Automatic Construction of Diagnoser for Complex Discrete Event Systems

E. Gascard, Zineb Simeu-Abazi

Automatic Construction of Diagnoser for Complex Discrete Event Systems

Eric Gascard
TIMA laboratory (CNRS - Grenoble INP - UJF)
46 avenue Félix Viallet 38031 Grenoble Cedex FRANCE
Email: eric.gascard@imag.fr

Zineb Simeu-Abazi
G-SCOP laboratory (CNRS - Grenoble INP - UJF)
46 avenue Félix Viallet 38031 Grenoble Cedex FRANCE
Email: zineb.simeu-abazi@g-scop.grenoble-inp.fr

***Abstract*—This paper deals with the problem of fault diagnosis of complex discrete event systems in the context of communicating timed automata. Indeed, for the diagnosis, this kind of systems can be represented by timed models whose components communicate through channels. This paper starts with a description of our modelling methodology of discrete event systems as communicating timed automata. The proposed approach for diagnosis (detection and isolation) is based on the methodology known as the diagnoser approach. This paper extends the approach of diagnoser through the taking into account of the various communicating synchronized automata representing the components of the system. It proposes an automatic step of construction of the global model. The application of the proposed algorithm allows to obtain the diagnoser of the studied system. Starting from a model of the complex system, this approach computes a deterministic automaton, called a diagnoser, which uses observable events to detect the occurrence of a failure. The different steps of the proposed method are described by algorithms and illustrated through a batch process.**

I. Introduction

For the complex systems, the development of a methodology of fault diagnosis is of principal importance. Indeed, for such systems, the diagnosis contributes to the improvement of the availability, the growth of production and of course, the reduction of maintenance costs. It is a key action in the improvement of performance of industrial feature. A fault diagnosis system is able to detect (an indication that something is going wrong in the system) and isolate faults (determination of the type of faults and their location).

In continuous systems, residuals are used for diagnosis. Residuals describe inconsistencies responses between the actual system behavior and the model. The residual value is used to detect any fault. In the Discrete Event Systems (or DES) area, the most common diagnosis approach is the so-called model-based diagnosis, which uses the inputs and outputs of the system under supervision to detect the fault and isolate (locate, distinguish) the source of failure.

In this paper we study the problem of fault detection and isolation by model-based diagnosis methods in the context of timed discrete event systems modeled as networks of communicating timed automata (CTA). This work is a continuation of the PhD thesis from Dr. Michal Knotek [1], [2]. For the validation of the proposed method, Uppaal notation [3] is used to illustrate CTA, knowing that our models can easily be adapted to other timed automata tools.

Existing model-based diagnosis approaches typically fall into the following three categories. The first one is called *off-line diagnosis*: the behaviors of the system under supervision are stored in a file which can be exploited by the diagnoser. The second called *passive on-line diagnosis*, the system under supervision and the diagnoser run in parallel, with the latter observing passively the behaviors progressively, see e.g. [4]. In the third called *active on-line diagnosis*, input test sequences for fault diagnosis are computed, so that the diagnoser may influence the choice of inputs, see e.g. [5].

This paper uses the principle of *passive on-line diagnosis* where the construction of the diagnoser is made off-line.

The early work on the model-based diagnosis problem has been reported by Sampath et al. in [4]. Their untimed approach consists in transforming the DES to be diagnosed into a finite state automaton, called a diagnoser, that uses the history of events to detect the occurrence of a failure. However with the presence of real-time systems anywhere, timed DES models become essential. There has been some research on diagnosis of timed discrete event systems, we restrict our discussion to work closely related to automata, e.g. [6]–[12].

In [6], passive on-line diagnosis in timed discrete-event systems is performed by Zad et al., based on a framework incorporating time as an extra event, called clock tick. The authors design a diagnoser using the methodology presented in [13] for untimed cases.

In [7], Tripakis proposes a passive on-line diagnosis to dense-time automata based on state estimation in a timed automaton with ε-transitions, its complexity to diagnose faults from an observation is exponential in the size of the plant and in the size of the observation [14].

In [8], a timed extension of the Sampath et al. diagnosis approach is proposed. The diagnoser is a timed automaton constructed off-line. In their approach to handle the state space problem, the authors use zone representation for partitioning the state space into a set of symbolic states (zones). However no comment were made about the synthesis of their diagnoser.

Lunze and Supavatanakul present in [9] a model-based diagnosis method of DES described by timed automata. They apply the idea of consistency-based diagnosis to timed automata.

These related works require the computation of a global model of the system (centralised approach) which is not always possible with large discrete event systems due to the state

explosion problem. To handle large DES, there exist methods relying on a decentralised model [15] such as [10], [11]. These decentralised diagnoser approaches compute a diagnosis for each component of the system (local diagnosis) and then build a diagnosis of the whole system (global diagnosis) by merging these local diagnoses.

In [12], Lamperti and Zanella propose a diagnostic method that mixes the diagnoser approach of Sampath et al. with an extended version of the decentralised model.

The presentation of the proposed approach in this paper is structured as follows: Section II presents the main ideas of our approach and discusses the differences between our method with the existing ones. The timed automata with the necessary notation are presented in section III. In section IV, we present the diagnoser construction procedure and a case study to illustrate our method. Section V concludes the paper.

II. Proposition of a new method for diagnosis of timed systems

The system to be diagnosed is described as a network of communicating timed automata composed of a controller $\mathcal{C}$, a plant $\mathcal{P}$, n actuators $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and m sensors $\mathcal{S}_1, \ldots, \mathcal{S}_m$. We consider only permanent faults on actuators and sensors. Our method proposes *passive on-line diagnosis*. The observable events are commands issued by the controller to the actuators and sensor readings. The unobservable events are actuator readings modifying the plant behavior and commands issued by the plant behavior to the sensors. The diagnoser has access to only observable events.

A first difference with the Sampath et al. approach concerns the model used to represent the system: we use communicating timed automata and we do not represent faults as unobservable events, but as particular states of the timed automata.

Figure 1 shows the principle architecture of the method which is decomposed into two major steps.

The first step is the automatic construction of a timed automaton, named *draft diagnoser* $\mathcal{G}$ as follows:

- Computation of a restricted part of the composition $\mathcal{C}\|\mathcal{A}_1\|\ldots\|\mathcal{A}_n\|\mathcal{S}_1\|\ldots\|\mathcal{S}_m\|\mathcal{P}$. We do not explore the whole global model: when multiple transitions are possible, if some involve controller, we choose only these transitions. Indeed, the plant status is determined by the controller's actions. It is useless to add some states that cannot be reached. We say that *we give greater priority to controller transitions*. So, each state of $\mathcal{G}$ corresponds to a vector $(\ell^c, \ell^{a1}, \ldots, \ell^{an}, \ell^{s1}, \ldots, \ell^{sm}, \ell^p)$ where each element is a state of a component.
- Simplification of the labelled transitions of the restricted global model: we remove guards and updates that involve unobservable events. So, the transitions of our *draft diagnoser* are composed of a guard: a Boolean expression on observable events (sensors values or control command for actuators) with the associated time.

The second step is the construction of the *diagnoser* from the previous timed automaton: we "determinise" the automaton and construct if necessary new transitions for the faults isolations.

A second difference with the Sampath et al. approach relates the ability in the isolation of faults. Indeed, thanks to our diagnoser, the timing constraints taken into account, we are able to distinguish faults indistinguishable in the untimed approach.

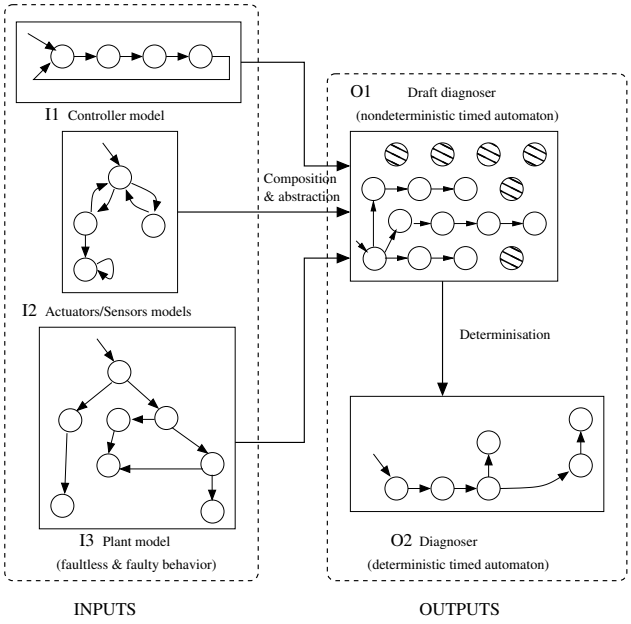


Fig. 1. Principle of the diagnoser construction

III. Background on timed automata

A. Formal syntax of timed automaton

A timed automaton [16] is a finite-state machine extended with clock variables. It uses a dense-time model where a clock variable evaluates to a real number. All the clocks progress synchronously. Throughout the paper we use Uppaal syntax to illustrate timed automata.

A timed automaton is described by $\mathcal{A} = (L, \ell_0, Sync, Clk, V, E, Init, I)$ where:

- L is a finite set of states of the timed automaton, called *locations*; $\ell_0 \in L$ is the initial location;
- $Sync$ is a set of synchronisation actions which includes actions, co-actions, and internal τ-action. An action *emit* over a channel a is denoted by $a!$ and its co-action *receive* is denoted by $a?$. The τ-action is an internal action such that there is no synchronisation label;
- Clk is the set of clocks;
- V is the set of integer variables;
- $E \subseteq L \times \mathcal{B}(Clk, V) \times Sync \times Updates \times L$ is a set of edges (transitions) between locations with a synchronisation action conditioned by a guard and performing a sequence of assignments. We shall write $\ell \xrightarrow[u]{g,s} \ell'$ when $(\ell, g, s, u, \ell') \in E$. We use $\mathcal{B}(Clk, V)$ as the set of constraints allowed in guards and invariants. We use $Updates$ as the set of sequences of assignments of the

form $x_1 = e_1, \ldots, x_n = e_n$, where x_i refers to an integer variable or a clock and e_j is an expression. Assignments are performed after the guard evaluation and in a sequential manner (not concurrently). On synchronising edges, the assignments on the !-side (the emitting side) are evaluated before the ?-side (the receiving side).
- $Init \subseteq Updates$ is a set of assignments that assign the initial values to variables;
- $I : L \rightarrow \mathcal{B}(Clk, V)$ is function associated with each location. For each location ℓ, $I(\ell)$ denotes its *invariant*, i.e. the timed automaton may stay in the location, as long as the invariant is satisfied.

B. Illustrative example: Batch process

We will illustrate our diagnoser construction on a simple didactic example: a small batch neutralization process. The process represents a mixture of two ingredients in one tank to obtain a final product. The tank is equipped by two level sensors $L1, L2$ and two input valves $V1, V2$. Filling in tank must respect the following control sequence: valve $V1$ is opened, an ingredient 1 flows into tank. When the level $L1$ is reached, the valve $V1$ is closed and $V2$ is opened. After the sensor $L2$ indicates that level is reached, the valve $V2$ is closed.

We model each equipment (valves and sensors), the control sequence and the tank's behavior as timed automata. The figures are directly exported from Uppaal. Initial locations are marked using a double circle. Edges are by convention labelled by the triplet: guard, action, and assignment in that order. The internal τ-action is indicated by an absent action-label.

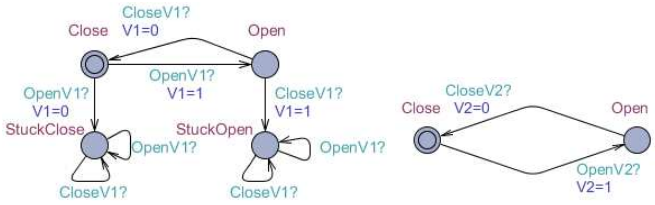
1) Timed automaton for the controller: The commands to open and close the valves are modelled as synchronisation actions $OpenV1!, CloseV1!, OpenV2!, CloseV2!$. The state of the levels $L1$ and $L2$ are expressed as integer variables, initialized to 0, and updated to 1 when the level is reached. Figure 2 describes the timed automaton of the controller.

Fig. 2. Timed automaton for the controller

2) Timed automata for the valves: We model the real state of the valves $V1$ and $V2$ as integer variables $V1$ and $V2$, initialized to 0. When the valve is opened, the associated variable equals 1, 0 when the valve is closed. For the modelling of valve $V1$, we consider the following faults:

- F_1: Fault valve $V1$ being stuck close. Practically it means, that tank stays in the initialized state. Controller waits from event $L1$ which can not occur because of the stuck valve.
- F_2: Fault valve $V1$ being stuck open. This fault can physically cause an overflow.

Figure 3(a) describes the timed automaton of the valve $V1$ with faults consideration. Valve $V1$ in the faultless mode can be in the two states: $Close, Open$. The initial state is $Close$. The state can be changed by the synchronisation co-action $CloseV1?$ and $OpenV1?$. The description of the faulty behavior of $V1$ corresponds to setting the variable $V1$ with an incorrect value leading to the faulty states $StuckClose$ and $StuckOpen$. Figure 3(b) describes the timed automaton of the valve $V2$ without faults.



(a) Valve $V1$ (b) Valve $V2$

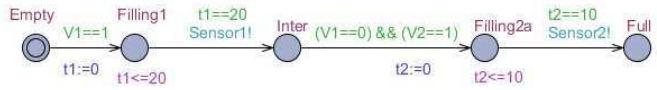
Fig. 3. Timed automata for the valves

3) Timed automata for the sensors: The real state of the sensors $L1$ and $L2$ is modelled as integer variables $L1$ and $L2$, initialized to 0. When the level $L1$ (resp. $L2$) is reached, the associated variable equals 1. For the modelling of sensor $L1$, we consider the following fault F_3: Sensor $L1$ stays in close position. It means when the level $L1$ is reached, this sensor does not indicate it. Figure 4(a) describes the timed automaton of the sensor $L1$ with faults consideration. Sensor $L1$ in the faultless mode can be in the two states: $Down, Up$. The initial state is $Down$. The state can be changed by the synchronisation co-action $Sensor1?$. The description of the faulty behavior of $L1$ corresponds to setting the variable $L1$ with an incorrect value leading to the faulty state $StuckDown$. Figure 4(b) describes the timed automaton of the sensor $L2$ without faults.

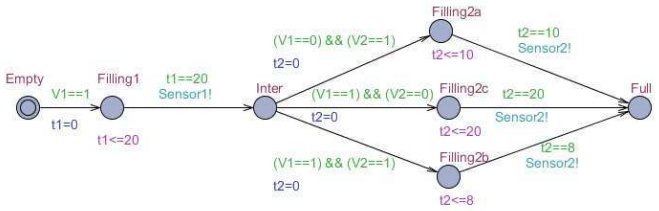
(a) Sensor $L1$ (b) Sensor $L2$

Fig. 4. Timed automata for the sensors

4) Timed automaton for the tank behavior: The physical behavior of the tank is represented by an Uppaal timed automaton. First, we define its faultless behavior (figure 5(a)), second we extend it with faulty behavior (figure 5(b)). For the diagnosis purposes, it is needed that the tank model covers any possible behavior. We use two clocks, $t1, t2$, to observe the filling up to the level $L1$ and $L2$. Dynamics is implemented using timed transitions (guards are expressed as conditions on clocks) which observed the respective clock, e.g. from state $Empty$ to $Inter$ (level $L1$ is reached by opening the valve $V1$ while the valve $V2$ is still closed), the automaton reaches the state $Inter$ in time 20. These timed transitions produce an action corresponding to the sensor reading $L1, L2$. For this purpose, the synchronisation action $Sensor1!$ and $Sensor2!$ are used.



(a) Batch process with faultless behavior



(b) Batch process with faultless and faulty behavior

Fig. 5. Timed automata for the batch process

IV. Diagnoser construction

For this section, we fix a complex discrete event system $\mathcal{M}$ modelled as a network of communicating timed automata sharing the same sets of synchronisation actions $Sync$, clocks Clk, integer variables V and initial assignments $Init$. Our diagnoser construction procedure involves two phases. In the first phase, we construct a timed automaton named *draft diagnoser* $\mathcal{G}$, which is the abstracted partial model of the composition of the components. In the second phase, the diagnoser $\mathcal{D}$ is obtained by determinisation of $\mathcal{G}$.

A. Construction of the draft diagnoser

The algorithm for computing the draft diagnoser is given by Algorithm 1.

Algorithm 1 Computation of the draft diagnoser

Require: $\mathcal{C}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{S}_1, \ldots, \mathcal{S}_m$

Ensure: draft diagnoser $\mathcal{G} = (L^g, \ell_0^g, Sync, Clk, V, E^g, Init, I^g)$

1: Initialization: $\ell_0^g = (\ell_0^c, \ell_0^{a1}, \ldots, \ell_0^{an}, \ell_0^{s1}, \ldots, \ell_0^{sm}, \ell_0^p)$; $L^g = \{\ell_0^g\}$; $Waiting = \{\ell_0^g\}$;
2: **while** $Waiting \neq \emptyset$ **do**
3: $\quad \ell^g = Waiting.pop()$;
4: $\quad lst\_trans = Transitions(\ell^g)$;
5: $\quad (lst\_trans_{Ctrl}, lst\_trans_{Plant}) = Partition(lst\_trans)$;
6: $\quad$ {we partition the set of transitions according to the role of the controller or the plant in the transition}
7: $\quad$ **if** $lst\_trans_{Ctrl} \neq \emptyset$ **then**
8: $\quad\quad$ {we use a transition directed by the controller}
9: $\quad\quad$ **for all** transition $e \in lst\_trans_{Ctrl}$ **do**
10: $\quad\quad\quad$ **if** $e.destination \notin L^g$ **then**
11: $\quad\quad\quad\quad Waiting.push(e.destination)$;
12: $\quad\quad\quad\quad L^g = L^g \cup \{e.destination\}$;
13: $\quad\quad\quad$ **end if**
14: $\quad\quad\quad$ **if** $e.sync = \emptyset$ **then**
15: $\quad\quad\quad\quad$ {case $\ell_u^c \xrightarrow{g} \ell_v^c$ where g is a Boolean expression on the status of the sensors}
16: $\quad\quad\quad\quad E^g = E^g \cup \{\ell^g \xrightarrow{g} e.destination\}$;
17: $\quad\quad\quad$ **else**
18: $\quad\quad\quad\quad$ {case $\ell_u^c \xrightarrow[sentactuator_i=1]{actuator_i!} \ell_v^c$ synchronised with $\ell_u^{ai} \xrightarrow[v=value]{actuator_i?} \ell_v^{ai}$}
19: $\quad\quad\quad\quad E = E \cup \{\ell^g \xrightarrow{sentactuator_i==1} e.destination\}$;
20: $\quad\quad\quad$ **end if**
21: $\quad\quad$ **end for**
22: $\quad$ **else**
23: $\quad\quad$ {we use a transition directed by the plant}
24: $\quad\quad$ **for all** transition $e \in lst\_trans_{Plant}$ **do**
25: $\quad\quad\quad$ **if** $e.destination \notin Q$ **then**
26: $\quad\quad\quad\quad Waiting.push(e.destination)$;
27: $\quad\quad\quad\quad Q = Q \cup \{e.destination\}$;
28: $\quad\quad\quad$ **end if**
29: $\quad\quad\quad$ **if** $e.sync = \emptyset$ **then**
30: $\quad\quad\quad\quad$ {case $\ell_u^p \xrightarrow[clk_i=0]{g_1} \ell_v^p$ where g_1 is a Boolean expression on the status of the actuators}
31: $\quad\quad\quad\quad E = E \cup \{\ell^g \xrightarrow[clk_i=0]{} e.destination\}$;
32: $\quad\quad\quad$ **else**
33: $\quad\quad\quad\quad$ {case $\ell_v^p \xrightarrow{g_2, sensor_j!} \ell_w^p$ synchronised with $\ell_u^{sj} \xrightarrow[v=value]{sensor_j?} \ell_v^{sj}$ where g_2 is a Boolean expression on the clock clk_i}
34: $\quad\quad\quad\quad E = E \cup \{\ell^g \xrightarrow{g_2 \&\& v==value} e.destination\}$;
35: $\quad\quad\quad$ **end if**
36: $\quad\quad$ **end for**
37: $\quad$ **end if**
38: **end while**

The interresting idea is to apply a reachability analysis on the composition $\mathcal{C}\|\mathcal{A}_1\|\ldots\|\mathcal{A}_n\|\ \mathcal{S}_1\|\ldots\|\mathcal{S}_m\|\mathcal{P}$ and to limit the exploration of the state space by giving greater priority to controller transitions. The main data structure is a (FIFO) queue $Waiting$ to hold the reachable states of $\mathcal{G}$ in postorder. To express the composition $\mathcal{C}\|\mathcal{A}_1\|\ldots\|\mathcal{A}_n\|\mathcal{S}_1\|\ldots\|\mathcal{S}_m\|\mathcal{P}$, the states of $\mathcal{G}$ are defined as tuples of states of the components. So, the initial state of $\mathcal{G}$, ℓ_0^g is expressed as $(\ell_0^c, \ell_0^{a1}, \ldots, \ell_0^{an}, \ell_0^{s1}, \ldots, \ell_0^{sm}, \ell_0^p)$.

Priority given to controller transitions is expressed in lines 5-7: we partition the set of available transitions according to the role of the controller and use only these transitions, if present, for the reachability analysis. We construct an abstracted model: we do not label the transitions with unobservable events.

In lines 19 and 31, the real status of the actuators, unobservables, do not appear as guard of the building transitions. However, we use some integer variables $sentactuator_i$ to memorize the request on $actuator_i$.

B. Construction of the diagnoser

The timed automaton obtained by Algorithm 1 can not be used as a diagnoser due to its nondeterminism. So we need a step of determinisation. Algorithm 2 presents our determinisation procedure. It is based on the subset construction method: the basic idea underlying the transformation is the use of sets of states of the

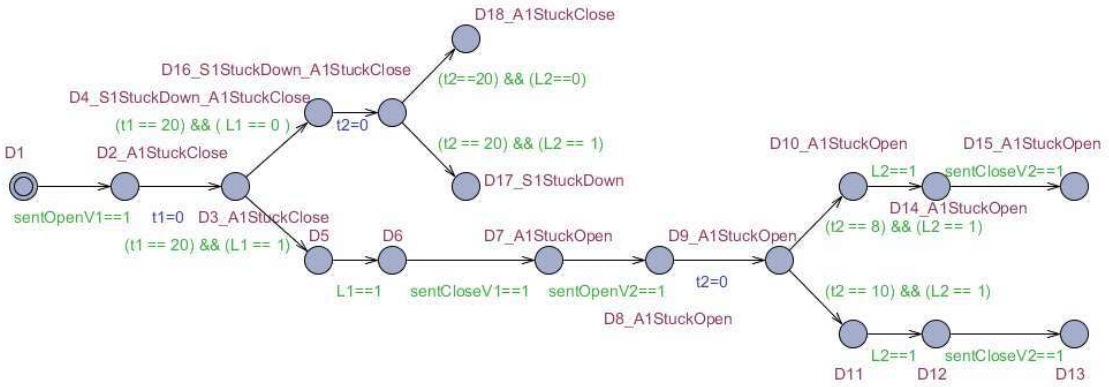


Fig. 6. Diagnoser of the batch process constructed by Algorithm 2

nondeterministic automaton as states in the corresponding equivalent deterministic automaton. The initial state of the diagnoser is formed by a singleton that contains the initial state in the draft diagnoser. As in Algorithm 1, we use a (FIFO) queue $Waiting$ to hold the new states of the diagnoser in postorder.

Algorithm 2 Computation of the diagnoser

Require: draft diagnoser $\mathcal{G} = (L^g, \ell_0^g, Sync, Clk, V, E^g, Init, I^g)$

Ensure: diagnoser $\mathcal{D} = (L^d, \ell_0^d, Sync, Clk, V, E^d, Init, I^d)$

1: Initialization: $\ell_0^d = \{\ell_0^g\}; L^d = \{\ell_0^d\}; Waiting = \{\ell_0^d\};$
2: **while** $Waiting \neq \emptyset$ **do**
3: $\ell^d = Waiting.pop();$
4: $(states\_with\_arcs, states\_without\_arcs) = Partition(\ell^d);$
5: {we partition the set of states ℓ^d according to they are or not source of transitions in $\mathcal{G}$}
6: **for all** $\ell^g \in states\_with\_arcs$ **do**
7: $\{\ell^d = \{\ldots, \ell^g, \ldots\}$ and ℓ^g has some transitions in $\mathcal{G}\}$
8: $lst\_trans = Transitions(\ell^g);$
9: **for all** $e \in lst\_trans$ **do**
10: **if** $\exists \ell'^d \in L^d$ such that $e.destination \in \ell'^d$ **then**
11: {$e.destination$ is present in of another state of the diagnoser, so it is unnecessary to create a new state due to property **??** }
12: $E^d = E^d \cup \{\ell^d \xrightarrow[e.assignment]{e.guard} \ell'^d\};$
13: **else if** $\exists\{\ell^d \xrightarrow[e.assignment]{e.guard} \ell'^d\} \in E^d$ **then**
14: {there exists already a transition in the diagnoser starting from ℓ^d with the same guard/assignment of transition e, so we complete the subset of states ℓ'^d with $e.destination$}
15: $\ell'^d = \ell'^d \cup \{e.destination\};$
16: **else**
17: Create new state $\ell'^d = \{e.destination\};$
18: $L^d = L^d \cup \{\ell'^d\};$
19: $E^d = E^d \cup \{\ell^d \xrightarrow[e.assignment]{e.guard} \ell'^d\};$
20: $Waiting.push(\ell'^d);$
21: **end if**
22: **end for**
23: **end for**
24: **for all** $\ell^g \in states\_without\_arcs$ **do**
25: $\{\ell^d = \{\ldots, \ell^g, \ldots\}$ and ℓ^g has no transition in $\mathcal{G}\}$
26: **if** $\exists\{\ell^d \xrightarrow[assignment]{guard} \ell'^d\} \in E^d$ such that $guard$ is empty **then**
27: $\ell'^d = \ell'^d \cup \{\ell^g\}$
28: **else if** $\exists\{\ell^d \xrightarrow[assignment]{guard} \ell'^d\} \in E^d$ such that $guard$ is satisfied in ℓ^g **then**
29: $\ell'^d = \ell'^d \cup \{\ell^g\}$
30: **else**
31: Create new state $\ell'^d = \{\ell^g\};$
32: $L^d = L^d \cup \{\ell'^d\};$
33: $E^d = E^d \cup \{\ell^d \xrightarrow{to\_be\_defined} \ell'^d\};$
34: **end if**
35: **end for**
36: **end while**

While there is a state ℓ^d of the diagnoser for which the transitions have not been determined (loop **While** line 2), we do the following:

- According to the nature of source of transitions in the draft diagnoser, the set of states ℓ^d is decomposed. The set of states $states\_with\_arcs$ and $states\_without\_arcs$ (line 4) are obtained.
- From $states\_with\_arcs$ (lines 6-23), we create new sets of states ℓ'^d (new states in the diagnoser) composed by the set of destination of transitions e having their sources in $states\_with\_arcs$ and with the same guards/assignments. We create new transitions in the diagnoser $\ell^d \xrightarrow[e.assignment]{e.guard} \ell'^d$.
- From $states\_without\_arcs$ (lines 24-35), we complete the set of states ℓ'^d computed at the previous step by using transitions with empty guard or we create new sets of states composed by a single state $\ell'^d = \{\ell^g\}$. For these later states, theirs incoming transitions labelled

with $to\_be\_defined$ must be defined manually as the negation of a Boolean formula composed by the disjunction of all the others guards starting from ℓ^d.

Application to the batch process:
Figure 6 shows the diagnoser: we have substituted the guard $to\_be\_defined$ by $(t2 == 20)\&\&(L2 == 0)$. This new guard is deduced by its neighbor $(t2 == 20)\&\&(L2 == 1)$.

The isolation of faults in the diagnoser is characterized by the particular labelling of faulty states: labels are composed by the name of the faulty component and its type of fault. These informations come from the labelling of states of the draft diagnoser which mention the status of each component. For example, in figure 6, several states of the diagnoser are named with $A1StuckClose$, $A1StuckOpen$ or $S1StuckDown$.

C. Formal verification of the diagnoser

The construction of the diagnoser without verifying its correctness is needless. We must prove that the diagnoser is correct, in the sense that it announces a fault if and only if the fault has occurred. We apply model-checking techniques [17] proposed in [18] for obtaining the formal verification of our diagnosers. Thanks to our modeling in Uppaal, we can use its model-checker. Verification compares reachable states in global model $\mathcal{M}$ (composed by timed automata components, $\mathcal{M} = \mathcal{C}\|\mathcal{A}_1\| \ldots \|\mathcal{A}_n\|\mathcal{S}_1\| \ldots \|\mathcal{S}_m\|\mathcal{P}$) and diagnoser $\mathcal{D}$. The fact that an actuator or a sensor contains the fault event F_k corresponds to the fact that $\mathcal{M}$ reaches the state $\mathcal{M}.F_k$. In a similar way, the fault isolation F_k by the diagnoser is formulated $\mathcal{D}.F_k$.

To verify the correctness of the diagnoser, we are taking the synchronous composition of the global model $\mathcal{M}$ and the diagnoser model $\mathcal{D}$. We must verify:

1) that there are no missed detections. If a fault F_k occurs then the diagnoser must eventually lead to state in which it is isolated. Formally, this property is expressed by the CTL formulae $A\Box(\mathcal{G}.F_k \rightarrow A\Diamond\mathcal{D}.F_k)$.
2) that there are no false alarms. Every diagnoser result of fault isolation F_k is caused by this fault. Hence, the property to verify is formulated as follows: $A\Box(\mathcal{D}.F_k \rightarrow \mathcal{G}.F_k)$.

V. Conclusion

This paper proposes a model-based approach to passive online fault diagnosis for timed systems. We suppose the system to be diagnosed is described as a network of communicating timed automata. Each component describes an actuator or a sensor or the control sequence or the process. We consider only permanent faults on actuators and sensors. The first contribution of this paper is the formalization of the systems that we are able to take into account. The second contribution is the definition of a diagnoser for such systems and algorithms to build it are given. Our construction method involves two steps. Firstly we build the abstracted partial model of the composition of the components, we obtain a draft version of our diagnoser: it is a nondeterministic timed automaton. So, in a second step, we determinise it to obtain the final diagnoser. Our approach has been implemented and experimented on a batch process. The diagnoser software is based on two parts: the core implemented in C++ and the graphic interface implemented in GTK+. The core implements Algorithms 1 & 2 and generates the Uppaal description of the diagnoser.

We have applied model-checking techniques for the formal verification of our diagnosers. The property to be checked expresses that the diagnoser is correct if it announces a fault if and only if the fault already exists in the system model. A perspective of this work is to extend our approach to take into account uncertainly in task durations that appear in some process, for example in manufacturing systems. Another issue not addressed in this paper is the study of the diagnosability of failures, and obtain necessary and sufficient conditions for failure diagnosability.

References

[1] M. Knotek, "Fault diagnostics based on temporal analysis," Ph.D. dissertation, University Joseph Fourier - Grenoble (FRANCE) and BRNO University of Technology (Czech Republic), 2006.

[2] Z. Simeu-Abazi, M. Di Mascolo, and M. Knotek, "Fault diagnosis for discrete event systems: Modelling and verification," *Reliability Engineering & System Safety*, vol. 95, no. 4, pp. 369–378, 2010.

[3] G. Behrmann, A. David, and K. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems*, ser. LNCS, vol. 3185, 2004, pp. 200–236.

[4] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555–1575, 1995.

[5] F. Lin, "Diagnosability of discrete event systems and its applications," *Discrete Event Dynamic Systems*, vol. 4, no. 2, pp. 197–212, 1994.

[6] S. Zad, R. Kwong, and W. Wonham, "Fault diagnosis in discrete-event systems: incorporating timing information," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 1010–1015, 2005.

[7] S. Tripakis, "Fault diagnosis for timed automata," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ser. LNCS, vol. 2469, 2002.

[8] H. Derbel, M. Yeddes, N. Hadj-Alouane, and H. Alla, "Diagnosis of a class of timed discrete event systems," in *8th International Workshop on Discrete Event Systems*, 2006, pp. 256–261.

[9] J. Lunze and P. Supavatanakul, "Diagnosis of discrete-event system described by timed automata," in *Proceedings of IFAC 15th World Congress*, 2002, pp. 77–82 Vol J: Fault Detection and Supervision.

[10] Y. Pencolé and M.-O. Cordier, "A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks," *Artificial Intelligence*, vol. 164, 2005.

[11] M. Sayed-Mouchaweh, A. Philippot, and V. Carre-Menetrier, "Decentralized diagnosis based on boolean discrete event models: application on manufacturing systems," *International Journal of Production Research*, vol. 46, pp. 5469–5490, 2008.

[12] G. Lamperti and M. Zanella, "Continous diagnosis of discrete-event systems," in *Proceedings of the Workshop on Principles of Diagnosis, DX'03*, 2003, pp. 105–112.

[13] S. H. Zad, R. Kwong, and W. Wonham, "Fault diagnosis in discrete-event systems: framework and model reduction," *IEEE Transactions on Automatic Control*, vol. 48, no. 7, pp. 1199–1212, 2003.

[14] P. Bouyer and F. Chevalier, "Fault diagnosis using timed automata," in *Foundations of Software Science and Computational Structures*, ser. LNCS, vol. 3441, 2005, pp. 219–233.

[15] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, "Diagnosis of large active systems," *Artificial Intelligence*, vol. 110, pp. 135–183, 1999.

[16] R. Alur and D. Dill, "A theory of timed automata," *Theorical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[17] E. M. Clarke and B.-H. Schlingloff, "Model checking," in *Handbook of Automated Reasoning*, J. A. Robinson and A. Voronkov, Eds., 2001, vol. 2, pp. 1635–1790.

[18] M. Knotek, Z. Simeu-Abazi, and F. Zezulka, "Fault diagnosis based on timed automata: Diagnoser verification," in *IMACS Multiconference on Computation Engineering in Systems Applications, CESA'06*, 2006.