HAL
open science

Comparing detailed and abstract timed models of automated discrete manufacturing systems

Matthieu Perin, Jean-Marc Faure

To cite this version:

Matthieu Perin, Jean-Marc Faure. Comparing detailed and abstract timed models of automated discrete manufacturing systems. 9th annual IEEE International Conference on Automation Science and Engineering (CASE 2013), Aug 2013, Madison (WI), United States. pp.TuBT5.2. hal-00865616

HAL Id: hal-00865616

https://hal.science/hal-00865616v1

Submitted on 24 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Comparing detailed and abstract timed models of automated discrete manufacturing systems

Matthieu Perin[1] and Jean-Marc Faure[1,2]

***Abstract*—This paper proposes an equivalence relation that permits to compare the external behaviors of two models of the same component of an automated discrete system: a detailed model suitable to design the control of this component and a more abstract model where only the order and the durations of the operations performed by this component are considered. A first definition of the relation is given assuming that both models are deterministic; then a second definition that integrates tolerances on values and time is proposed when this assumption is no more true. These definitions are exemplified on a simple case study.**

I. Introduction

A component of a discrete automated manufacturing system, e.g. a conveyor, machine, robot, may be modeled in a detailed fashion as a closed-loop system where a controller is connected to plant elements. This modeling is helpful when designing the controller, for instance to check whether a specific control algorithm ensures liveness and safety; it is then very suitable for control engineers. However, a more abstract model of the same component may be built by considering that this component performs some operations on a discrete flow of parts without focusing on the details of these operations - the internal behavior of the closed loop does not matter- but only on the time they require and their start/end events. This modeling is relevant when focus is put on time performance evaluation of the whole automated system; therefore it is more appropriate for production engineering and management concerns. These models must be timed because time is a significant concern for both control and performance evaluation and are built separately in practice.

Several works have addressed previously the construction of detailed models of the controller ([1], [2]), plant elements ([3], [4]) or the complete closed-loop ([5]) with different classes of DES -Discrete Event Systems- formalisms. However, few results on the comparison of detailed and abstract models have been published, to the best of our knowledge. Consistency of the models of a component is a crucial concern, however, to avoid for instance that the duration of an operation in an abstract model be different of that obtained from the analysis of the corresponding detailed model. Moreover, the analysis of the internal behavior of a component requires obviously its detailed model but very often abstract models of the components that send and receive parts to/from this component or communicate with it.

[1]LURPA, École Normale Supérieure de Cachan, F-94200 Cachan, {perin,faure} at lurpa.ens-cachan.fr

[2]Institut Supérieur de Mécanique de Paris, F-93400 Saint-Ouen, faure at supmeca.fr

The aim of this work (Figure 1) is to contribute to bridge the gap between these two points of view. More precisely, an equivalence relation to compare the external behaviors of the detailed and abstract models of the same component will be defined. Two cases will be considered: the two models are fully deterministic or the models (or at least one of them) include non-determinism on time or values of some variables, e.g. the duration of an operation is not constant but stands between a lower and upper bound.

This paper is structured as follows. The formalism that has been selected to build the detailed and abstract models is presented in the next section. Section III focuses on the definition of an equivalence relation to compare these models assuming that they are deterministic; this definition and comparison methods which are based on it are illustrated in section IV. The case of non-deterministic models is addressed in section V and concluding remarks as well as prospects for further work are given in the last section.

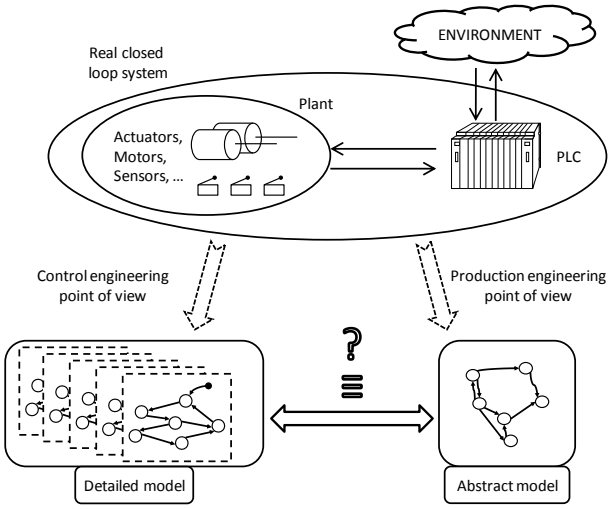


Fig. 1. Aim of the work

II. Formalism used

The models used in this paper will be described by using the formalism of Timed Automata with Discrete Data (TADD) fully described in [6]. This formalism is an extension of the timed automata formalism presented in [7] and is particularly suitable to model urgency and discrete variables evolutions in time. As detailed in [4] and [5] indeed, building meaningful timed models of plants or closed-loop systems requires to select a formalism that is able to model urgency.

Only the elements of the semantics of TADD that have been used in this work are described below for room saving.

A network of TADD is a set of automata $A_i^{\text{TADD}} = (L_i, l_i^0, V, C, E_i, I_i)$ with $i \in 1, \ldots, n, n \in \mathbb{N}^*$; the sets of clocks ($C$) and variables ($V = V_B \cup V_Z$) -composed of Boolean variables (V_B) and integer variables (V_Z)- are shared by all automata. The network thus is $\{ A_i^{\text{TADD}} \} = (\bar{L}, \bar{l}^0, V, C, E, I)$ with the following definitions:

- $\bar{L} = (L_1 \times L_2 \cdots \times L_n)$ is the set of locations,
- $\bar{l} = (l_1, l_2, \ldots, l_n)$ is the active locations vector,
- $\bar{l}^0 = (l_1^0, l_2^0, \ldots, l_n^0)$ is the initial locations vector,
- V is a set of common discrete variables; let $v \in V$ be a variable, $\tilde{v}$ is the variable value at a given time -either $\tilde{v} \in \{\text{true}, \text{false}\}$ if $v \in V_B$ or $\tilde{v} \in \mathbb{Z}$ if $v \in V_Z$- and $\tilde{V}$ is the set of all the variables values,
- C is a set of common clocks, let $c \in C$ be a clock, $\tilde{c} \in [0, +\infty)$ is the clock value at a given time and $\tilde{C}$ is the set of all the clocks values,
- $E = \bigcup_i E_i$ is the set of the edges of the network. Each edge $e = (l, g, t, u, a, 2^C, l')$ is going from a source location $l \in L_i$ to a target location $l' \in L_i$, with a guard g as a Boolean expression over the set of variables V, a timed constraint t over the set of clocks C, and an action a setting the new values of the variables of V. The term 2^C is introduced for the reset of clocks: each clock may be reset or not. $r \subseteq C$ is defined as the set of the clocks to be reset in a specific edge. $u \in \{true, false\}$ is the *urgent attribute* that defines the edge as an *urgent edge*; no time constraint is allowed (i.e. t = true, the always satisfied time constraint) when the urgent attribute is true,
- $I(\bar{l}) = \bigwedge_j I_j(l_i)$ is defined as the common invariant function over the set of clocks C. An invariant may indeed be associated to every location.

A network of TADD may evolve according to two concurrent semantics:

- **The edge firing semantics** is possible for each edge if the source location is active, the guard is *True* (i.e. the variables values of $\tilde{V}$ satisfy the guard), the time constraint is satisfied (i.e. $\tilde{C}$ satisfies the time constraint) and the invariant of the target location is satisfied by the new clocks values $\tilde{C}'$ including the clocks resets of the edge. The firing changes the active location from the source location to the target one, the set $\tilde{V}'$ of the new variables values is deduced from the assignment action a of the edge. Clocks values remain unchanged except for clocks that belong to r.
- With **the time evolution semantics**, only the value of every clock is increased by the same amount. The active location of every automaton of the network is unchanged and $\tilde{C}$ becomes $\tilde{C}'$ iff:
  - The values $\tilde{C}$ and $\tilde{C}'$ satisfy the invariant of the current active locations,
  - *No urgent edge may be fired from the current active locations*.

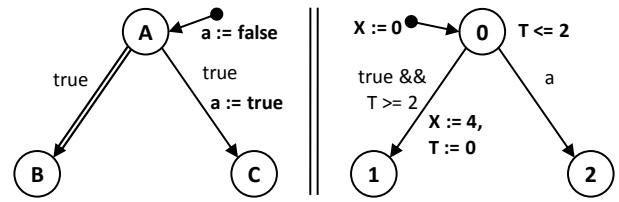


Fig. 2. A Network of Timed Automata with Discrete Data (TADD)

These two semantics are exemplified on the network of TADD given in figure 2. This figure depicts a network of two TADD using one clock T, one Boolean variable a, one integer variable X, and composed of:

- Locations ($\{A, B, C\} \times \{0, 1, 2\}$), which are represented by circles, location names being in bold font.
- The initial locations (here, locations A and 0), which are represented with a source edge with the initialization of variables (clocks are always initialized to zero).
- Locations invariants (**T <= 2** for location 0 in this example), in bold font.
- Edges, which are represented by arrows, the urgent ones using double line arrows (like from location A to B).
- Guards and time constraints, in normal font. The guard and time constraint of an edge may be combined by a disjunction or conjunction operator, respectively noted || and && (true && T >= 2, for example). The Boolean operator NOT will be noted !.
- Actions on variables, which will be represented by the assignment operator **:=** in bold font; actions are separated by comas when several actions are associated to an edge. Clock resets will follow the same pattern, the assignment being limited to reset (assignment to zero), like in the edge from location 0 to location 1.

From the initial state (locations A and 0 are active, variables and clocks have their initial values), two evolutions are possible:

- **(A,0)** $\rightarrow$ **(B,0)**, as the source locations vector is active and the guard (true) is always satisfied. The next active locations vector will be (B,0), and the values of both variables and clocks remain unchanged (no assignment and no clock reset on this edge),
- **(A,0)** $\rightarrow$ **(C,0)**, as the source locations vector is active and the guard is satisfied. The next active locations vector will be (C,0), and the values of X and T remain unchanged but the Boolean variable a will become true.

No evolution of the right model is possible because the guard of the edge $0 \rightarrow 2$ is not satisfied and the time constraint of the edge $0 \rightarrow 1$ cannot be satisfied as the edge

A $\rightarrow$ B may be fired and is urgent (thus preventing any evolution based on the time semantics).

If the edge A $\rightarrow$ C is fired, two other evolutions are thus possible:

- **(C,0)** $\rightarrow$ **(C,1)**, by using the time evolution semantics to increase the value of the clock T from 0 to 2, then by firing the edge 0 $\rightarrow$ 1. The next active locations vector will be (C,1) and the value of Boolean variable a remains unchanged (true), the variable X is set to 4 and the clock T is reset,
- **(C,0)** $\rightarrow$ **(C,2)**, as a is true and the guard is hence satisfied. The new active locations vector will be (C,2) and all the values remain unchanged (neither variable assignment nor clock reset on the fired edge).

In the remainder of this paper, neither the guards and time constraints which are always true, nor the initialization of variables to false or 0 will be represented for readability reasons.

Last, a sequence -or run- of a TADD is defined in [6]. This definition may be extended to a network of TADD $(\bar{L}, \bar{l}^0, V, C, E, I)$ as:

$$Seq = s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} s_3 \xrightarrow{\delta_3} s_4 \ldots s_j \qquad (1)$$

with :

- $s_j = (\bar{l}_j, \tilde{V}_j, \tilde{C}_j) \in S = \bar{L} \times \tilde{V} \times \tilde{C}$ is a state of the network of TADD defined by the active location vector and the values of the variables and clocks.
- For each sequence index $j \in \mathbb{N}$, δ_j is a change of state that may be:
  - Either $e_j \in E$, the evolution through the edge e_j,
  - Or $d_j \in \mathbb{N}^*$, clock increase of duration d_j such as $\tilde{C}_{(j+1)} = \tilde{C}_j + d_j$.

Figure 3 represents a sequence of the example network of figure 2.

$$Seq = \begin{array}{c} A \\ 0 \\ \hline F \\ \hline 0 \\ \hline 0 \end{array} \xrightarrow{A \rightarrow C} \begin{array}{c} C \\ 0 \\ \hline T \\ \hline 0 \\ \hline 0 \end{array} \xrightarrow{2} \begin{array}{c} C \\ 0 \\ \hline T \\ \hline 0 \\ \hline 2 \end{array} \xrightarrow{0 \rightarrow 1} \begin{array}{c} C \\ 1 \\ \hline T \\ \hline 4 \\ \hline 0 \end{array} \ldots \begin{array}{c} \tilde{l}_j \\ \hline \tilde{a}_j \\ \hline \tilde{X}_j \\ \hline \tilde{T}_j \end{array}$$

Fig. 3. Example of sequence

III. Equivalence relation definition

A. Choice of the equivalence

Numerous equivalence relations that can be used to compare two formal models have been previously defined; a good synthesis of these works is presented in [8]. Globally, these relations can be put in two categories: relations based on the languages recognized or marked by a model, like bisimulation, weak simulation, ready simulation and relations based on the responses of a model to input sequences, which are often named trace equivalences. The equivalence relations of the first category cannot be selected for this study whose objective is to compare a detailed and an abstract model of a same physical device because the detailed model includes usually many more states and transitions than the abstract one then the languages of the two models are surely far different. At the opposite, an equivalence relation based on the input/output traces may permit the comparison.

It matters first to define what are the input and output variables of a TADD:

- **The input variables**, $V_{\mathrm{i}} \subseteq V$, of a given TADD are the variables which are used in the guards of its edges and are assigned in another model; an *input sequence* represents the evolutions of the values of these variables during time,
- **The output variables**, $V_{\mathrm{o}} \subseteq V$, of a given TADD are the variables which are assigned in the guards of these edges and read then used by another model; an *output sequence* represents the evolutions of the values of these variables during time.

The sets of input and output variables of the detailed and corresponding abstract models are generally not identical. Hence, comparison of the behaviors of the two models will require first to find the sets of common variables then to check whether the following trace equivalence relation holds or not:

For each input trace built on the common input variables set, the output trace on the common output variables set produced by the two models is identical.

It must be noted that this definition of an equivalence relation assumes that the two models are deterministic because any model must be trace equivalent to itself. This assumption will be kept in this section and the next one but removed in section V.

B. Building a timed trace for TADD comparison

The selected equivalence relation implies that traces be built from the formal models to compare. The aim of this subsection is to show how a *timed trace* can be built from a sequence then to propose a formal definition of the trace that will be used to compare two TADD.

Figure 5 is an example of sequence built from the TADD of Figure 4. For each state of this sequence, the active location, the values of the output variables a, b, c, x, y, z and the clock are respectively given.

In this sequence:

- The values of all output variables are pointed out in each state,
- The clock may be reset between two states, as this is the case for the change L3 $\longrightarrow$ L3 from the 6thto the 7thstate in Figure 5.

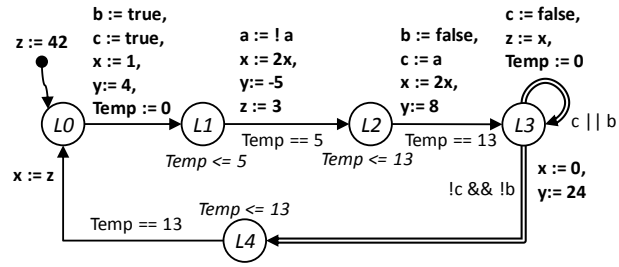


Fig. 4. TADD example with Boolean variables a,b and c, discrete variables x,y and z and a clock Temp.

$$
Seq = \begin{array}{c} L0 \\ \hline F \\ F \\ F \\ 0 \\ 0 \\ 42 \\ \hline 0 \end{array}
\xrightarrow{L0\to L1}
\begin{array}{c} L1 \\ \hline F \\ T \\ T \\ 1 \\ 4 \\ 42 \\ \hline 0 \end{array}
\xrightarrow{5}
\begin{array}{c} L1 \\ \hline F \\ T \\ T \\ 1 \\ 4 \\ 42 \\ \hline 5 \end{array}
\xrightarrow{L1\to L2}
\begin{array}{c} L2 \\ \hline T \\ T \\ T \\ 2 \\ -5 \\ 3 \\ \hline 5 \end{array}
\xrightarrow{8}
\begin{array}{c} L2 \\ \hline T \\ T \\ T \\ 2 \\ -5 \\ 3 \\ \hline 13 \end{array}
$$

$$
\xrightarrow{L2\to L3}
\begin{array}{c} L3 \\ \hline T \\ F \\ T \\ 4 \\ 8 \\ 3 \\ \hline 13 \end{array}
\xrightarrow{L3\to L3}
\begin{array}{c} L3 \\ \hline T \\ F \\ F \\ 4 \\ 8 \\ 8 \\ \hline 0 \end{array}
\xrightarrow{L3\to L4}
\begin{array}{c} L4 \\ \hline T \\ F \\ F \\ 0 \\ 24 \\ 8 \\ \hline 0 \end{array}
\xrightarrow{13}
\begin{array}{c} L4 \\ \hline T \\ F \\ F \\ 0 \\ 24 \\ 8 \\ \hline 13 \end{array}
\cdots
\begin{array}{c} \tilde{l}_j \\ \hline \tilde{a}_j \\ \tilde{b}_j \\ \tilde{c}_j \\ \tilde{x}_j \\ \tilde{y}_j \\ \tilde{z}_j \\ \hline Temp_j \end{array}
$$

Fig. 5. Example of sequence for the TADD of figure 4

To permit that two TADD be compared, only the values of the output variables which belong to the set W of the output variables common to the two TADD ($W \subseteq V_{\mathrm{o}}$) must be kept and a unique clock, reset only at the initialization must be introduced, as proposed by [7].

A *timed trace* over the subset of variables W can then be defined for the automaton (L, l_0, V, C, E, I) by:

$$
\begin{aligned}
Tra(W) = & (\tilde{W}_0, 0) \xrightarrow{\alpha_0} (\tilde{W}_1, t_1) \xrightarrow{\alpha_1} (\tilde{W}_2, t_2) \xrightarrow{\alpha_2} \\
& (\tilde{W}_3, t_3) \xrightarrow{\alpha_3} (\tilde{W}_4, t_4) \ldots (\tilde{W}_j, t_j)
\end{aligned} \tag{2}
$$

with :

- t_j, absolute date (value of the unique clock reset only at the initialization of the trace)
- $\tilde{W}_j$, values of variables from the variable subset $W \subseteq V$ at absolute date t_j
- For each trace index $j \in \mathbb{N}$, α_j is a change of the trace:
  - Either ϵ_j, an action on variables such that $\tilde{W}_j \overset{\epsilon_i}{\mapsto} \tilde{W}_{(j+1)}$ and $t_{(j+1)} = t_j$.
  - Or $d_j \in \mathbb{N}$, a time consumption such that $t_{(j+1)} = t_j + d_j$ and $\tilde{W}_{(j+1)} = \tilde{W}_j$.

The timed trace obtained from the sequence of Figure 5 is given at Figure 6, by assuming that $W = \{a, b, x, y\}$. It must be underlined that *the last value for each trace index is the value of the absolute date and not of the clock Temp*.

$$
Tra(W) = \begin{array}{c} F \\ F \\ 0 \\ 0 \\ \hline 0 \end{array}
\xrightarrow{L0\to L1}
\begin{array}{c} F \\ T \\ 1 \\ 4 \\ \hline 0 \end{array}
\xrightarrow{5}
\begin{array}{c} F \\ T \\ 1 \\ 4 \\ \hline 5 \end{array}
\xrightarrow{L1\to L2}
\begin{array}{c} T \\ T \\ 2 \\ -5 \\ \hline 5 \end{array}
\xrightarrow{8}
\begin{array}{c} T \\ T \\ 2 \\ -5 \\ \hline 13 \end{array}
$$

$$
\xrightarrow{L2\to L3}
\begin{array}{c} T \\ F \\ 4 \\ 8 \\ \hline 13 \end{array}
\xrightarrow{L3\to L4}
\begin{array}{c} T \\ F \\ 0 \\ 24 \\ \hline 13 \end{array}
\xrightarrow{13}
\begin{array}{c} T \\ F \\ 0 \\ 24 \\ \hline 26 \end{array}
\ldots
\begin{array}{c} \tilde{a}_j \\ \tilde{b}_j \\ \tilde{x}_j \\ \tilde{y}_j \\ \hline \tilde{t}_j \end{array}
$$

Fig. 6. *Timed trace* obtained from the sequence of figure 5

C. Trace equivalence definition

Let two TADD $(L^{\mathrm{A}}, l_0^{\mathrm{A}}, V^{\mathrm{A}}, C^{\mathrm{A}}, E^{\mathrm{A}}, I^{\mathrm{A}})$ and $(L^{\mathrm{B}}, l_0^{\mathrm{B}}, V^{\mathrm{B}}, C^{\mathrm{B}}, E^{\mathrm{B}}, I^{\mathrm{B}})$ to be compared according to a timed trace equivalence relation.

Some additional notations are to be introduced to define formally this equivalence relation:

- $V_{\mathrm{i}}^{\mathrm{A}}$ ($V_{\mathrm{i}}^{\mathrm{B}}$) is the set of input variables of TADD A (TADD B),
- $V_i = V_{\mathrm{i}}^{\mathrm{A}} \cap V_{\mathrm{i}}^{\mathrm{B}}$ is the subset of input variables common to TADD A and B,
- $V_{\mathrm{o}}^{\mathrm{A}}$ ($V_{\mathrm{o}}^{\mathrm{B}}$) is the set of output variables of TADD A (TADD B),
- $V_o = V_{\mathrm{o}}^{\mathrm{A}} \cap V_{\mathrm{o}}^{\mathrm{B}}$ is the subset of output variables common to TADD A and B.

Output timed traces $Tra^A(V_o)$ and $Tra^B(V_o)$) can be defined for the two TADD as stated by 2. These traces are obtained when a timed input trace $Tra(V_i)$ is sent to both models. This trace specifies the values of the common inputs for different values of the absolute clock; an example of such a trace for two common inputs: IN1 (Boolean variable) and IN2 (discrete variable) is given at Figure 7.

$$
Tra(V_i) = \begin{array}{c} T \\ 0 \\ \hline 0 \end{array}
\xrightarrow{\epsilon_1}
\begin{array}{c} T \\ 4 \\ \hline 0 \end{array}
\xrightarrow{5}
\begin{array}{c} T \\ 4 \\ \hline 5 \end{array}
\xrightarrow{\epsilon_2}
\begin{array}{c} F \\ 0 \\ \hline 5 \end{array}
\xrightarrow{3}
\begin{array}{c} T \\ 4 \\ \hline 8 \end{array}
\xrightarrow{\epsilon_3}
$$

$$
\begin{array}{c} T \\ 0 \\ \hline 8 \end{array}
\xrightarrow{\epsilon_4}
\begin{array}{c} T \\ -5 \\ \hline 8 \end{array}
\xrightarrow{2}
\begin{array}{c} T \\ -5 \\ \hline 10 \end{array}
\ldots
\begin{array}{c} I\tilde{N}1_j \\ I\tilde{N}2_j \\ \hline \tilde{t}_j \end{array}
$$

Fig. 7. Example of input timed trace

The set of all output traces obtained from the model A (respectively B), for a set $Traces(V_i)$ of input traces $Tra(V_i)$ will be noted $Traces^A(V_o)$ (resp. $Traces^B(V_o)$).

With these notations, the trace equivalence relation between two TADD A and B is:

$$
\begin{aligned}
B \equiv A \Rightarrow & \\
& \forall Tra(V_i), \forall j \in \mathbb{N}, \\
& \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{A}}, t_{\mathrm{j}}^{\mathrm{A}}) \in Tra^{A}(V_o) \text{ and } \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{B}}, t_{\mathrm{j}}^{\mathrm{B}}) \in Tra^{B}(V_o), \\
& (\tilde{V}_{\mathrm{j}}^{\mathrm{A}}, t_{\mathrm{j}}^{\mathrm{A}}) = (\tilde{V}_{\mathrm{j}}^{\mathrm{B}}, t_{\mathrm{j}}^{\mathrm{B}})
\end{aligned} \tag{3}
$$

The definition 3 means that two timed models A and B are said trace equivalent iff:

- Whatever the input trace applied to the two models and for each index of this trace (first line),
- For every couple of elements of the output traces of A and B (second line),
- The values of the common output variables and the time stamp which compose these two elements are equal (last line).

IV. Illustration

A. Presentation of the example

The example used in this paper is a pneumatic manipulator of a testing station depicted at Figure 8. The plant is composed of two pneumatic actuators and their associated sensors (2 for the vertical cylinder and 3 for the horizontal cylinder) and one suction cup linked to a vacuum pump. This plant is controlled by a logic controller which is not represented on this figure where only its inputs and outputs are listed. The testing station includes also a testing device which checks the correctness of mechanical parts.

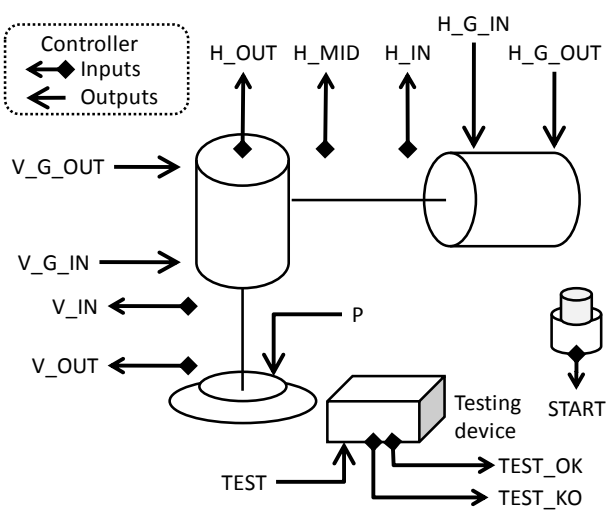


Fig. 8. Testing station layout

The expected behavior of this manipulator is the following:

1) When the START button is pushed, the READY variable -true when the manipulator is available, false otherwise- is reset, the suction cup goes down and picks a part (it is assumed that a part is present at the picking place -rightmost position- when the START button is pushed).
2) The part is lifted, moved to the middle position of the horizontal axis then lowered to the testing device.
3) This device checks the part and sends to the controller the result of the test (TEST_OK or TEST_KO).
4) If the *test was passed*, the suction is stopped and the part is transferred to another station by a conveyor which is not considered in this example. The suction cup comes back to its initial position (move up, then move to the right) and READY variable is set. If *the test failed*, the part is brought to the leftmost position (move up, then move to the left) and is released into a bin of non-conform parts. Then the suction cup comes back to its initial position and READY variable is set.

B. Detailed model of the manipulator

This model describes the internal behavior of the closed-loop system composed of the plant elements and the controller. Therefore, a network of five TADD, two (H_act and V_act) for the actuators, two (H_sen and V_sen) for the sets of sensors of the horizontal and vertical axes and one for the controller has been built.

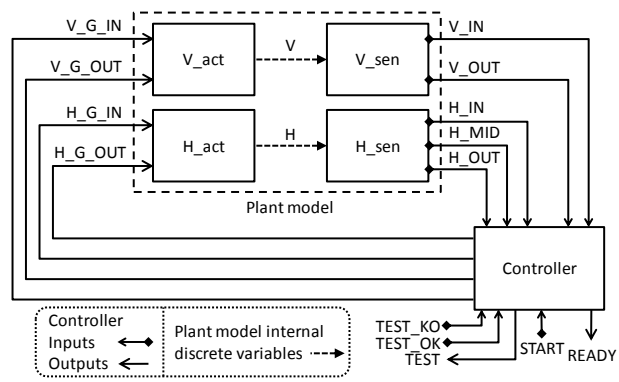
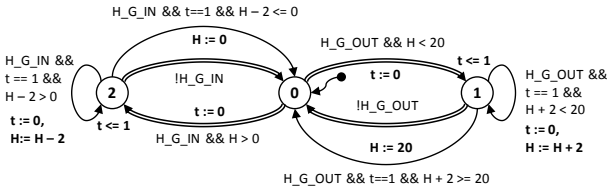


Fig. 9. Structure of the detailed model of the example with controller Inputs/Ouputs and internal variables of the plant model (H: length of rod out of the horizontal cylinder, V: same for the vertical cylinder)

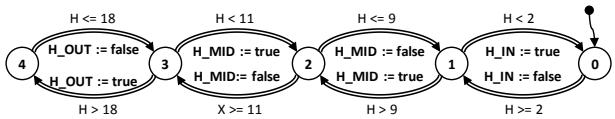
The structure of this network is depicted at Figure 9 and examples of actuator model and sensors model are given on Figure 10. As the aim of this paper is not to discuss the construction of these models, no further explanation will be given on this topic; the interested reader is referred to [5] for an accurate presentation.

C. Abstract model of the manipulator

This model (Figure 11) does not consider the internal states of the closed-loop elements (positions of the rods of the cylinders, states of the sensors, values of the inputs/outputs of the controller) but describes the external behavior of the manipulator. When START is pushed, the part is transferred to the testing device in 15 time units then a new part can be tested after 17 (test passed) or 25 (test failed)



(a) Model of the horizontal pneumatic cylinder



(b) Model of the associated sensors

Fig. 10. Models associated to the horizontal movement of the manipulator (actuator and sensors)

time units; hence, the cycle time of this manipulator is 32 or 40 time units. This abstract model is well appropriate to evaluate time performances, a crucial concern for production management.

Last, it must be clearly underlined that this abstract model is fully deterministic because:

- the time constraint of the edge L1 $\rightarrow$ L2 (resp. L3 $\rightarrow$ L0 and L4 $\rightarrow$ L0) is equal to the upper bound of the invariant of the location source of this edge; therefore, an evolution from this location is only possible when this bound is met.
- the guards of the two edges that start from L2 are exclusive.

The duration of one cycle is equal to 32 time units if the test is passed (TEST_OK true and TEST_KO false) or 40 time units if the test failed (TEST_KO true).

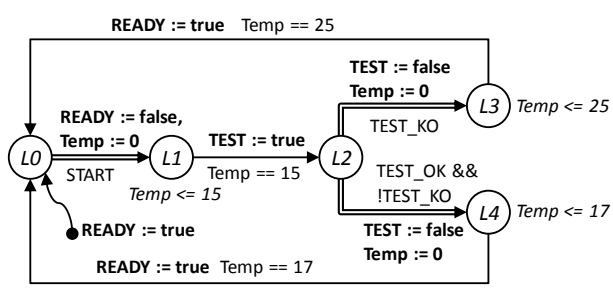


Fig. 11. Abstract model of the manipulator

D. Verification of the equivalence between the two models

To verify whether the two models are trace equivalent, according to 3, the set of common input and output variables must be defined. In the case of this example, these sets are respectively $V_i = \{START, TEST_OK, TEST_KO\}$ and $V_o = \{TEST, READY\}$.

Verification of the equivalence relation with these definitions of V_i and V_o must be tool-supported. The well-known and efficient tool UPPAAL ([9], [10]) has been selected for this study. Hence, the TADD models are to be translated into UPPAAL models according three simple translation rules that guarantee that the semantics is preserved:

- the locations are directly translated, with their invariants.
- a guard of a UPPAAL edge is the conjunction of the guard and time constraint of the corresponding TADD edge.
- an urgent UPPAAL communication channel is defined for the whole model; every urgent edge is synchronized with this channel, what is not the case for the non-urgent edges ([10]).

Once the UPPAAL models obtained, two solutions can be considered to verify the trace equivalence of the two models:

- by using the simulation capabilities of the tool. The same input trace is sent to the two models and the output traces that they produce are compared. This simulation is repeated for a set of representative input traces.
- by model-checking, a formal verification technique. In that case, a formal property constructed from (3) is checked whatever the input trace could be.

The first solution permits to verify the equivalence for typical cases but does not guarantee the completeness of the analysis. The second one is more general but requires that an observer be built to express correctly the formal property; construction of this observer is out of the scope of this paper but a complete presentation of this topic can be found in [11].

The two solutions have been implemented for the models of the manipulator and have given consistent results; the detailed and abstract models are trace equivalent.

V. Trace equivalence with tolerances

The trace equivalence relation (3) can be applied only when the two models to compare are fully deterministic because it relies on a strict equality on the output values for every date of the trace. When one of the models, or both, includes non-determinism on time or values of some variables, new equivalence relations with tolerances on time or values are to be built. This issue will be illustrated on the model of Figure 12 where the sets of common input and output variables are the same than those of Figure 11, the duration of one cycle is always equal to 32 or 40 time units but depends no more on the test result but on a non-deterministic choice when firing the edge from L3 to L0. This modeling is more abstract than that of Figure 11 but can be helpful when focus is put only on evaluation of the bounds, and not the distribution, of time performances.

To permit comparison of two models with non-determinism on time, a tolerance on time ΔT must be first defined:

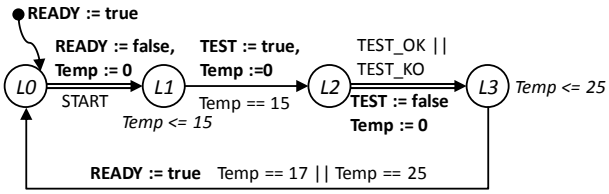


Fig. 12. Non-deterministic abstract model of the manipulator

$$\Delta T = [\Delta T_{inf}; \Delta T_{sup}], \quad \text{with } \Delta T_{inf}, \Delta T_{sup} \in \mathbb{Z} \text{ and } \Delta T_{inf} \leq \Delta T_{sup} \tag{4}$$

With this definition, a trace equivalence relation with tolerance on time can be defined from (3):

$$\begin{aligned} & B \cong_{\Delta T} A \Rightarrow \\ & \forall Tra(V_i), \forall j \in \mathbb{N}, \\ & \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{A}}, t_{\mathrm{j}}^{\mathrm{A}}) \in Tra^A(V_o) \text{ and } \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{B}}, t_{\mathrm{j}}^{\mathrm{B}}) \in Tra^B(V_o), \\ & \quad t_{\mathrm{j}}^{\mathrm{A}} - t_{\mathrm{j}}^{\mathrm{B}} \in \Delta T \text{ and} \\ & \quad \tilde{V}_{\mathrm{j}}^{\mathrm{B}} = \tilde{V}_{\mathrm{j}}^{\mathrm{A}} \end{aligned} \tag{5}$$

It must be underlined that the tolerance on time is defined on the global clock. On the other side, automated manufacturing systems are composed of numerous components whose behavior is cyclic. For this kind of components, this tolerance may become very large if the non-deterministic behavior of one model increases the time shift between the two models to compare at every cycle. A solution to this issue is to consider the two traces for only one cycle, i.e. to reset the global clock at the beginning of each cycle by using the value of the variable READY in the example for instance. Then, two trace equivalence relations with tolerance on time can be defined: one global where the clock is reset only at the initialization and one cyclic where the clock is reset at the beginning of each cycle.

If $\Delta T = [0, 8]$, the new abstract model and the detailed model of the manipulator are trace equivalent for a cyclic trace equivalence relation, because the variable READY is set 32 or 40 time units after it has been reset.

In a similar manner, tolerances on the values of the output variables $v \in V_o$ may be introduced. These tolerances can be defined only for the discrete variables because introducing a tolerance on the value on a Boolean variable does not make sense or with other words means that this variable does not matter.

$$\forall v \in V_o, \Delta V(v) = [\Delta V_{\mathrm{inf}}^{\mathrm{v}}; \Delta V_{\mathrm{sup}}^{\mathrm{v}}] \quad \text{with } \Delta V_{\mathrm{inf}}^{\mathrm{v}}, \Delta V_{\mathrm{sup}}^{\mathrm{v}} \in \mathbb{Z} \text{ and } \Delta V_{\mathrm{inf}}^{\mathrm{v}} \leq \Delta V_{\mathrm{sup}}^{\mathrm{v}} \tag{6}$$

With this definition, a trace equivalence relation with tolerance on the values of the output variables can be defined from (3):

$$\begin{aligned} & B \cong_{\Delta V} A \Rightarrow \\ & \forall Tra(V_i), \forall j \in \mathbb{N}, \\ & \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{A}}, t_{\mathrm{j}}^{\mathrm{A}}) \in Tra^A(V_o) \text{ and } \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{B}}, t_{\mathrm{j}}^{\mathrm{B}}) \in Tra^B(V_o), \\ & \quad t_{\mathrm{j}}^{\mathrm{A}} = t_{\mathrm{j}}^{\mathrm{B}} \text{ and} \\ & \quad \forall v \in V_o, \tilde{v}_{\mathrm{j}}^{\mathrm{A}} - \tilde{v}_{\mathrm{j}}^{\mathrm{B}} \in \Delta V(v) \end{aligned} \tag{7}$$

With $\tilde{v}_{\mathrm{j}}^{\mathrm{A}}$ the value of variable v of automaton A at trace index j and $\tilde{v}_{\mathrm{j}}^{\mathrm{B}}$ the same for automaton B.

By combining (5) and (7), a general equivalence relation with tolerances (8) can be set up:

$$\begin{aligned} & B \cong_{\Delta V, \Delta T} A \Rightarrow \\ & \forall Tra(V_i), \forall j \in \mathbb{N}, \\ & \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{A}}, t_{\mathrm{j}}^{\mathrm{A}}) \in Tra^A(V_o) \text{ and } \forall (\tilde{V}_{\mathrm{j}}^{\mathrm{B}}, t_{\mathrm{j}}^{\mathrm{B}}) \in Tra^B(V_o), \\ & \quad t_{\mathrm{j}}^{\mathrm{A}} - t_{\mathrm{j}}^{\mathrm{B}} \in \Delta T \text{ and} \\ & \quad \forall v \in V_o, \tilde{v}_{\mathrm{j}}^{\mathrm{A}} - \tilde{v}_{\mathrm{j}}^{\mathrm{B}} \in \Delta V(v) \end{aligned} \tag{8}$$

This relation states that two models are trace equivalent iff, whatever the input trace, the values of the clock and the output variables that they deliver stand in a given interval, for each trace index.

VI. Conclusion

This paper has shown that two timed models of the same component of an automated discrete manufacturing system that have been built with different objectives, a detailed model developed to design the control of this component and a more abstract model for production management purposes, can be compared on the basis of an equivalence relation defined on the input/output traces that they produce. Comparison of the models is then possible by using a tool for formal analysis of timed discrete models. The equivalence relation has been afterwards extended by introducing tolerances on time and values of the output variables so as to deal with models that may include non-determinism on time and values.

On-going work focuses on improvement of the applicability of the approach by developing a method to construct automatically observers and formal properties from the equivalence relations that have been proposed.

References

[1] M. Lindahl, P. Pettersson, and W. Yi, "Formal design and analysis of a gear controller: an industrial case study using uppaal," *LNCS, Proc. of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1384, pp. 281–297, 1998.

[2] V. Vyatkin and H.-M. Hanisch, "Practice of modeling and verification of distributed controllers using signal net systems," in *Proc. of International Workshop on Concurrency, Specification and Programming (CS&P)*, Berlin, Germany, October 2000.

[3] B. Rohée, B. Riera, V. Carré-Ménétrier, and J.-M. Roussel, "A methodology to design and check a plant model," in *Proc. of the 3rd IFAC Workshop on Discrete-Event System Design (DESDes'06)*, Rydzyna, Pologne, Jun. 2006, pp. 246–250.

[4] M. Perin and J.-M. Faure, "Building meaningful timed plant models for verification purposes," in *Proc. of the 13th IFAC Symposium on information control problems in manufacturing, INCOM'09*, Moscow, Russia, 2009, pp. 970–975.

[5] ——, "Building meaningful timed models of closed-loop DES for verification purposes," *Control Engineering Practice"*, emphpending for publication with doi: 10.1016/j.conengprac.2012.05.002.

[6] A. Janowska and P. Janowski, "Slicing of timed automata with discrete data," *Fundamenta Informaticae*, vol. 72, no. 1-3, pp. 181–195, 2006.

[7] R. Alur, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[8] R. van Glabbeek, "The linear time-branching time spectrum i - the semantics of concrete, sequential processes," in *Handbook of Process Algebra, chapter 1*, 2001, pp. 3–99.

[9] UPPAAL website, "http://www.uppaal.com," 2011.

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

[11] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.