State Tracking of Uncertain Hybrid Concurrent Systems[1]

Emmanuel Benazera[2] and **Louise Travé-Massuyès**[3] and **Philippe Dague**[4]

Abstract. In this paper we propose a component-based hybrid formalism, that represents physical phenomena by combining concurrent automata with continuous uncertain dynamic models. The formalism eases the modeling of complex physical systems, and adds concurrency to the supervision of hybrid systems. Uncertainties in the model are integrated as probabilities at the discrete level and intervals at the continuous level. Our modeling framework is rather generic while focusing on the construction of intelligent autonomous supervisors by integrating a continuous/discrete interface able to reason on-line in any region of the physical system state-space, for behavior simulation, diagnosis and system tracking.

1 INTRODUCTION

In the past few years, numerous works have been presented to model embedded systems with hybrid models and reason about them for simulation, diagnosis [9] or verification [1] purposes. The modeling framework usually expresses the different operating modes of the system as a set of finite automata and associates to each mode continuous knowledge encoded through standard numeric differential equations. In this paper we propose a component-based hybrid formalism, that represents physical phenomena by combining concurrent automata with continuous uncertain dynamic models. However it is not sufficient to add continuous knowledge to automata, because moving between operating modes requires the automatic construction of the structure of the newly assembled continuous model. It means computing both the characterization of the region of the state-space of the operating mode (denoted as a *configuration*), and a proper causal ordering between the active variables in that mode. No pre-study of the behavior of the physical system is required to determine the state-space regions associated with the current system configuration(s) because the search at continuous level is casted into a boolean constraint satisfaction problem. A reasoning continuous/discrete interface (C/D I) is thus added, which provides an online generation of the characterization of the new model structure by making use of enhanced Truth Maintenance techniques [18] on the logical model. This is keypoint to achieve the diagnosis of the hybrid system for which detection is provided by the continuous layer and state identification is performed at the discrete logical level by searching for the current configuration consistent with observations. At the same time, the logical framework allows the description of purely discrete component behavior in the same manner as in [17]. Section 2 describes the discrete and the continuous layers; Section 3 presents the interface that integrates both layers together; Section 4 presents the algorithms required to reason about hybrid models and to track multiple trajectories in both simulation and diagnosis; Section 5 discusses our research, compares and references some related work.

[1] This work is supported by CNES (French Space Research Center) and ASTRIUM.
[2] Laboratory for Analysis and Architecture of Systems, Toulouse, France
[3] Laboratory for Analysis and Architecture of Systems, Toulouse, France
[4] LIPN - UMR 7030 Université Paris 13, France

2 Hybrid System Formulation

2.1 Hybrid Systems as Transition Systems

The set of all components of the physical system to be modeled is denoted by $Comps$. Every component in that set is described by a hybrid transition system. The set of all variables used to describe a component is denoted V and is partitioned in the following manner:

- $\Pi = \Pi_M \cup \Pi_C \cup \Pi_{Cond} \cup \Pi_D$ — set of discrete variables of 4 distinct types (Mode, Command, Conditional, Dependent),
- $\Xi = \Xi_I \cup \Xi_D$ — set of continuous variables of 2 distinct types (Input, Dependent).

Mode variables Π_M represent components nominal or faulty modes, such as *on* or *stuck*. Command variables Π_C are endogeneous and exogeneous commands modeled as discrete events to the system (e.g. software commands). Continuous input variables Ξ_I are exogeneous continuous signals to the system determined by its environment (e.g. known inputs or disturbances). Conditional variables Π_{Cond} are specific discrete variables that represent conditions on continuous variables. Discrete and continuous dependent variables are all other variables. Finally the set Obs contains observable variables of the physical system. Each observable signal has an explicit sampling period. Our hybrid transition system is an extension of the standard transition system [8] that adds (qualitative or quantitative) constraints to the states.

Definition 1 (Hybrid Transition System – HTS) *A Hybrid Transition System HTS is a tuple (V, Σ, T, C, Θ) with:*

- $V = \Pi \cup \Xi$ *— set of all variables.* $\forall v \in V$*, the domain of* v *is* $D[v]$*, finite for variables in* Π*, intervals or real values in* $\Re$ *otherwise.*
- Σ *— set of all interpretations over* V*.*
  Each state in Σ *assigns a value from its domain to any variable* $v \in V$*.*
- T *— finite set of transition variables.*
  Each variable τ_m *in* T *ranges over its domain* $D[\tau_m]$ *of possible transitions of the mode variable* $m \in \Pi_M$*. Each* τ_m^i *in* $D[\tau_m]$ *is a function* $\tau_m^i : \Sigma \rightarrow 2^\Sigma$*, associated to a mapping function* $l_{\tau_m^i}$ *.*
- C *— set of (qualitative or quantitative) continuous constraints over* V*.*
  Each constraint c *in* C *at least depends on one mode variable in* Π_M*.* $\forall m \in \Pi_M$*, we note* $C[m]$ *the set of constraints associated to the variable* m*.*

- Θ — *set of initial conditions.*
  Θ is a set of assertions over V such that they define the set of initial possible states, i.e. the set of states s in Σ such that $s \models \Theta$.

Note that in a HTS, due to the continuous constraints in C, some transitions can trigger according to conditions over continuous variables. At the discrete/continuous interface level, these conditions have a corresponding discrete variable in Π_{Cond}, which captures their truth value. Throughout this paper we illustrate the formalism and later on the diagnosis operation on a simple example: figure 1 shows the HTS of a room R submitted to a temperature source. It has two nominal modes: *open* (a door or a window is opened), *closed*, and a faulty *unknown* mode. The room temperature x is influenced by the temperature of the source x_e according to a first-order differential equation which accounts for the room characteristics Q_c (closed) and Q_o (open). The actions that move the room from one mode to another are modeled as observed single discrete commands $cmd = open$ and $cmd = close$. Figure 2 presents the model of a thermostat T, with faulty modes *stuck_on*, *stuck_off* and *unknown*, as well as required transitions. This thermostat switches according to the room temperature x (it should be in its *on* mode when the temperature $x \leq m$ to warm up the room, and back to its *off* mode when $x \geq M$ to cool it down). x is hence influenced by the heater setting temperature h (in mode *on*) or by the outside temperature x_{ext} (in mode *off*). The temperature variation $\dot{x}$ is observed through a sensor with additive noise $\dot{x}_{noi}$. Initially, $x = x_{ext}$, the room is *closed* and the thermostat is *on*. Variables of both HTS are:

$$
\begin{array}{rcl}
R.mode \in \Pi_M & = & (closed, open, unknown) \\
R.cmd \in \Pi_C & = & (none, open, close) \\
R.c \in \Pi_{cond} & = & (R.x \leq m, R.x > m \wedge R.x < M, R.x \geq M) \\
R.x \in \Xi_D & \in & [-\infty, +\infty] \\
R.\dot{x} \in \Xi_D & \in & [-\infty, +\infty] \\
R.\Delta x \in \Xi_D & \in & [-\infty, +\infty] \\
R.\dot{x}_{noi} \in \Xi_I & \in & [-1, 1] \\
R.Qc & \in & [0.05, 0.15] \\
R.Qo & \in & [0.02, 0.05] \\
R.a & \in & [0.9, 1.1] \\
R.x_{ext} & = & 4 \\
T.mode \in \Pi_M & = & (off, on, stuck\_on, stuck\_off, unknown) \\
T.M & = & 17 \\
T.m & = & 10 \\
T.h & = & 20 \\
Obs & = & \{\dot{x}\}
\end{array}
$$

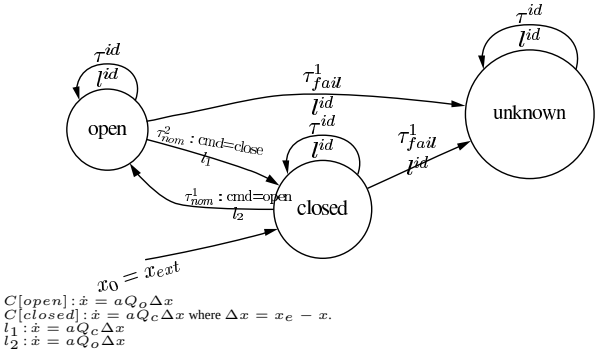


Figure 1. room HTS with unknown mode

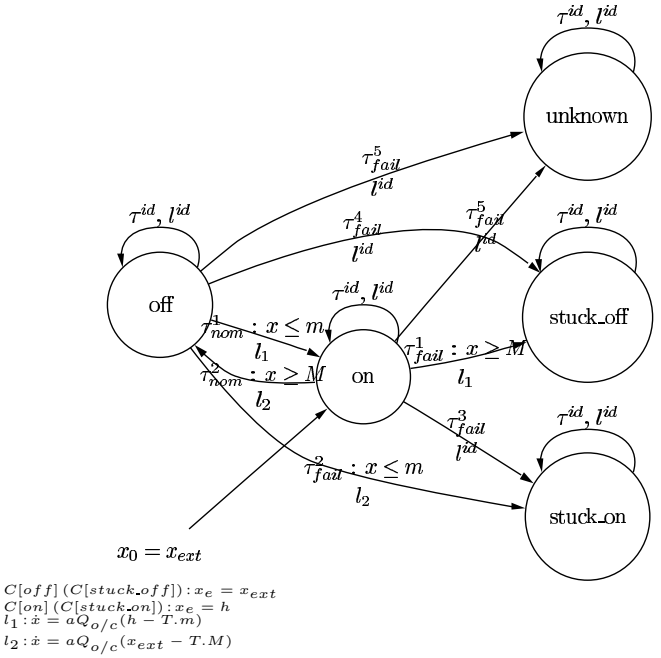


Figure 2. thermostat HTS with fault modes

2.1.1 States and Time

Considerations about time are central because both the discrete and the continuous frameworks use time representations that are different. At the continuous level, time is explicit in the equations that represent the physical system behavior, we call it *physical time* θ. Physical time is discretized according to the highest frequency sensor, providing the HTS reference sampling period T_s. $x(kT_s)$, or $x(k)$ for short, specifies the value of the continuous vector of state-variables in Ξ at physical time kT_s. We call *abstract time* the time at the discrete level. It is dated according to the occurrence of discrete events. At date t, the discrete state π_t of a HTS is the tuple (M_t, Q_t), where M_t is the vector of instances of mode variables, and Q_t the vector of instances of variables of Π in qualitative constraints. Discrete state-variables are in $\Pi \setminus \Pi_{Cond}$. Abstract time dates are indexed on physical time, which informs about how long a component has been in a given discrete state. If $t = kT_s$, then we write the indexed date t^k. When there is no ambiguity it is simply denoted by t. The *hybrid state* s_{tk} of a HTS is the tuple $(\pi_{tk}, x(k))$.

2.1.2 Transitions

Transitions describe changes between modes over time. The transition variable associated to a mode variable m is denoted τ_m such that its domain is $D[\tau_m] = \{\tau_m^i \in T_N\} \cup \{\tau_m^j \in T_F\} \cup \{\tau^{id}\}$, with:

- T_N the set of *nominal* transitions that express switches from one nominal mode to another,
- T_F the set of *faulty* transitions that move the HTS into a faulty mode,
- τ^{id} the *identity* transition that allows a HTS to stay in its current mode.

Because transitions cannot always be considered as instantaneous against the frequency of the sensors, we introduce delays on nominal transitions. Delay $d_{\tau_m^i}$ is such that once a transition τ_m^i is *enabled* it is triggered after $d_{\tau_m^i} T_s$, i.e. after $d_{\tau_m^i}$ physical time units.

While a transition is *enabled* and waiting for its delay to expire, it is said to be in *standby*. For a matter of simplification, the delay will be referred as d when there is no ambiguity. A delay on transition can also be modeled by adding modes and clocks to the hybrid transition system [4]. We do not use this representation here because we think that it does not enforce the easy representation of a component as a transition system by creating modes that are irrelevant for the diagnosis purpose. To model faults, we define fault modes of which we know the behavior, such as *stuck_on* or *stuck_off*, and a unique mode *unknown* that is rather specific because it has no constraints and covers all interpretations in Σ. Modeled faults are often abrupt faults in the sense that they do not represent tenuous parameter changes. Thus fault transitions have no delay, i.e. their duration is one physical time unit.

Definition 2 (pre and post assertions) *For a given transition τ_m^i and a given state $s_{t^k} \in \Sigma$, we note assertions $pre(\tau_m^i) = m^j \wedge \phi^i_{\Pi_{C\cup Cond}}$ and $post(\tau_m^i) = m^{j'}$ where:*

- *m^j and $m^{j'}$ are two instances of the mode variable m,*
- *$\phi^i_{\Pi_{C\cup Cond}}$ is a logical condition over instances of variables of both Π_C and Π_{Cond}.*

We refer to the *guard* of a transition as the condition statement $\phi^i_{\Pi_{C\cup Cond}}$ that triggers the transition. Only fault transitions can be spontaneous, so their guard can be always true. Traditionnally, probabilities are also attached to every nominal and faulty transitions. In our example, T is represented as follows ($\bigcirc$ is the next operator from temporal logic):

$$\begin{aligned}
R.\tau^1_{nom} &: R.mode = closed \wedge R.cmd = open \quad \bigcirc \quad R.mode = open \\
R.\tau^2_{nom} &: R.mode = open \wedge R.cmd = close \quad \bigcirc \quad R.mode = closed \\
R.\tau^1_{fail} &: R.mode = open \vee R.mode = closed \quad \bigcirc \quad R.mode = unknown
\end{aligned}$$

$$\begin{aligned}
T.\tau^1_{nom} &: T.mode = off \wedge R.x \leq m \quad \bigcirc \quad T.mode = on \\
T.\tau^2_{nom} &: T.mode = on \wedge R.x \geq M \quad \bigcirc \quad T.mode = off \\
T.\tau^1_{fail} &: T.mode = on \wedge R.x \geq M \quad \bigcirc \quad T.mode = stuck_off \\
T.\tau^2_{fail} &: T.mode = off \wedge R.x \leq m \quad \bigcirc \quad T.mode = stuck_on \\
T.\tau^3_{fail} &: T.mode = on \quad \bigcirc \quad T.mode = stuck_on \\
T.\tau^4_{fail} &: T.mode = off \quad \bigcirc \quad T.mode = stuck_off \\
T.\tau^5_{fail} &: T.mode = on \vee T.mode = off \quad \bigcirc \quad T.mode = unknown
\end{aligned}$$

There is no delay when the thermostat (room) switches between *on* (*open*) and *off* (*closed*) modes.

2.2 Moving between modes

When a transition triggers, the component switches from one mode to another, the corresponding HTS needs to transfer its continuous state vector x as well. For that reason each transition τ_m^i is associated with a *mapping function* $l_{\tau_m^i} : \Sigma \rightarrow \Sigma$ over the dependent variables in V. It initializes the value of a subset of variables in the hybrid state resulting from applying τ_m^i to $s_{t_l^k}$ where l is the abstract time index. Other variables in $s_{t_l^k}$ keep their previous value. The identity mapping function is denoted l^{id}. Triggering a transition is a two steps operation [1]. First, mode change is performed by applying the transition τ_m^i to the current hybrid state and moving to the resulting mode after its delay has expired (*transition relation* $\xrightarrow{\tau_m^i}$):

$$\frac{\tau_m^i \in T, (s_{t_l^k}, s_{t_{l+1}^{k+d}}) \in \Sigma^2, s_{t_l^k} \models pre(\tau_m^i)}{s_{t_l^k} \xrightarrow{\tau_m^i} s_{t_{l+1}^{k+d}}} \quad (1)$$

Second, initialization is performed by making use of the mapping function, and physical time goes on (*time-step relation* $\xrightarrow{\theta}$):

$$\frac{(\pi_{t_{l+1}}, x(k+d)) = l_{\tau_m^i}(s_{t_l^k})}{(\pi_{t_{l+1}}, x(k+d)) \xrightarrow{\theta} (\pi_{t_{l+1}}, x(\theta))} \quad (2)$$

where $x(\theta)$ is the continuous state associated to the discrete state $\pi_{t_{l+1}}$ over the continuous time θ. In the systems we are interested in, most of the discontinuities are driven by controller actions and preserve the state variables continuity. In our example, the temperature is obviously continuous when the thermostat switches from *on* to *off* and we use the temperature $T.M$ at this point to compute $\dot{x} = aQ_c(x_e - T.M)$. However it has been shown in [10] that in specific cases, retrieving a mapping function from the models of both considered modes is far from trivial and requires deep understanding of the physics of the phenomena abstracted in the discontinuity.

2.3 Component modes behavior

We described how transitions express component's dynamics between modes. At this point we want to represent each intra-mode behavior with two goals in mind: on the one hand the representation must encode the available qualitative or quantitative knowledge; on the other hand it must be suitable for efficient reasoning. For purely discrete components, usually software drivers as well as complex electronic devices, the behavioral model is given by a set of boolean constraints over $\Pi_C \cup \Pi_D$ that are associated to each mode variable value in the same manner as in [17]. For continuous components, the continuous behavior is expressed by discrete-time continuous constraints over Ξ. Each constraint is attached to a mode of the transition system. The discrete-time continuous constraints are of the following standard form:

$$\left\{ \begin{array}{lcl} x(k+1) & = & Ax(k) + \sum_{j=0,\ldots,r} B_j u(k-j) \\ y(k+1) & = & Cx(k+1) \end{array} \right. \quad (3)$$

where $x(k)$, $y(k)$, and $u(k)$ represent the continuous state vector of dimension n, ouput (observed) variables vector of dimension p and input (control) variables of dimension q at time kT_s, respectively; A, B_j and C are matrices of appropriate dimensions. To provide a suitable framework for reasoning, continuous constraints are encoded in a specific two levels formalism [15] which includes a causal model and an analytical constraint level. The causal model is obtained from equation (3) by expressing it as a set of *causal influences* among the (state, input or output) variables. Influences may be of different types: dynamic, integral, static and constant. The following definition expresses first and second order dynamic influences:

Definition 3 (Dynamic influence) *A dynamic influence i_{ij} is a tuple $(\xi_i, \xi_j, K, T_d, T_r, cond)$ for first order differential relations and $(\xi_i, \xi_j, K, T_d, \zeta, w, cond)$ for second order relations with :*

- *$\xi_i \in \Xi$ and $\xi_j \in \Xi$ are two continuous variables such that ξ_i influences ξ_j,*
- *K is the parameter* gain*, representing the static gain of the influence,*
- *T_d is the parameter* delay*, representing the time needed by ξ_j to react to ξ_i,*
- *T_r is the parameter* response time *representing the time needed by ξ_j to get to a new equilibrium state after having been perturbed,*
- *ζ is the* damping ratio *of the system,*
- *w is the* undamped natural frequency *of the system,*

- *cond is the parameter* condition *which specifies the logical condition under which the influence is active.* $cond$ *ranges over elements of* V.

The underlying operational model of dynamic influences is provided by the following equation:

$$\xi_j(k+1) = \sum_{p=0,\ldots,n-1} a_p \xi_j(k-p) + \sum_{q=0,\ldots,m} b_q \xi_i(k+1-q) \quad (4)$$

where ξ_i and ξ_j are continuous variables, n is the influence order and $m \leq n$ (causal link). Usually an equation is modeled by a set of influences. When necessary, uncertainties can be taken into account in the influence parameters and as additive disturbances. The first are represented by considering that parameters a_p and b_q have time independent bounded values, i.e. they are given an interval value. The latter can be introduced as a bounded value constant influence acting on ξ_j. From the superposition theorem that applies to the linear case, the computation of the updated value of variable $\xi_j \in \Xi$ in an equation eq consists in processing the sum of the activated influences from eq having exerted on ξ_j during the last time-interval. The prediction update of all the state and observed variables $x(k)$ and $y(k)$ from the knowledge of control variables $u(k)$ and influence activation conditions is performed along the causal model structure. Our representation of uncertainties leads to the prediction of continuous variable trajectories in the form of bounded envelopes. In other words, the system state $x(k)$ at every time instant $t = kT_s$ is provided in the form of a rectangle of dimension n.

Definition 4 (Causal system description – CD) *The causal system description associated to the set of continuous constraints of a* HTS *is a directed graph* $G = (\Xi, I)$ *where* I *is a set of edges supporting the influences among variables in* Ξ*, with their associated conditions and delays.*

The numerical intervals obtained from equation (4) are refined at the analytical model level with global constraints by performing a tolerance propagation algorithm [6] on the set of variables. Back to the example, the feasible continuous states of Σ are specified by the influences in each HTS:

$$R.i_1\ (static) : \text{if } (R.mode = closed) \text{ then } R.\Delta x \overset{gain=Q_c}{\longrightarrow} R.\dot{x}$$
$$R.i_2\ (static) : \text{if } (R.mode = open) \text{ then } R.\Delta x \overset{gain=Q_o}{\longrightarrow} R.\dot{x}$$
$$R.i_3\ (integral) : R.\dot{x} \overset{gain=a}{\longrightarrow} R.x$$
$$R.i_4\ (static) : R.x \overset{gain=-1,delay=1}{\longrightarrow} R.\Delta x$$
$$T.i_1\ (constant) : \text{if } (T.mode = on \vee T.mode = stuck\_on) \text{ then } T.h \longrightarrow R.\Delta x$$
$$T.i_2\ (constant) : \text{if } (T.mode = off \vee T.mode = stuck\_off) \text{ then } R.x_{ext} \longrightarrow R.\Delta x$$
$$T.i_3\ (constant) : T.\dot{x}_{noi} \longrightarrow R.\dot{x}$$

Influences without explicit conditions are valid in all modes except in the *unknown* mode. Figure 3 presents the nominal CD for the room and the thermostat.

2.4 Hybrid Component System

Once components have been modeled as HTS, constituting a generic reusable database of models, they need to be assembled in a *Hybrid Component System* to model the entire physical plant. Components are hence instantiated. Within the whole plant model, components are concurrent, i.e. able to evolve independently which allows us to reason on subparts of the model.

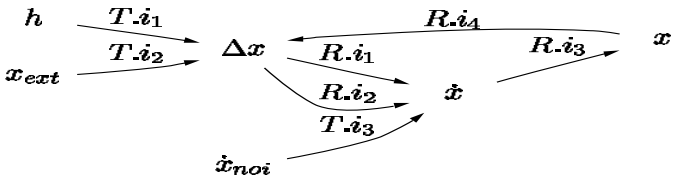


Figure 3. Causal nominal system description of the thermostat and room example

Definition 5 (Hybrid Component System – HCS) *A Hybrid Component System* HCS *is a tuple* $(Comps, V, \Sigma, T, C, \Theta)$ *with* $Comps$ *being a set of* n *components modeled as concurrent hybrid transition systems* $H_i = (V_i, \Sigma_i, T_i, C_i, \Theta_i)$, $\left(\bigcup_{i=1,\cdots,n} V_i\right) = V$, $\Sigma \subseteq \bigotimes_i \Sigma_i$, $T = \bigcup_i T_i$, $C = \bigcup_i C_i$, $\Theta = \bigcup_i \Theta_i$.

We track the evolution of a HCS over a temporal window in the form of a trajectory as a succession of states. At each time-step, constraints and commands first synchronize on shared variables in Π_D, Π_C and Ξ (the room and the thermostat share Δx). Shared variables serve as time-dated communication channels between automata. The automata must nevertheless synchronize between states. The synchronization uses transitions and is such that given components of the HCS:

- HTS that received a command synchronize on the corresponding nominal transition,
- non commanded HTS synchronize on the identity transition τ^{id}.

When synchronized, HTS instances are introduced into the trajectory whereas other HTS are not copied at each time-step. Intuitively we want to only introduce the minimal subset of the HTS necessary for tracking and diagnosis purposes. In [11] and for discrete-only models, this subset is computed using a pre-compilation of prime implicants of mode variables. In our implementation, transitions synchronize a posteriori, and only when needed by the reasoner to operate. This saves big amounts of memory as when tracking a physical system in its nominal long-term state, very few components need to be reintroduced.

The concurrency process is complexified by the introduction of delays on transitions. Figure 4 presents an example of the synchronization of four concurrent HTS, H_1 to H_4. Four transitions are enabled on shared variables at time-step t_l^k and synchronize over the three next time-steps with different delays, except for d_{τ_2} and d_{τ_4} that are equal. H_1 and H_2, as well as H_3 and H_2 have constraints that share variables. Due to different commands, the concurrence makes the four HTS change mode at time t_l^k whereas other HTS in the model stay inactive (they are not represented on the figure). Then the synchronization effort takes into account delays of triggered transitions as well as the links between HTS through shared variables:

- H_2 and H_4 have the same delay and thus participate a same hybrid state at time-step $t_{l+1}^{k+d_{\tau_2}}$,
- H_1 and H_2 synchronize at $t_{l+2}^{k+d_{\tau_1}}$. This is done with the identity transition on H_2.
- H_1 (or H_2) and H_4 don't synchronize at $t_{l+2}^{k+d_{\tau_1}}$ because they don't share any variables,
- H_1 and H_2 share variables but *don't synchronize* at $t_{l+1}^{k+d_{\tau_2}}$ because τ_1 is already in *standby*.

The last remark is of importance because it relies on the hypothesis that *we cannot track or diagnose a physical component while it is switching from one mode to another*, i.e. when one of its transitions is in *standby*, as the required transient models are often unknown or too complex. The consequence is that components only synchronize in their non-standby states.

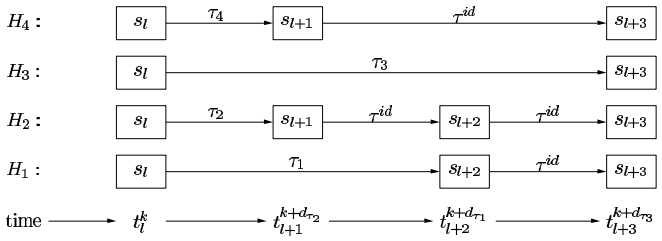


Figure 4. synchronization over 3 states of four HTS.

3 Continuous/Discrete Interface

3.1 Configurations

Depending on the mode at a given time, a HCS has its hybrid state that ranges over several continuous regions. These regions are known to be difficult to determine and compute, if not undecidable. We propose an on-line mechanism to keep track of the state-space partition by sheltering every continuous functional piece with a conjunction of logical conditions we denote as a *configuration.*

Definition 6 (HCS configuration) *A configuration for a HCS at time-step t^k is a logical conjunction $\delta_{t^k} = (\bigwedge_i m^i) \wedge (\bigwedge_j \Pi^j_{Cond})$ where the m^i are instanciations of component modes in Π_M and the Π^j_{Cond} are variables of Π_{Cond}.*

The configurations are automatically drawn from conditions on both transition guards and influences that define structural changes in the model. A configuration can be attached to one or more modes in Π_M. In our example, the continuous state is easily partitioned by the thermostat's transitions into three regions determined by the three conditions on variable x, defining 27 configurations:

$$
\begin{array}{lcl}
\mathcal{C}_1 & : & R.mode = closed \wedge T.mode = on \wedge R.x \leq m \\
\mathcal{C}_2 & : & R.mode = closed \wedge T.mode = on \wedge (R.x > m \wedge R.x < M) \\
\mathcal{C}_3 & : & R.mode = closed \wedge T.mode = off \wedge (R.x > m \wedge R.x < M) \\
\mathcal{C}_4 & : & R.mode = closed \wedge T.mode = off \wedge R.x \geq M \\
\ldots &&
\end{array}
$$

Whatever the complexity of the conditions defining the regions of the physical system, it is easy to logically express any condition as a boolean variable of Π_{Cond}, whose 1/0 corresponds to the condition and its negation. This however leads to a number of partitions that is not optimal relatively to the exact number of state-space regions in which the physical system evolves. Note that the configuration associated to the *unknown* mode encompasses the overall state-space.

3.2 Causal ordering for static equations

When switching from one mode to another, some equations and variables are added or retracted according to the new configuration. Consequently, due to the possible presence of static continuous equations in the model, a proper causal ordering of variables is to be found when entering the new mode. A brute force approach would consist in generating a new causal structure for every different mode. The problem of performing an on-line incremental generation of the causal structure has been previously addressed [16] but it is solved here in a slightly different manner. This is done by first casting the problem into a boolean constraint satisfaction problem: every continuous equation and variable in the HCS is associated to boolean variables in Π whose truth values state if the variables or equations are active or not. Rules over the boolean variables are automatically built to represent the conditions of these activations and form a logical representation of the causal-ordering problem.

3.3 Overview

The previous configuration and causal ordering problems are solved on-line by using a truth maintenance system (TMS) to reason on the corresponding boolean constraint satisfaction problems. We use the context switching algorithms of [18] because we are not interested in generating all configurations of the physical system but to switch from one to another as fast as possible. The HCS reacts to events,

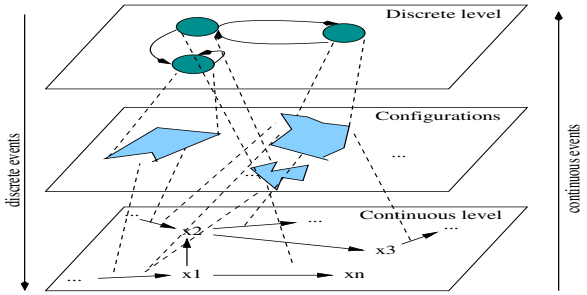


Figure 5. 3-layers interactions

i.e. observations from sensors as well as commands, and propagates them to the model's discrete and continuous levels through the logical interface and the way back. Figure 5 sums up these interactions. The C/D I, made of the variables in Π_{Cond} associated to influence conditions and transition guards, as well as the causal ordering logical model, ensures the logical consistency of the changes triggered by the flow of events.

4 Simulation and Diagnosis of a Hybrid Component System

4.1 Simulation

A HCS simulation is a run of concurrent hybrid transition systems that generates possible nominal trajectories of the HCS according to issued commands and inputs over the time. The uncertainty on the continuous constraint parameters determines the precision of the computed envelopes that enclose the observed behavior of the physical system at each time step.

Sometimes the truth value of a condition in a configuration may be undetermined when checked against a rectangular enclosing of the continuous state-variables. The problem arises from the fact that some variables over which configurations rely are not measured. When the computed bounds of such a continuous variable ξ_i span over more than one configuration region relying on that variable, we

say that the current *configuration is splitting the continuous state on variable* ξ_i. Figure 6 shows a configuration split for the thermostat

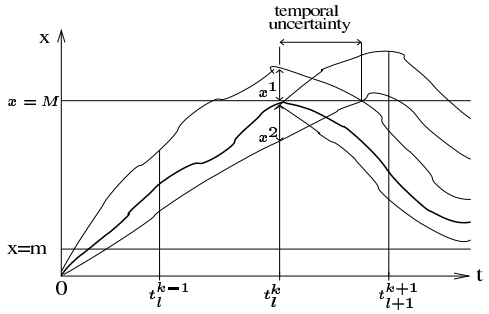


Figure 6. Transition guard split

example when crossing $x = M$. The current configuration splits on regions x^1 and x^2 and the two possible trajectories are tracked simultaneously. In applications, this situation happens rather frequently and multiple consecutive splits of a guard on the same variable can occur because sensor frequencies are usually beneath the *temporal uncertainty* induced by the envelopes. We first want to split the continuous state into logical branches then refine consequently the bounds on all continuous variables in every explored branch. For a given continuous variable ξ_i, the logical split of a configuration δ_{t^k} returns the set of possible configurations to be tracked:

$$[\delta_{t^k}](\xi_i) = \bigvee_j \left(\Pi^j_{Cond_{\xi_i}} \wedge \left(\bigwedge_n \Pi^n_{Cond} \right) \right) \tag{5}$$

where $\Pi^j_{Cond_{\xi_i}}$ are variables of Π_{Cond} relying on ξ_i and Π^n_{Cond} other conditions in δ_{t^k}. Relation (5) is used to compute the splitted areas because it is much faster than exploring the overall continuous state space. The following algorithm is applied on every tracked trajectory:

1. The configuration δ_{t^k} is checked against the rectangular region defined by variables' predicted envelopes to find a variable ξ_i over which it is splitting the state-space,
2. The state-space is logically splitted with relation (5). For each configuration $\delta^j_{t^k}$ in $[\delta_{t^k}](\xi_i)$, its corresponding continuous region is denoted $x^j_{\xi_i}(k)$ and its corresponding discrete state $\pi^j_{t^k,\xi_i}$.
3. Envelopes over variables in Ξ are refined in every region $x^j_{\xi_i}(k)$ by filtering them on the constraints defined by the conditions in the configuration [6].
4. $(\pi^j_{t^k,\xi_i}, x^j_{\xi_i}(k))$ constitute new hybrid states enclosed in new trajectories to be tracked.

The three preceding steps are applied for remaining variables on the growing set of generated trajectories. Finally the resulting set of computed hybrid states is:

$$[s_{t^k}] = \bigotimes_{i,j} (\pi^j_{t^k,\xi_i}, x^j_{\xi_i}(k)) \tag{6}$$

In our example, the thermostat's configurations only split on the temperature x. On figure 6, until time-step t^k_l, the configuration of the HCS is

$$\mathcal{C}_2 : R.mode = closed \wedge T.mode = on \wedge R.x > m \wedge R.x < M$$

At time-step t^k_l, due to the crossing of $x = M$, the current configuration is splitted on x. A new partial hybrid state comes from equation (5):

$$R.mode = closed \wedge T.mode = on \wedge R.x \geq M$$

Then bounds of variable x are refined in each configuration by filtering the values with respective constraints $R.x > m \wedge R.x < M$ and $R.x \geq M$. As transition $T.\tau^2_{nom}$ turns *enabled* with the second configuration, the configuration is instantaneously ($T.\tau^2_{nom}$ has no delay) updated to:

$$\mathcal{C}_4 : R.mode = closed \wedge T.mode = off \wedge R.x \geq M \tag{7}$$

From that point the system tracks two distinct trajectories.

4.2 Fault Detection

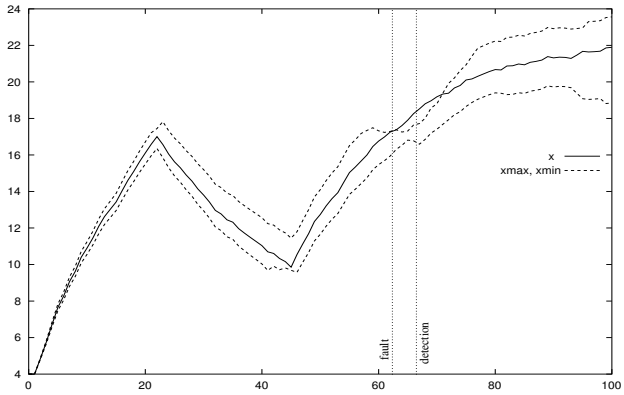
The detection algorithm then uses the above prediction of the endogeneous continuous variable values to obtain robust decisions about the existence of faults, based on *adaptive thresholds* provided by the envelopes' upper and lower bounds. This is performed by comparing the predicted and observed values of variables across time. The adaptive thresholds principle fairly reduces the possibility of false alarms when tracking the system. However, to achieve better robustness, we usually mark a variable as mibehaving after it has been outside of its bounds for at least n_{misb} physical time-steps. After that delay, the diagnosis operation is triggered.

For dynamic influences, the algorithm sensitivity relies on a mixed strategy which combines an observer type strategy (*closed-loop* mode, i.e. the measure of a variable y at time t is used to elaborate the prediction of y at time $t+1$) with a pure simulation strategy (*open-loop* mode, i.e. the prediction of y at time $t+1$ is obtained from the prediction of y at time t) to determine the thresholds and further assess variable states. We call this strategy a *semi-closed loop* (SCL) strategy [13]. The mode control (open-loop or closed-loop) depends on whether the observed value of a variable y is in the predicted envelope (normal situation) or out of it (alarming situation). As soon as the variable becomes alarming, running on a closed-loop mode might drive the prediction to follow the fault, turning the detection procedure insensitive to the fault. The prediction temporal window is hence scaled up by switching to the open-loop mode. Note that the fault detection mechanism is very efficient at ruling out wrong trajectories issued from multiple successive splits on the same boundary constraint.

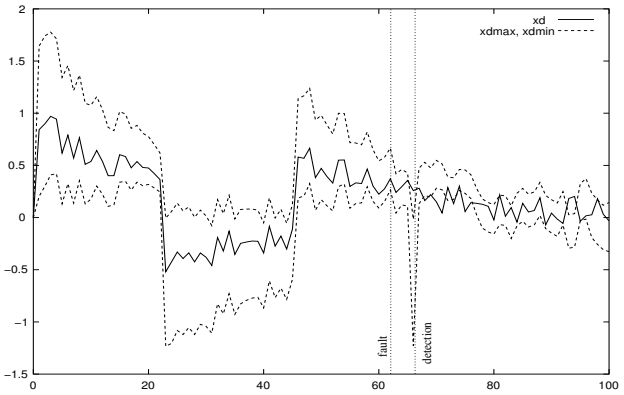
Figure 7 shows three scenarios with faults where detection is applied. On the first scenario the thermostat fails to switch at time-step 63 and sticks to its *on* mode. In the second scenario the constant $T.h$ is degraded from time-step 46 to a lower value, so the heater is slower to warm the room. Scenario three presents a fault characterized by an abrupt structural change in the thermostat model. For all scenarios, $n_{misb} = 1$.

4.3 Diagnosis

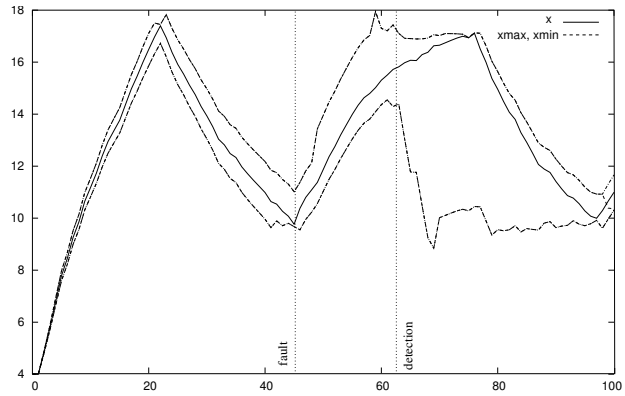
When a fault is detected, a diagnosis comes back to find the current configuration of the HCS according to observations, inputs and commands. This must be performed over a finite temporal window [11], but because of the fault detection at a continuous level the problem of losing solutions is strongly reduced. The temporal window is usually set up to the physical time that corresponds to the longest chain of non-repeated transitions. In our example 20 physical time-steps cover an *on-off* complete sequence.



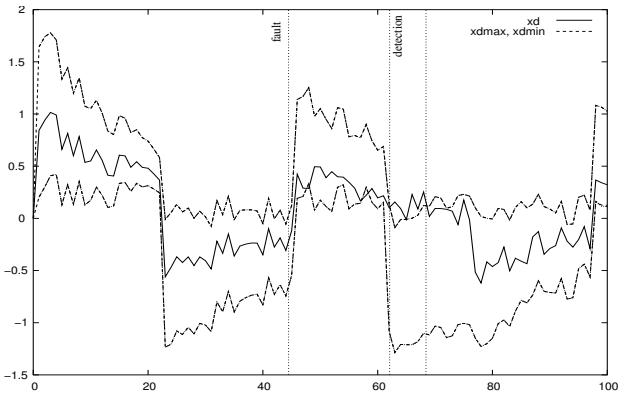
(a) Scenario 1, x: After detection and diagnosis, a few more time-steps are necessary for the prediction to catch up with the physical system. This comes from the fact that the estimation of the time of the fault is not accurate enough: because of the time uncertainty due to the envelopes, the estimation is a few time-steps late.



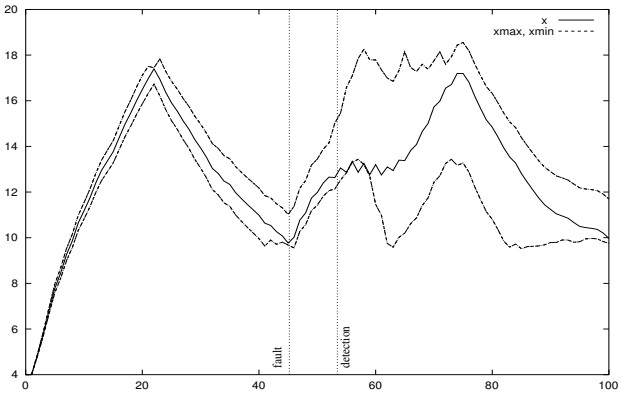
(b) Scenario 1, $\dot{x}$: the fault is detected at time-step 68.



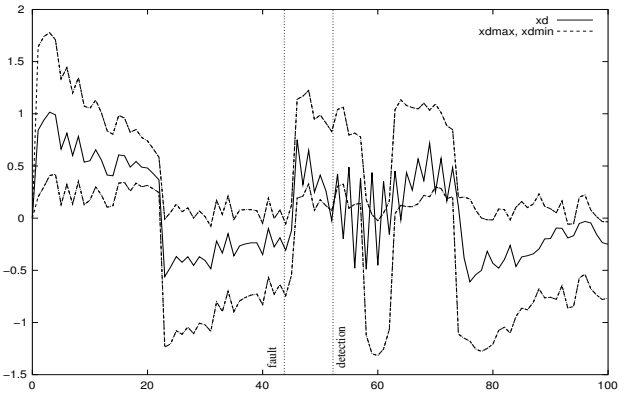
(c) Scenario 2, x: After the fault is diagnosed, the *blind state-tracking method* uses the nominal behavior of the thermostat and predicts all possible switches at each time-step: the very wide envelope shows that it is not sure if the thermostat is *on* or *off*.



(d) Scenario 2, $\dot{x}$: The fault is not so abrupt as to be detected instantaneously. Measures go *in* the predicted bounds again at time-step 69. This is due to the fact that when using the *blind state-tracking method*, the thermostat's controller model is still switching on valid thresholds.



(e) Scenario 3, x: The thermostat switches on valid thresholds and the *blind state-tracking method* keeps a relatively good tracking of the temperature after the fault occured. This is due to the fact that the physical model of the room is still valid.



(f) Scenario 3, $\dot{x}$: After a thermostat's structure change, the heater setting temperature $T.h$ is oscillating. When turned *off*, T keeps its nominal behavior.

Figure 7. Three fault scenarios

Definition 7 (HCS **Diagnosis)** *A diagnosis* $diag(t)$ *over* m *time-steps for a* HCS *is such that* $diag(t) = \{\delta_t\}_{t=1,\cdots,m}$ *with the consitency of:*

$$HCS \cup Obs_{t=1,\ldots,m} \cup \left(\bigcup_{t=1,\cdots,m} \delta_t \right) \quad (8)$$

Solving relation (8) is a three steps operation. First, existing conflicts (a set of influences which cannot be unfaulty altogether) are exhibited from the causal system description (CD) of the HCS, each influence stamped with a temporal label and activation condition. They are then turned into diagnosis candidates by a failure-time oriented enhanced version of the hitting set algorithm [14]. Temporal information is drawn from maximizing on each components the delays of the influences downstream the faulty variables in CD.

Second, at the configurations level, the TMS negates the activation conditions of the conflicting influences and fastly iterates through the logical remaining configurations to reinsure the consistency. Finally, every found configuration is checked against the past observations over the temporal window before being approved as in [11] except that candidate generation and consistency checks are interleaved and run from present time back to the beginning of the temporal window. Configuration solutions to the diagnosis problem contain a mode instanciation of every necessary component in the HTS explaining the observations. Note that on figure 7, for all three scenarios, the diagnosis operation is performed in less than 0.1 seconds on a Pentium II 300 Mhz, which is beneath the measures' frequency, so the detection time-step is equal to the diagnosis time-step.

4.3.1 Diagnosis example with a fault mode

When applied to the first scenario, the diagnosis starts as soon as $\dot{x}$ goes out of its bounds for all currently tracked trajectories: iterating through the system nominal CD from figure 3, at timestep 68 the influences in conflict are $\Gamma = \{T.i_3,\ T.i_2,\ R.i_1,\ R.i_3,\ R.i_4\}$. Relatively to the current configuration (7) it is equivalent to add the constraints $\Gamma_C = \{\bigvee_{m^i=D[T.mode]} T.mode = m^i, R.mode = closed, T.mode = off \vee T.mode = stuck\_off, \bigvee_{m^j \in D[R.mode]} R.mode = m^j\}$ which are activation conditions on the influences in conflict. As $R.i_4$ has a delay of 1, the elements of the last conflict are stamped with the current physical time minus 1. Other conflicts elements are stamped with the current physical time.

The TMS then seeks for consistency on both the configurations and the transition model starting from the current configuration by inserting the negation of the elements in Γ_C: $\Gamma_{\neg C} = \{T.mode = unknown, R.mode = open \vee R.mode = unknown, T.mode = on \vee T.mode = stuck\_on \vee T.mode = unknown, R.mode = unknown\}$ and returns the following possible configurations ranked according to the probabilities attached to transitions and to the number of faults leading to them:

1 : $(R.mode = closed) \wedge (T.mode = stuck\_on) \wedge (R.x \geq M)$
$2a$: $(R.mode = closed) \wedge (T.mode = unknown) \wedge (R.x \geq M)$
$2b$: $(R.mode = unknown) \wedge (T.mode = stuck\_on) \wedge (R.x \geq M)$
3 : $(R.mode = unknown) \wedge (T.mode = unknown) \wedge (R.x \geq M)$

Other configurations with the thermostat in modes *on, stuck_off*, or the room in mode *open* are ruled out during the search process because there are no transitions or past observations and commands consistent with these configurations. Diagnosis 1 fits with the fault in the first scenario (thermostat took transition τ^3_{fail}). The state vector is reinitialized according to the mapping function of τ^3_{fail} (l^{id}) before the tracking continues.

4.3.2 Diagnosis example with the unknown mode

Scenarios 2 and 3 primarily lead to diagnosis $2a$ where the thermostat is in the *unknown* mode. This mode is useful at the discrete level because it assures that there is always a solution to the diagnosis problem[5]. At the continuous level however, it has no model, so it is not possible to track a HTS in that mode. Isolating the *unknown* automata so as to continue the prediction of the behavior of others HTS in the model often leads to tracking based on a wrong model: in scenario 2, once the mode of T has been diagnosed to be *unknown*, influences referring to T are inactive which is equivalent to predict R's behavior with $T.h = 0$. Our current solution to that problem is to use a dedicated *blind state-tracking method* that is applicable thanks to the semi-closed loop fault detection strategy described in subsection 4.2. When a component is found to be in its *unknown* mode, the nominal model of the component is used instead. The detection module runs on open-loop prediction mode until the measures fall into the envelopes again. This is guaranteed to occur because the open-loop predicted envelopes widen with time (uncertainty propagation of interval models). Triggered by this event, the detection module then switches to closed-loop prediction mode and is able to track the system until the measures get out of their bounds again, and so on. This is the method applied on scenarios 2 and 3 on figure 7. However in scenario 2, an improved solution could be to use parameter estimation techniques as proposed in [9] because the structure of the model is still valid. But drawbacks are the additional computational cost and the fact this would leave the system untracked for a period of time (proper parameter estimation requires to wait for properly excited data). More research is needed to integrate existing parameter estimation and model fitting techniques into our framework. Also note that such faults generally result from the natural degradation of the monitored physical system and could be taken into account in causal models [12].

5 Summary, Discussion and Related Work

In this paper we extend previous work on diagnosis in the AI community by presenting a formalism that merges concurrent automata with continuous dynamic system models and reasons about its configurations using logical tools. The problem of reasoning about and diagnosing complex physical plants without computing their continuous reachable state-space is addressed. The approach integrates numerous techniques from different fields into a runnable standalone application, which is able to deal with real-world problems such as satellite state-tracking [3]. The modeling, simulation and diagnosis tools are implemented, including the engine that splits the configurations. The program generates a C++ runtime that is intended to be demonstrated on an autonomous spacecraft test bench at CNES.

Other formalisms for building comprehensive and tracktable hybrid systems include [10] and [4]. But none of these approaches provide an intuitive component-based framework allowing engineers to build reusable models of equipments. Moreover the models often include numerous functional modes that are irrelevant to the diagnosis task. For instance [4] introduces additional modes to deal with delayed transitions, and [10] rather focuses on the expression of the approximations able to produce sound hybrid models of complex physical systems. Besides, it examines types of discontinuities that are rarely encountered in controlled systems. In such systems, most

[5] Note that the *unknown* mode is also a dead-end since no nominal transition can lead out of this mode.

of the discontinuities are driven by controller actions and preserve state variables continuity.

Our work takes numerous ideas from the discrete-only work at the basis of Livingstone [17, 11] and adds and links continuous knowledge to it. The difficult problem of the temporal window that required aggregating in a history all past states in every tracked trajectory is now strongly reduced as it is less likely that a wrong trajectory is tracked without detecting anomalies at the continuous level. [9] introduced a diagnosis-dedicated hybrid formalism relying on error bounds for the detection parts, but without concurrence nor transitions triggered autonomously from the continuous level; it uses probabilities, parameter estimation as well as data fitting to refine the diagnosis. [20] unifies traditional continuous state observers with hidden Markov models belief update in order to track hybrid systems with noise but do not include concurrent models nor any mapping function discussion. The approach is interesting because it makes extensive use of probabilities where we chose to rely on bounded uncertainties (intervals) at the continuous level and on probabilities at the discrete level. In fact these are different uncertainties as the uncertainty is uniformly distributed in the case of intervals whereas [20] relies on normal laws. In our point of view using probabilities at the discrete levels allows to prune an otherwise prohibitive search, but intervals offer a more compact representation of uncertainties on continuous variables. However, the point would need more discussion and research. Similar approaches also include [21] that combines a Petri net and signal analysis to estimate the discrete modes and overcome an exponential cost in the number of sensors, but lacks an efficient diagnosis engine; and [7] that uses a dedicated bayesian network as well as a method of smoothing that helps successfully diagnose faults with a very low belief state. Note that the model checking community has recently investigated the use of interval-based numerical models [5].

An advantage of our approach is that any type conditions associated to transitions and influences (e.g. continuous functions as guards) can be modeled and tracked without being directly observed. Finally on-line performances can be enhanced as the formalism allows the logical model to be pre-compiled before use by generating prime-implicants on transition guards [19] and influence conditions. However it still happens that trajectories cannot be discriminated due to too much imprecision on parameters that leads to overlapping envelopes. A solution to this problem has been to merge such envelopes and corresponding trajectories. Another remark concerns the splits that occur and are not linked to any real mode or structure changes in the model: when starting the thermostat and room models with external temperature $x_{ext} < m$, a split occurs when first crossing at $x = m$. These splits however are sound and refine the bounds on continuous variables as they allow the system to reduce temporal uncertainty at the crossing point.

Further work will focus on reconfiguration by reasoning on configurations with the same core algorithms as for diagnosis. This will be done by identifying a set of goal configurations and find under uncertainty a valid plan made of least costly endogeneous commands to reach each goal. We think that additive improvements could also include automatic controller synthesis as in [2] as well as parameter estimation based on the causal structure of the continuous level in order to refine the tracking of the system when in its *unknown* mode. In a near future more results are to come out as our implementation is intended to be tested on spacecraft models and run on-board ground based satellite hardware.

6 Acknowledgements

We are very thankful to Marie-Claire Charmeau and Bernard Polle for providing information about application and valuable comments on this work.

REFERENCES

[1] R. Alur, C. Courcoubetis, N.Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A.Olivero, J.Sifakis, and S.Yovine, 'The algorithmic analysis of hybrid systems', in *Proceedings of the 11th International Conference on Analysis and Optimization of Discrete Event Systems*, pp. 331–351, (1995).

[2] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, 'Effective controller synthesis of switching controllers for linear systems', *Proceedings of the IEEE, Special Issue on Hybrid Systems*, **88**, 1011–1025, (July 2001).

[3] E. Benazera, L. Travé-Massuyès, and P. Dague, 'Hybrid model-based diagnosis for autonomous spacecrafts', in *Proceedings of the first ESA Workshop on On-Board Autonomy, October 2001, Nordwijk, Netherlands*, pp. 279 – 286, (2001).

[4] T. Henzinger, 'The theory of hybrid automata', in *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pp. 278–292, New Brunswick, New Jersey, (1996).

[5] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi, 'Beyond HYTECH: Hybrid systems analysis using interval numerical methods', in *HSCC*, pp. 130–144, (2000).

[6] E. Hyvonen, 'Constraint reasoning based on interval arithmetic: The tolerance propagation approach', *Artificial Intelligence*, **58**(1-3), 71–112, (1992).

[7] Uri Lerner, Ronald Parr, Daphne Koller, and Gautam Biswas, 'Bayesian fault detection and diagnosis in dynamic systems', in *AAAI/IAAI*, pp. 531–537, (2000).

[8] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems - Specification*, Springer-Verlag, 1992.

[9] S. McIlraith, G. Biswas, D. Clancy, and V. Gupta, 'Towards diagnosing hybrid systems', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis DX-99*, (1999).

[10] P. J. Mosterman and G. Biswas, 'A comprehensive methodology for building hybrid models of physical systems', *Artificial Intelligence*, **121**, 171–209, (2000).

[11] P. Nayak and J. Kurien, 'Back to the future for consistency-based trajectory tracking', in *Proceedings of AAAI-2000, Austin, Texas*, (2000).

[12] R. Pons, L. Travé-Massuyès, and M. Porcheron, 'Model-based diagnosis and maintenance of time-varying dynamic systems', in *Proceedings of the Tenth International Workshop on Principles of Diagnosis DX-99*, pp. 211–219, (1999).

[13] L. Travé-Massuyès, T. Escobet, R. Pons, and S. Tornil, 'The ca-en diagnosis system and its automatic modeling method', *Computación i Sistemas Journal*, **5**(2), 128–143, (2001).

[14] L. Travé-Massuyès and J.A. Jimenez, 'Fault detection and isolation in the ca-en system', Technical report, LAAS-CNRS, Toulouse, France, (2001).

[15] L. Travé-Massuyès and R. Milne, 'Tigertm: Gas turbine condition monitoring using qualitative model based diagnosis', *IEEE Expert Intelligent Systems & Applications*, (1997).

[16] L. Travé-Massuyès and R. Pons, 'Causal ordering for multiple modes systems', in *Proceedings of the Eleventh International Workshop on Qualitative Reasoning*, pp. 203 – 214, (1997).

[17] B. C. Williams and P. Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of AAAI-96, Portland, Oregon*, pp. 971–978, (1996).

[18] B. C. Williams and P. Nayak, 'Fast context switching in real-time reasoning', in *Proceedings of AAAI-97, Providence, Rhode Island*, (1997).

[19] B. C. Williams and P. Nayak, 'A reactive planner for a model-based executive', in *Proceedings of IFCAI-97*, (1997).

[20] B.C. Williams, M. Hofbaur, and T. Jones, 'Mode estimation of probabilistic hybrid systems', Technical report, Massachusset Institute of Technology, (2002).

[21] Feng Zhao, Xenofon D. Koutsoukos, Horst W. Haussecker, James Reich, Patrick Cheung, and Claudia Picardi, 'Distributed monitoring of hybrid systems: A model-directed approach', in *IJCAI*, pp. 557–564, (2001).