to build consistency models instead of predictive models. HyDE passed all the acceptance tests, which included a set of pre-defined fault scenarios and acceptance bounds on the time to diagnosis. Additional tests were run on other fault scenarios (not included in acceptance tests) and HyDE was able to diagnose all but one of these scenarios (resulting from inadequate information in the model). More details of HyDE on ADAPT are presented in [Scott Poll, *et. al.,* 2007].

3. **The Autonomous Lander Demonstrator project (ALDER)** demonstrated autonomy capabilities relevant to a small spacecraft mission on traditional flight hardware integrated with traditional flight software. Diagnosis systems, including HyDE, were ported to VxWorks and executed on the Aitech S950 processor, interoperating with autonomy technologies for planning, diagnosis, computer vision and adaptive control. A HyDE component model of a simple propulsion system (tanks, valves, regulators, attitude control system and main engine) detected common faults such as stuck valves and regulator failures.

Several other projects are using HyDE for offline diagnosis using simulated data. The International Space Station (ISS) Electrical Power System (EPS) recovery procedure automation project at NASA Ames Research Center uses HyDE to supply current hybrid state to an execution system that will attempt to partially automate recovery procedures based on this information. The Aircraft Landing Gear Diagnosis project at NASA Langley Research Center is using HyDE to model the landing gear and wheels of an aircraft in an effort to diagnose faults during landing. The Spacecraft Engine Diagnosis project at NASA Marshall Space Flight Center uses HyDE to model components of the J2X engine, which is expected to power the upper stage of the NASA Crew Launch Vehicle (CLV). HyDE was also integrated with the CLARAty (Coupled Layer Architecture for Robotic Autonomy) architecture at NASA Jet Propulsion Laboratories.

Conclusions and Future Work

We presented the Hybrid Diagnosis Engine (HyDE), a general framework for stochastic and hybrid model-based diagnosis. HyDE supports multiple modeling paradigms and multiple strategies for the steps in the diagnosis reasoning. HyDE is extensible by adding new modeling

paradigms and user-defined algorithms for diagnosis reasoning steps. HyDE is being used on several projects that span the spectrum from consistency-based reasoning to stochastic reasoning and discrete models to hybrid models.

HyDE is still a work in progress and we expect to keep adding to the capabilities to HyDE through addition of new modeling paradigms and algorithms. We have identified several major areas for future work. We would like provide mechanisms for validation and verification (V&V) of models and algorithms, which would be prerequisite for deployment on real systems. We would like to add support for parametric faults. This would require support for modeling of parameters representing such faults, additional algorithms for parameter estimation and modifications to existing algorithms for detection and isolation of such faults. We would also like to use the existing models and algorithms for suggesting recovery sequences. Livingstone 2 [Kurien and Nayak, 2000] demonstrated how this can be done for finite-domain models. We would like to extend this to HyDE models and algorithms.

Acknowledgements. We thank the following people for providing us with information and pictures on the use of HyDE on their projects: DAME: Howard Cannon (NASA Ames Research Center), Edward Balaban (Perot Systems Government Services, Inc.); ADAPT: Scott Poll (NASA Ames Research Center), Adam Sweet (Perot Systems Government Services, Inc.); and ALDER: Jeremy Frank (NASA Ames Research Center), Edward Balaban (Perot Systems Government Services, Inc.). We also acknowledge the help of the following people in supplying information on the use of HyDE: Peter Robinson (Science Applications International Corp.), Vandi Verma (Perot Systems Government Services, Inc.), Cuong C. Quach and Sixto Vazquez (NASA Langley Research Center), Jon Patterson (NASA Marshall Space Flight Center), Bradley Burchett (Rose-Hulman University), and Thomas Slack (University of Memphis).

References

[Hamscher, *et al*., 1992] Walter Hamscher, Luca Console, and Johan De Kleer. *Readings in Model-based Diagnosis.* San Mateo, CA: Morgan Kaufmann, 1992.

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

[Hofbaur and Williams, 2002] Michael Hofbaur and Brian Williams. *Mode Estimation of Probabilistic Hybrid Systems*, in Proc. 5th International Workshop on Hybrid Systems: Computation and Control (HSCC '02), Stanford, CA, USA, pp. 253-266, 2002.

[Dearden and Clancy, 2002] Richard Dearden and Dan Clancy. *Particle Filters for Real-time Fault Detection in Planetary Rovers*, in Proc. 13th International Workshop on Principles of Diagnosis (DX '02), Semmering, Austria, pp. 1-6, 2002.

[Williams and Nayak, 1996] Brian Williams and Pandu Nayak. *A model-based approach to reactive self-configuring systems*, in AAAI, pp. 971–978, (1996).

[Kurien and Nayak, 2000] James Kurien and Pandu Nayak. *Back to the Future with Consistency-based Trajectory Tracking*, AAA/IAAI 2000, pp370-377.

[Gertler, 1988] Janos Gertler. *Fault Detection and Diagnosis in Engineering Systems.* New York: Marcel Dekker, 1988.

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

[Narasimhan and Biswas, 2003] Sriram Narasimhan and Gautam Biswas. *Model-based Diagnosis of Hybrid Systems.* Eighteenth Intl. Joint Conf. on Artificial Intelligence, Acapulco, Mexico, Aug., 2003.

[Narasimhan, *et al*., 2003] Sriram Narasimhan, Lee Brownston, and Daniel Burrows. *Explanation constraint programming for model-based diagnosis of engineered systems*, Aerospace Conference, 2004. Proceedings. 2004 IEEE, Vol.5, Iss., 6-13 March 2004 Pages: 3495- 3501 Vol.5.

[Narasimhan, *et al*., 2004] Sriram Narasimhan, Richard Dearden, and Emmanuel Benazera. *Combining Particle Filters and Consistency-based Approaches for Monitoring and Diagnosis of Stochastic Hybrid Systems*, 15th International Workshop on Principles of Diagnosis (DX04), Carcassonne, France, June 2004.

[Ledeczi, *et al*., 2001] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P.: *The Generic Modeling Environment*, Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.

[Williams and Ragno, 2003] Brian C. Williams, and Robert Ragno. *Conflict-directed A* and its Role in Model-based Embedded Systems*, Special Issue on Theory and Applications of Satisfiability Testing, Journal of Discrete Applied Math, January 2003.

[Balaban, *et al.*, 2007] Edward Balaban, Howard Cannon, Sriram Narasimhan, and Lee Brownston. *Model-Based Fault Detection and Diagnosis System for NASA Mars Subsurface Drill Prototype*, IEEE Aerospace Conference, Big Sky, Montana, March 3-10, 2007.

[Scott Poll, *et. al.,* 2007] Scott Poll, et. al. *Evaluation, Selection, and Application of Model-Based Diagnostic Tools and Approaches*, AIAA Infotech@Aerospace 2007, Rohnert Park, CA, May 7-10.