Building Meaningful Timed Models of Closed-loop DES for Verification Purposes

Matthieu Perin, Jean-Marc Faure

Building Meaningful Timed Models of Closed-loop DES for Verification Purposes

Matthieu Perin[a], Jean-Marc Faure[a,b]

[a] *Laboratoire Universitaire de Recherche en Production Automatisée, École Normale Supérieure de Cachan, F-94230 Cachan (e-mail: {perin, faure}@lurpa.ens-cachan.fr)*
[b] *Institut Supérieur de Mécanique de Paris, F-93400 Saint-Ouen (e-mail: jean-marc.faure@supmeca.fr)*

Abstract

Formal verification methods require that a model of the system to analyze, in the form of a network of automata for instance, be built previously. Every evolution of this formal model must represent a real evolution of the modeled system; if the model contains indeed spurious evolutions, meaningless states, which do not correspond to physically possible states, can be reached and the verification results are surely not trustworthy. This paper focuses on construction of the formal model of a closed-loop system which can be represented as a Discrete Event System (DES) and where all evolutions and states are meaningful wrt to the real system behavior. A closed-loop system is composed of a physical system to control, named plant, and a controller. A modular approach to build the plant model is presented in the first part of the paper; to prevent from meaningless evolutions and states in this model, a solution based on the concept of urgent edges is proposed and exemplified. Then, construction of the formal model of the closed-loop system is addressed; it is shown that restriction of the evolutions of this model to the only meaningful ones can be easily achieved by introducing variables that represent the modification of the inputs of the logic controller and the stability condition of the control specification.

Keywords:

Formal methods, Plant model, Concurrent evolutions, Urgency, GRAFCET[1], Model-checking

1. Introduction

Programmable Logic Controllers (PLCs) are widely used in automation systems. These components execute software that implement the control specifications; to ensure dependability during operation of the automation system, it really matters to check whether the specifications, as well as their implementation if no automatic certified code generator is used, are conform to what is expected. Formal methods, based on sound, well-defined models, and in particular formal verification methods by model-checking (Bérard et al. (2001)), are promising solutions to meet this objective, as pointed out in Johnson (2007).

[1] Acronym from the French translation of Step Transition Control Graph

Numerous valuable results have been published in the domain of model-checking of PLCs software and it is not possible to give an exhaustive references list; however, it can be mentioned that the formalisms used in these works are mainly transition systems (Bauer et al. (2004); Gourcuff et al. (2008); Schlich et al. (2009); Pavlovic and Ehrich (2010) for instance), NCES (Net Condition/Event Systems) or TNCES (Timed NCES) (Hanisch et al. (1997); Vyatkin and Hanisch (1999); Lobov et al. (2005a,b); Hanisch et al. (2006)), or timed automata (Bauer et al. (2004); Behrmann et al. (2004), for instance).

According to Frey and Litz (2000), three approaches can be defined for model-checking of PLCs:

- Basic model-checking consists of an exhaustive analysis of the controller model, written in formal language, in order to prove that some properties hold on this model. This method gives the strongest results when verification of safety properties is addressed, because no assumption on the behavior of the environment is introduced, but may provide no answer for liveness properties, as shown in Machado et al. (2006).
- Constraints based model-checking still uses a controller model described in a formal language but adds some assumptions (over the input / output variables for example) in order to reduce the possibility of evolutions of the analyzed model. This approach provides less strong results, because the results are correct iff the assumptions are true, but permits to obtain answers for some simple liveness properties.
- The model-based model-checking approach uses, in addition to the controller model, a plant model also described in a formal language. This method always provides results for liveness properties.

A detailed comparison of the first and third approaches can be found in Machado et al. (2006). As the first concern of controller designers is to be sure that the controller is able to command the plant as required, verification of all liveness properties is mandatory. This explains why this paper focuses only on model-based model-checking.

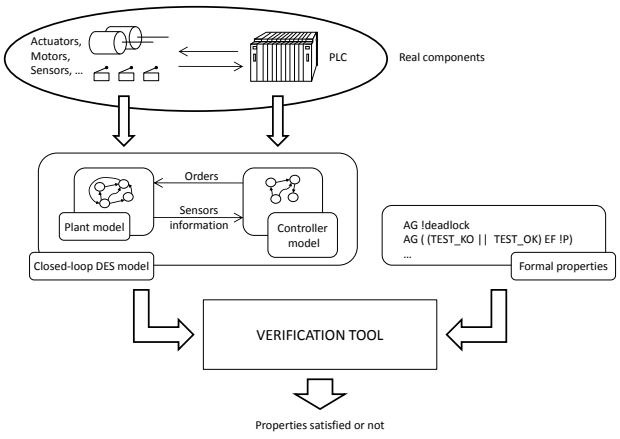


Figure 1: Model-based model-checking principle.

Figure 1 describes the principle of model-based model-checking applied to a closed-loop discrete event system (DES) where:

- The real closed-loop DES is composed of a PLC connected to plant components. Two models: plant model and controller model, are then to be built.
- The formal models of both plant and controller are inputs, with one or several formal properties, of a model-checker.
- This verification tool provides back a Boolean answer: the properties are satisfied (or not) on the model.

In order to obtain trustworthy results, the properties and the models given to the model-checker have to be meaningful. The construction of meaningful formal properties will not be addressed in this paper; the interested reader is referred to Campos and Machado (2009) for further information on this topic. At the opposite, this work focuses on the construction of meaningful models, i.e. models where every evolution represents a change of state in the real world; a meaningful plant model for instance must contain only evolutions which are physically feasible. A first consequence of this objective is that timed models must be considered to model correctly the behavior of the closed-loop system where some evolutions consume time.

Model-based model-checking of timed models has been addressed previously in Lobov et al. (2005a,b); Hanisch et al. (2006); Bel Mokadem et al. (2010). The novelty of the work presented in this paper comes from the following three features:

- The plant model is built from generic models of plant components (sensors, actuators). This choice is close to the industrial practice in CAD[2] where specific systems are often constructed by assembling instances of libraries components and is mandatory to obtain industrial acceptance of model-based model-checking. This solution was not selected in the latter reference where only a case study is presented and no methodology to build the plant model is proposed. This is not the case for the first three references where such a methodology is presented by using three UML[3] diagrams (class diagrams, state-charts and collaboration diagrams); this approach is surely valuable. However, it has been considered in this work, on the basis of previous research works in cooperation with companies, that lots of automation engineers are not familiar with UML diagrams and that the use of three kinds of diagrams may provoke difficulties during design of the generic models. This is the reason why only one formalism (Timed Automata with Discrete Data) has been selected to represent these models in this study. Combination of the two approaches does not seem to be a too difficult issue, however.
- The formal model of the controller is obtained from a specification model and not from an implementation model, in a PLC programming language, like IL (Instruction List), LD (Ladder Diagram), FBD (Function Block Diagram) or ST (Structured Text). This is a significant difference from the previous works where the controller model depends on the programming language. The reason of this choice is that detection of errors in the

[2] Computer-Aided Design
[3] Unified Modelling Language

specification is less expensive than detection of errors in the implementation; model-checking of specifications must then be privileged. Moreover, it is well-suited to the industrial practice to avoid manual code generation but to generate automatically the code from the specification (model-based approach) with a certified/qualified automatic code generator like SCADE [4]; such practice is now commonly used for critical systems. It remains interesting even if no automatic code generator is used, for cost reasons (the sooner in the life-cycle errors are detected, the cheaper errors detection is).

- Last, both models (plant and controller models) are developed in a formalism, named Timed Automata with Discrete Data (TADD) that permits to build realistic models because its semantics contains powerful mechanisms to model urgency. Generally speaking, urgency is a concept which means that time evolution is stopped. This permits to give priority to non-timed evolutions over timed evolutions, then to remove unexpected, for non-realistic, concurrencies, while keeping realistic concurrencies between non-timed evolutions. Hence, construction of modular meaningful models is facilitated. The semantics of TNCES does not seem to contain such selective urgency mechanisms. The concept of urgency is used in the fourth reference (Bel Mokadem et al. (2010)) to build the model of the closed-loop; however, this reference relies on a specific formalism: that of the UPPAAL[5] tool (Bengtsson et al. (1996)). At the opposite, the objective of this paper is to present a generic, tool-independent method that can be implemented in several timed model-checkers.

The paper starts by presenting the formalism used in this work to model both plant and controller. Then, an example of a closed-loop DES that will illustrate the contributions is described. Section 4 presents a modular approach to build the plant model from generic components models; to prevent from meaningless evolutions in this model, a solution based on the concept of urgent edges is proposed and exemplified. The next section is devoted to the construction of the controller model, assuming that the controller is specified in GRAFCET (IEC 60848 (2002)). Construction of the complete model of the closed-loop system is dealt with in Section 6; restriction of the evolutions of this model to the only meaningful ones is achieved by introducing variables that represent the modification of the inputs of the logic controller and the stability condition of the control specification. Section 7 focuses on verification of properties on closed-loop models with include faultless or faulty plant models. The last section sums up the results and gives perspectives for future work.

2. Timed Automata with Discrete Data

The models used in this paper will be described using the formalism of Timed Automata with Discrete Data (TADD) described in Janowska and Janowski (2006). This formalism is particularly suitable to model urgency and discrete variable evolutions in time. Moreover, it does not depend on a particular verification tool; the models that will be presented may thus be used with different tools provided that sound transformation rules has been defined.

[4] SCADE website: http://www.esterel-technologies.com/products/scade-suite/

[5] UPPAAL website: http://www.uppaal.org

The complete formalism is defined in Janowska and Janowski (2006) and is an extension of the timed automata formalism presented in Alur (1994). The major additions are the urgent edges, Boolean guards and a clear definition of the behavior of discrete variables. The main features of this formalism are reminded below.

2.1. Variables, expressions and constraints

Let $V = V_b \cup V_z$ be the finite set of discrete variables composed of the set of Boolean variables V_b taking values in $\{true, false\}$ and the set of integer variables V_z taking values in $\mathbb{Z}$ used by the TADD A^{TADD}.

The following sets may then be defined:

- The set $Expr(V_z)$ of all arithmetical expressions over V_z is defined by:

$$expr ::= z \,|\, v \,|\, expr \diamond expr \,|\, -expr \,|\, (expr) \tag{1}$$

  with $z \in \mathbb{Z}$, $v \in V_z$, and $\diamond \in \{-, +, \times, /\}$, where / stands for the Euclidean division.

- $B(V)$ is the set of all Boolean expressions over V defined by:

$$b ::= true \,|\, expr \,\#\, expr \,|\, b \wedge b \,|\, b \vee b \,|\, \neg b \,|\, (b) \,|\, v \tag{2}$$

  where $expr \in Expr(V_z)$, $v \in V_b$ and $\# \in \{<, \leq, =, \geq, >\}$.

- $A(V)$ is the set of all the actions over V such that:

$$a ::= \epsilon \,|\, v_z := expr \,|\, v_b := \{b, true, false\} \,|\, a\,, a \tag{3}$$

  where ϵ is the null action, $v_z \in V_z$, $v_b \in V_b$, $b \in B(V)$ and $expr \in Expr(V_z)$.

Moreover, let C be a set of special variables taking values in $\mathbb{R}^+$ and named *clocks*; the set $N(C)$ of all timed constraints over a clock c is defined by:

$$n ::= true \,|\, c \,\#\, d \,|\, n \wedge n \tag{4}$$

with $c \in C$, $d \in \mathbb{N}$, and $\# \in \{<, \leq, =, \geq, >\}$.

2.2. Timed Automata with Discrete Data without urgency

The previous definitions allow to define a timed automaton with discrete data as a tuple (L, l_0, V, C, E, I) where:

- L is a finite set of locations.
- $l_0 \in L$ is the initial location.

- V is a set of discrete variables, let $v \in V$ be a variable, $\tilde{v}$ is the variable valuation and $\tilde{V}$ is the set of all the variables valuations.
- C is a set of clocks, let $c \in C$ be a clock, $\tilde{c} \in [0, +\infty)$ is the clock valuation and $\tilde{C}$ is the set of all the clocks valuations.
- $E \subseteq L \times B(V) \times N(C) \times A(V) \times 2^C \times L$ is a set of edges from a source location $l \in L$ to a target location $l' \in L$, with a guard $g \in B(V)$, a timed constraint $t \in N(C)$ and an action $a \in A(V)$. 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.
- $I : L \to N(C)$ assigns an invariant to a location.

The semantics of a TADD is given below, with the following additional definitions:

- $\tilde{C} \models I(l)$ means that the valuations of clocks satisfy the invariant of location l.
- $\tilde{V} \models g(e)$ means that the valuations of discrete variables satisfy the guard g of the edge $e \in E$.
- $\tilde{C} \models t(e)$ means that the valuations of clocks satisfy the timed constraint t of the edge e.
- $[r \mapsto 0]\tilde{C}$ stands for the modification of clocks valuations such that each valuation of clocks from r is reset.
- $[V_a \overset{a}{\mapsto} \mathbb{N}]\tilde{V}$ stands for the modification of the valuations of variables from $V_a \subseteq V$ according to the action a.

The semantics of a TADD is a transition system $\langle S, s_0, \to \rangle$ where a state $s \in S$ is a tuple $(l, \tilde{V}, \tilde{C})$, $s_0 = (l_0, \tilde{V}_0, \tilde{C}_0)$ is the initial state and $\to \subseteq S \times S$ is the transition relation such that:

$$(l, \tilde{V}, \tilde{C}) \to (l, \tilde{V}, \tilde{C} + d) \text{ if } \forall c, \forall d', 0 \leq d' \leq d, \tilde{c} + d' \models I(l) \tag{5}$$

$$(l, \tilde{V}, \tilde{C}) \to (l', \tilde{V}', \tilde{C}') \text{ if } \exists e : l \overset{g,t,a,r}{\longrightarrow} l', \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \overset{a}{\mapsto} \mathbb{N}]\tilde{V} \text{ and } \tilde{C}' \models I(l') \tag{6}$$

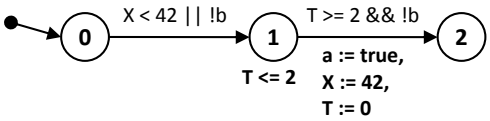


Figure 2: A Timed Automaton with Discrete Data .

Figure 2 is an example of TADD with one clock T, two Boolean variables (a,b) and one integer variable X. TADD will follow the following rules of representations:

- Locations (0, 1 and 2 in this example) are represented by circles, location names are in bold font.
- The initial location (here, location 0) is represented with a source edge.
- Locations invariants (**T <= 2** for location 1 in this example) are in bold font.
- Edges are represented by arrows.
- Guards and time constraints are in normal font. The guard and time constraint of an edge may be combined by a disjunction or conjunction operator, respectively noted || and && (T >= 2 && !b, for example). The Boolean complement operator (NOT operator) will be noted !.
- Actions on variables will be represented by the assignment operator **:=** in bold font; actions are separated by comas, when several actions are associated with an edge. Clock resets will follow the same pattern, the assignment being limited to reset (zero-assignment), like in the edge from location 1 to location 2.

2.3. Complete semantics of TADD

This semantics includes the concept of urgent edge. A Boolean attribute noted $u \in \{true, false\}$ is associated with every edge in that case. Edges with the urgent attribute u set to *true* are named *urgent edges* whereas those with the urgent attribute equal to *false* are referred to as normal edges (generally simply called edges without precision). The intuitive action of this attribute is to give priority to urgent edges over timed evolutions (equation 5). It is obvious that no constraint on clocks valuations must take place in an urgent edge ($t = true$).

The syntax of TADD is then modified: an edge becomes a 7-tuple of the form $e = (l, g, t, u, a, r, l')$ with two locations l and l', one guard g, one time constraint t, the urgent attribute u, one action a and a set of clocks to reset r.

With this addition, the complete semantics of a TADD is a transition system $\langle S, s_0, \rightarrow \rangle$ where a state $s \in S$ is a tuple $(l, \tilde{V}, \tilde{C})$, $s_0 = (l_0, \tilde{V}_0, \tilde{C}_0)$ is the initial state and $\rightarrow \subseteq S \times S$ is the transition relation such that:

$$
\begin{aligned}
&(l, \tilde{V}, \tilde{C}) \rightarrow (l, \tilde{V}, \tilde{C} + d) \text{ if} \\
&\qquad \forall c, \forall d', 0 \leq d' \leq d, \tilde{c} + d' \models I(l), \text{ and} \\
&\qquad \forall e = (l, g, t, u, a, r, l'), u = false \text{ or } (u = true \text{ and } \tilde{V} \not\models g \text{ or } \tilde{C} \not\models I(l'))
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
&(l, \tilde{V}, \tilde{C}) \rightarrow (l', \tilde{V}', \tilde{C}') \text{ if} \\
&\qquad \exists e : l \xrightarrow{g,t,u,a,r} l', \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \overset{a}{\mapsto} \mathbb{N}]\tilde{V} \text{ and } \tilde{C}' \models I(l')
\end{aligned} \tag{8}
$$

The introduction of urgent Boolean attributes removes potential concurrencies between urgent and timed edges. Nevertheless other concurrencies remain, as exemplified in the examples of Figure 3 where urgent edges are represented by double-lined arrows:

- several urgent edges starting from the same location are concurrent when this location is active and when their guards are satisfied (Figure 3a);

- several non-urgent edges starting from the same location are concurrent when this location is active and when their guards and time constraints are satisfied (Figure 3b); in the case of this example, the two edges are concurrent when the clock value is equal to 2;
- several non-timed (urgent or non-urgent) edges starting from the same location are concurrent when this location is active and when their guards are satisfied (Figure 3c).

These concurrency structures imply that the evolutions of a TADD model are non-deterministic which is compulsory to model the behavior of plant in a realistic fashion.

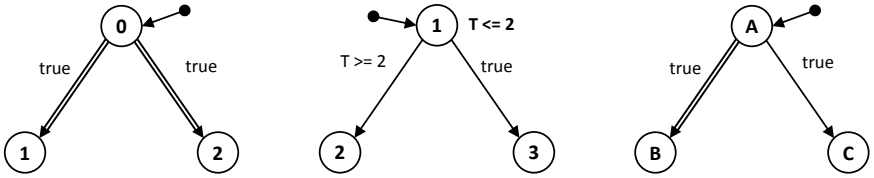


Figure 3: Three examples of TADD.

2.4. Network of Timed Automata with Discrete Data

A network of TADD is a set of automata $A_{\mathrm{i}}^{\mathrm{TADD}} = (L_i, l_i^0, V, C, E_i, I_i)$ with $i \in \mathbb{N}^*$; the sets of clocks (C) and variables (V) are shared by all automata. The network thus is $\{A_{\mathrm{i}}^{\mathrm{TADD}}\} = (\bar{L}, \bar{l}_0, V, C, E, I)$ 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 defined as the active location vector.
- $\bar{l}_0 = (l_1^0, l_2^0, \ldots, l_n^0)$ is defined as the initial location vector.
- V and C are the shared set of variables and clocks.
- $E = \bigcup_i E_i$
- $I(\bar{l}) = \bigwedge_j I_j(l_i)$ is defined as the common invariant function.
- $\bar{l}[l_i \mapsto l_i']$ stands for the change of the i^{th} value of vector $\bar{l}$ from l_i to l_i'.

The semantics of a network of TADD is a transition system $\langle S, s_0, \rightarrow \rangle$ where a state $s \in S$ is a tuple $(\bar{l}, \tilde{V}, \tilde{C})$, $s_0 = (\bar{l}_0, \tilde{V}_0, \tilde{C}_0)$ is the initial state and $\rightarrow \subseteq S \times S$ is the transition relation such that:

$$(\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}, \tilde{V}, \tilde{C} + d) \text{ if}$$
$$\forall c, \forall d', 0 \leq d' \leq d, \tilde{c} + d' \models I(\bar{l}), \text{ and} \quad (9)$$
$$\forall e = (\bar{l}, g, t, u, a, r, \bar{l}'), u = false \text{ or } u = true \text{ and } \tilde{V} \not\models g \text{ or } \tilde{C} \not\models I(l')$$

$$(\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l_i'], \tilde{V}', \tilde{C}') \text{ if}$$
$$\exists e_i : l_i \overset{g,t,u,a,r}{\longrightarrow} l_i', \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \overset{a}{\mapsto} \mathbb{N}]\tilde{V} \text{ and } \tilde{C}' \models I(\bar{l}') \quad (10)$$

2.5. Example of network of timed automata with discrete data

Figure 4 presents a simple network of TADD where:

- $\bar{L} = \{\{A, B, C\} \times \{1, 2, 3\}\}$.
- $\bar{l_0} = \{A, 1\}$
- $V = V_b \cup V_z = \{a\} \cup \emptyset$
- $C = \{T\}$
- $I(l) : 1 \mapsto T \leq 2$
- There is one urgent edge (A $\rightarrow$ B) and three non-urgent edges : 2 non-timed (A $\rightarrow$ C and 1 $\rightarrow$ 3) and one timed (1 $\rightarrow$ 2)

Two possible evolutions of this network are described below.

1. At the initial state (locations A and 1 active, a false and T equals 0) two edges are concurrent:
   - A $\rightarrow$ B because its guard is satisfied
   - A $\rightarrow$ C for the same reason.

   The other edges cannot be fired because 1 $\rightarrow$ 2 is timed thus forbidden by the urgent edge and the guard of 1 $\rightarrow$ 3 is false. The behavior of the network is non-deterministic. The next active location vector is either (B,1), if the first edge is fired or (C,1) if the second edge is fired; in the latter case, a is set to true.

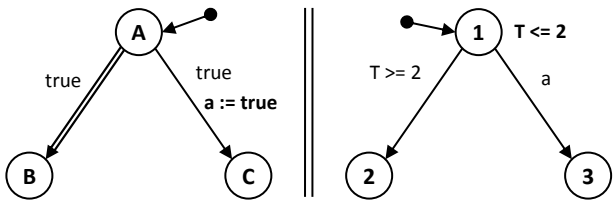


Figure 4: Network of two Timed Automata with Discrete Data.

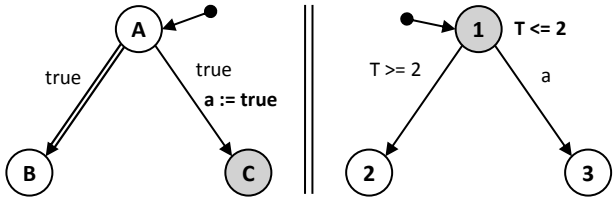


Figure 5: State of the example of figure 4 after the first evolution.

2. Figure 5 shows the active locations assuming that the edge A → C has been fired. When T=0 (firing an urgent edge does not consume time), there is no concurrency because only the guard of 1 → 3 is true. If this edge is not fired before the clock value reaches 2, two edges (1 → 2 and 1 → 3) become concurrent at this date. One of these edges is then fired at this date because location 1 cannot remain longer active owing to its invariant.

In this example, the guards which are always true have been represented; it will no longer be the case in the remainder of this paper, for brevity reasons.

3. Description of the example

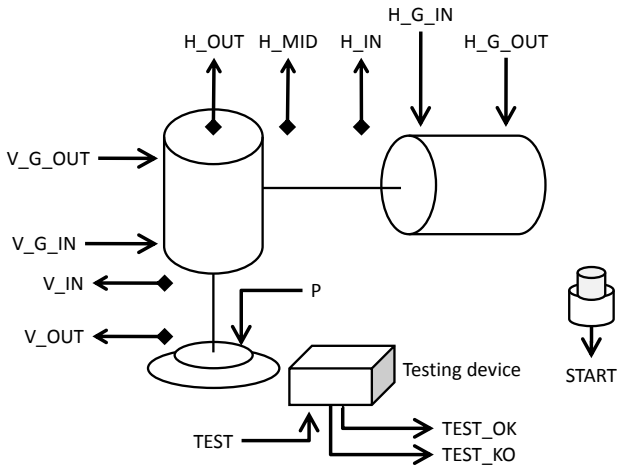


Figure 6: Testing station layout.

The contributions of this work will be exemplified on the example represented in Figure 6. This example is a testing station composed of:

- An horizontal bistable pneumatic cylinder plugged to a 5/3 double-piloted valve.
- A set of three sensors detecting three positions of the rod of this cylinder: rod at the rightmost position, rod at the leftmost position and rod in the middle position.

- A vertical bistable pneumatic cylinder plugged to a 5/3 double-piloted valve.
- A set of two sensors detecting two positions of the rod of this cylinder: rod at upper position and rod at lower position.
- A suction cup plugged to a vacuum pump, used to pick and transfer mechanical parts.
- A testing device, used to verify whether a part is conform or not.
- A push-button used to start the testing procedure.

A PLC will be used to control the plant; the inputs and outputs of the plant are given in Table 1. It matters to note that the inputs of the plant are the outputs of the controller and vice-versa.

Inputs	Description	Outputs	Description
H_G_IN	Command to move the cup to the right	H_IN	Rod at the rightmost position
H_G_OUT	Command to move the cup to the left	H_MID	Rod at the middle position
		H_OUT	Rod at the leftmost position
V_G_IN	Command to move the cup to the top	V_IN	Rod at the upper position
V_G_OUT	Command to move the cup to the bottom	V_OUT	Rod at the lower position
P	Command to activate the vacuum pump		
TEST	Command to the testing device to test part	TEST_OK	Part has passed the test
		TEST_KO	Part has failed the test
		START	Push-button starting the testing procedure

Table 1: List of inputs and outputs of the plant example.

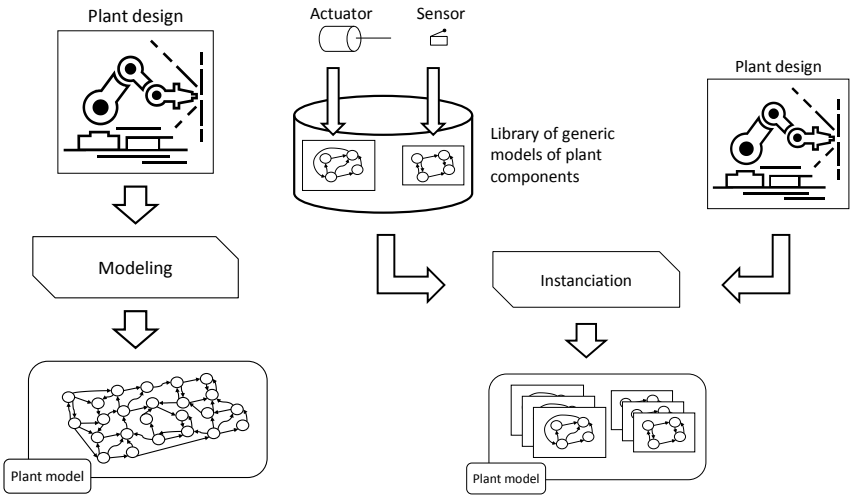
The expected behavior of the closed-loop system is the following:

1. When the START button is pushed, the suction cup goes down and picks the part (it is assumed that a part is present when the START button is pushed).
2. The part is lifted, moved to the middle position and then lowered to the testing device.
3. This device gives back the result of the test (TEST_OK or TEST_KO).
4. - If the test was passed, the suction is stopped and the part moves on. The suction cup comes back to its initial position (move up, then move right).
   - If the test was failed, the part is brought to the leftmost position (move up, then move left) and is released into a collector of non-conform parts. Then the suction cup comes back to its initial position.

4. Building meaningful plant models

Two approaches are *a priori* possible to build the plant model (Figure 7):

- The first one (Figure 7a) is called the monolithic approach and is based on expertise; a single monolithic model is built from the layout of the plant. This approach is obviously error-prone and can be applied reasonably only to toy-cases.
- The second one (Figure 7b) is called the modular approach; the complete plant model is built by instanciation of generic models of plant components (actuators, sensors, ...). This model thus is a set of small formal models which can be easily modified when the real plant changes (new components introduction, existing components modification). Only this approach is applicable to real cases, as detailed in Philippot et al. (2009), and this is the reason why it was selected for this study.



(a) Monolithic approach. (b) Modular approach.

Figure 7: Two approaches to build a plant model.

A first version of generic components models is presented in the next section; no urgent edge is introduced in this basic version. The plant model of the example is then obtained, by instantiation of generic models. The third subsection aims at showing that spurious evolutions leading to meaningless states may occur in the plant model. To solve this issue, an improvement of the solution presented in Perin and Faure (2009) is proposed.

4.1. Basic generic components models description

Figure 8 shows the generic model of a pneumatic cylinder plugged to a 5/3 double-piloted valve. The following variables have been introduced:

- Two logic input variables which are assigned by the controller model and only read by the cylinder model:
  - G_IN (go in): order to move the rod inside the body of the cylinder
  - G_OUT (go out): order to move the rod outside the body of the cylinder
- One output variable which is assigned and read by the cylinder model and only read by other models (such as sensors models):
  - X: integer variable modeling the length of the rod which is outside the cylinder. The value of this variable belongs to the interval [X_min,X_max] with X_max and X_min integers that satisfy X_max > X_min.
- One clock t.

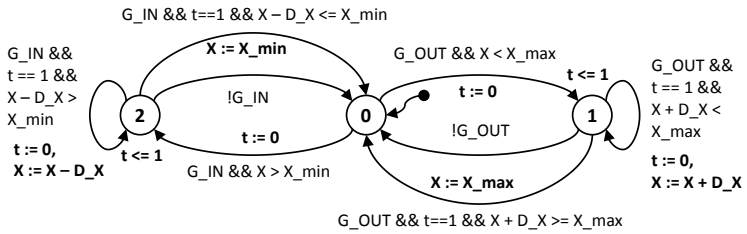


Figure 8: Basic generic model of bi-stable pneumatic cylinder plugged to a 5/3 double-piloted valve.

Details on this model are:

- The locations 0, 1 and 2 represent respectively the motionless rod, the rod going out and coming in the body of the cylinder. The invariant on clock t ensures that for each time step ($t \leq 1$) the variable X is modified (increased or decreased).
- The edges $0 \rightarrow 1$ and $0 \rightarrow 2$ model the beginning of a movement (going out or coming in) due to an order from the controller. The guards of these edges include constraints that mean that a movement is possible only if the physical limit is not already reached. The clock t is reset in order to start counting the elapsed time.
- The edges without time constraint $1 \rightarrow 0$ and $2 \rightarrow 0$ model the end of a movement when the order becomes false.
- The self-loop edges, from/to the locations 1 and 2, model the rod movement. Their guards ensure that the order is still true and that the physical limit (X_max or X_min) will not be exceeded once the edge fired. The time constraints implies that the edge is fired each time step; the value of X is then increased or decreased by D_X, parameter that represents the elementary movement.
- The edges with a time constraint $1 \rightarrow 0$ and $2 \rightarrow 0$ model the end of a movement when the physical limit is reached.

In order to reduce the number of generic models, a unique generic model is built for the set of sensors related to an actuator. This approach will be exemplified with a set of three sensors that is aiming at distinguishing five logical positions of a linear actuator (Figure 9): the extreme positions IN and OUT, the middle position MID and the two positions between the extreme and the middle ones. The physical detection ranges of the logical sensors are grayed; this means for instance that the logical position is IN when the variable X (position of the mobile part of the actuator) stands in the interval [X_min,X_IN_max]. It must be mentioned that, when coupling an instance of the actuator model and an instance of the sensor model, the values of the bound parameters X_IN_max, X_MID_min, X_MID_max, X_OUT_min, in the sensors model, and the D_X parameter, in the actuator model, must be consistent to permit every logical position be detected. A solution is to define D_X as the smallest detection range.

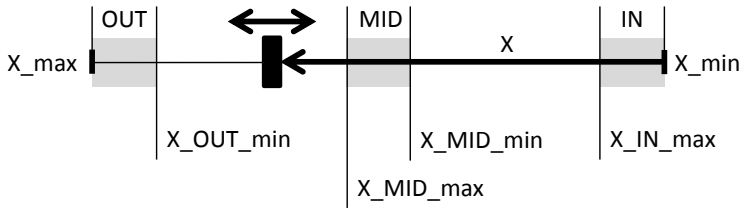


Figure 9: Ranges of the sensors.

Figure 10 shows the generic model of this set of sensors. The following variables have been introduced:

- One input variable which are assigned by another model and only read by the sensors model: X, integer that represents the physical position of a mobile moving on the axis. It is assumed that the minimal value of X is 0; no assumption on the maximal value is made.
- Three logical output variables: IN, OUT and MID. The logical position of the mobile is obtained from the values of these three variables.

This model comprises five locations that correspond to the five logical positions (IN, OUT, MID, Between IN and MID, Between MID and OUT). Every edge represents a change of logical position. It matters to underline that no initial location is defined because the initial physical position of the mobile is unknown. This location will be defined when instantiating the model, according to the physical initial position.

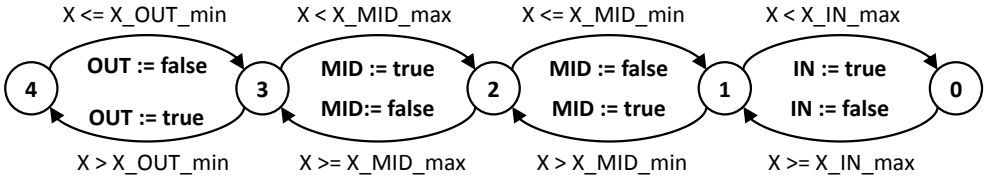
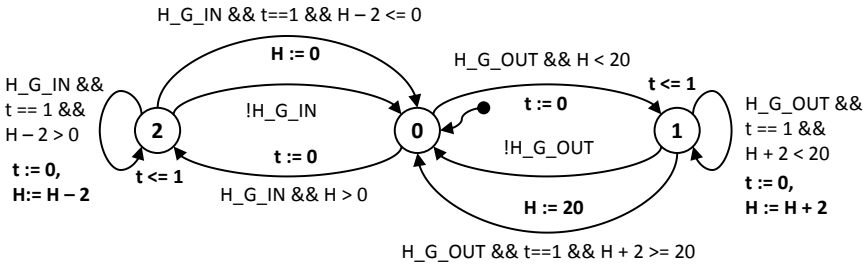


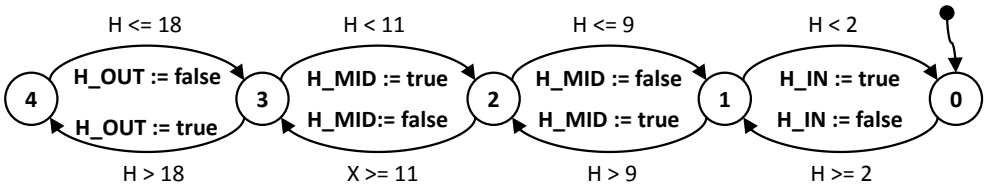
Figure 10: Basic generic model of a set of three logic sensors.

4.2. Plant model

The plant model is obtained by selecting the appropriate generic components models and by instantiating these models as many times as there are instances of these components in the plant. The result of the instantiation process for the horizontal cylinder and its associated set of sensors is presented in Figure 11.



(a) Instanciated model of the horizontal pneumatic cylinder.



(b) Instantiated model of the sensors detecting horizontal movements.

Figure 11: Instantiated models for the horizontal movement of the example.

It has been chosen to implement communications between automata by using exclusively shared variables in the plant model. To avoid multiple assignments, each shared variable is assigned by one and only one component model. In the case of Figure 11, H (position on the horizontal axis) is only assigned by the actuator model (Figure 11a) and H_IN, H_MID and H_OUT are only assigned by the sensors model (Figure 11b). As the initial horizontal position is the rightmost one, the initial location of the sensors model corresponds to the position IN.

The complete set of models for the plant is composed of:

- One instance of the generic pneumatic cylinder plugged to a 5/3 double-piloted valve model (Figure 11a), named H_Act, to model the horizontal actuator, where X, G_IN and G_OUT are instantiated respectively by H, H_G_IN and H_G_OUT, and where X_min, X_max and D_X are respectively settled to 0, 20, and 2.

- One instance of the generic pneumatic cylinder plugged to a 5/3 double-piloted valve model, named V_Act, to model the vertical actuator, where X, G_IN and G_OUT are instantiated respectively by V, V_G_IN and V_G_OUT, and where X_min, X_max and D_X are respectively settled to 0, 10, and 1.

- One instance of the generic model of sensors (Figure 11b), named H_Sen, to model the set of sensors detecting the position of the horizontal rod, where X, OUT, MID, and IN are instantiated respectively by H, H_OUT,

H_MID, and H_IN, and where X_IN_max, X_MID_min, X_MID_max, and X_OUT_min are respectively set to 2, 9, 11, and 18. As the initial position of the rod is supposed to be fully retracted, the variable H_IN is initialized to *true* and the initial location will be the rightmost location 0.

- One instance of the generic model of sensors, not described in this paper but similar to the 3-sensor generic model with also one initial location but only three locations standing for the possible sensors values and named V_Sen, to model the set of sensors detecting the position of the vertical rod, where X, OUT and IN are instantiated respectively by V, V_IN, and V_OUT and where X_IN_max, and X_OUT_min are respectively set to 1 and 9. As the initial position of the rod is supposed to be fully retracted, the V_IN variable is initialized to *true* and the initial location is defined to be consistent with this value.

- One instance of the generic model of a suction pad, one instance of the generic model of a testing device, and one instance of the generic model of a push-button. The latter two models will be used only during the verification process and are described in section 7.

4.3. Evolution of the plant model

The aim of this subsection is to show that spurious evolutions leading to meaningless states can occur in the plant model. This issue will first be exemplified on the previous example, and then generalized.

4.3.1. Illustration of the issue on the example

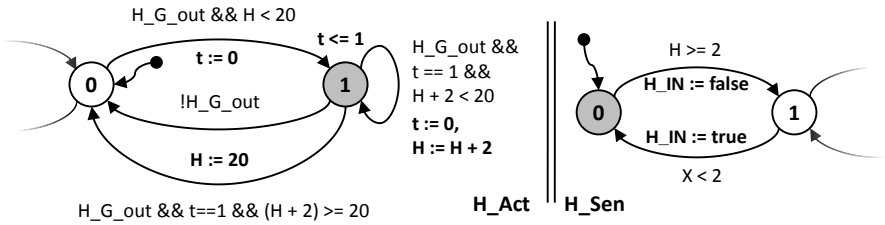
The following simple scenario represented in Figure 12 will be used to pinpoint the issue: the rod of the horizontal actuator is in its rightmost position (rod retracted) and this actuator receives from the controller the order to move to the middle position (the variable H_G_OUT is set by the controller and will remain true until the variable H_MID becomes true).

At the beginning of this scenario, the active locations of H_Sen and H_Act are their initial locations; H_IN is *true*, H_MID and H_OUT are *false* and H equals 0. When H_G_OUT becomes true, the active location of H_Act becomes the right location, which models the outgoing movement of the rod; then the clock t increases by 1 and the self-loop edge of H_Act is fired, setting the value of H to 2 and resetting t. The state described in Figure 12a is thus reached.

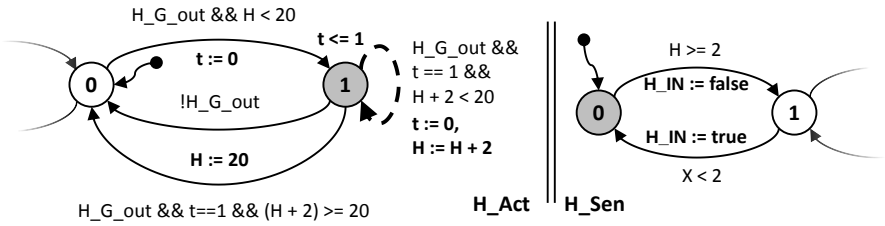
From this state, two concurrent evolutions are possible according to the selected semantics:

- If the semantics given by (7) is selected, the clock t increases by 1 and thus permits another evolution of H_Act (the invariant is satisfied), using the self-loop edge that models the end of one outgoing movement step; during this evolution, the value of H becomes 4. The active location of H_Sen remains the same. This evolution is represented in Figure 12b, where the dotted edge represents the fired edge.

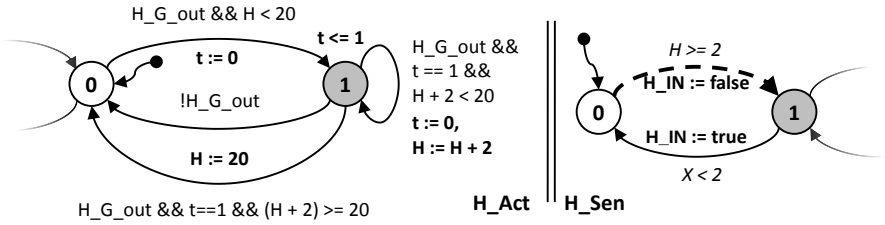
- If the semantics given by (8) is selected (Figure 12c), H_Sen evolves. The dotted edge is fired, which means that the sensor has detected that the rod is no longer in its rightmost position (H_IN is reset). The clock t remains equal to 0 and H_Act does not evolve.



(a) Initial state of the analysis (the active states are greyed and H=2).



(b) Evolution 1: H is increased whereas the sensor model does not evolve.



(c) Evolution 2: H does not increase but the sensor model evolves, resetting H_IN

Figure 12: Meaningless and meaningful evolutions of the plant model.

This discussion clearly shows that the evolution which consumes time (first evolution above) leads to a meaningless state that does not represent correctly the physical behavior of the plant components: the model of the actuator assigns the value 4 to H whereas the sensor model always yields the information H_IN true. This evolution is spurious. On the opposite, when the second interpretation is selected, the sensor model evolves at the right time. This evolution corresponds to the real behavior and thus is meaningful.

4.3.2. Generalization

The issue that was exemplified above is the general issue of concurrent evolutions in a set of timed automata. Edges can thus be concurrent with respect to only one semantics or to several ones. The first case is not really an issue, because it often models a real concurrency problem, such as concurrent access to shared resources. On the other hand, the second case can lead to modeling flaws. Indeed, plant designers often think that the time is spent (time evolution semantics given by equation 7) after all other possible evolutions have been made (location change semantics given in equation 8). This is not true, unless the urgent attribute is used, and leads to spurious evolutions and state spaces that include meaningless states, especially when physical systems evolutions with different time scales,

like the movement of an actuator rod and the detection of a sensor, are to be modeled.

4.4. Modification proposed

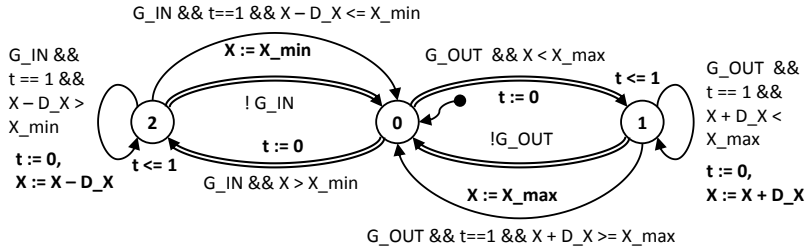
4.4.1. Instantaneous and timed evolutions

In order to solve the problem that was emphasized in the previous section, it matters to come back to the plant modeling process. The designer has to model the behavior of a physical system by means of a formalism for timed discrete event systems which owns particular semantics. Whatever the semantics choice, all evolutions of the formal model must represent real meaningful evolutions of the plant. These latter ones can be separated in two sets, according to a time consumption criterion:

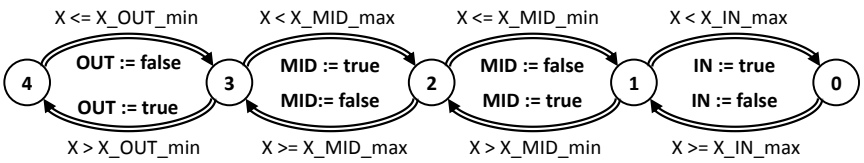
- *Instantaneous evolutions* are evolutions which have no duration, in the designer's viewpoint, such as the detection of a part by a sensor;
- *Timed evolutions* are evolutions which consume time, such as the movement of a rod.

In the remainder of this paper, it will be assumed that every edge of the network of TADD models either an instantaneous evolution or a timed evolution, but not both. If this is not the case, the edge must be split.

The solution consists in giving priority to instantaneous evolutions over timed evolutions by using TADD with urgency and by setting the urgent attribute to *true* for every instantaneous evolution and to *false* for every timed evolution. The formal models of the plant components that will be used for verification are merely obtained by instantiation of the modified meaningful generic models (Figures 13a for cylinders and 13b for sensors) and therefore contain only meaningful evolutions.



(a) Modified version of the generic model of the bistable pneumatic cylinder.

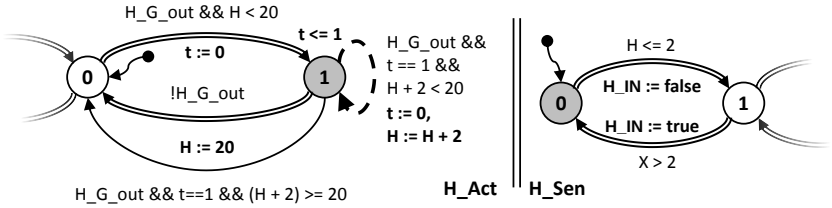


(b) Modified version of the generic sensor model.

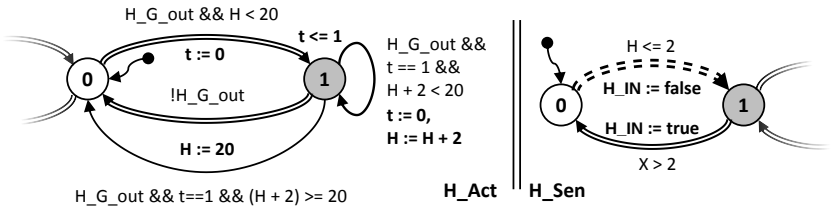
Figure 13: Modified version of generic models.

4.4.2. Application to the example

Once the two models H_Act and H_Sen are modified (Figure 14), when H reaches the value 2 (Figure 14a), only the instantaneous evolution of H_Sen (Figure 14b) remains possible, assuming that H_G_OUT remains *true*. The spurious evolution which was discussed in Section 4.3.1 is no longer possible, because the urgent edge has priority over the timed one.



(a) Initial state, H equals 2.



(b) Only the meaningful evolution remains possible.

Figure 14: Evolutions of the modified models.

5. Building the controller model

This section focuses on construction of the model of the controller, assuming that the controller behavior bas been specified in GRAFCET, an industrial specification language (IEC 60848 (2002)), and implemented into a PLC with cyclic I/O scanning.

A brief reminder of the GRAFCET syntax and evolution rules is first presented. Then, the behavior of the example (Figure 6) controller, in the form of a GRAFCET, is given to illustrate this reminder. A method to define formally the behavior of a GRAFCET model by a set of algebraic equations is proposed in the third part. This allows to model the behavior of the controller, PLC which executes such a set of equations, using TADD.

5.1. Brief reminder on the GRAFCET standard

Only the elements of the GRAFCET syntax and semantics that are useful to understand the rest of this paper are reminded below. The interested readers are referred to David (1995), Bierel et al. (1997), Provost et al. (2011) for deeper presentations.

5.1.1. GRAFCET syntax

A GRAFCET model describes the expected behavior of a logic controller which receives logic input signals and generates logic output signals. A GRAFCET (Figure 15) comprises *steps*, graphically represented by squares, and *transitions*, represented by horizontal lines; a step can only be linked to transitions and a transition only linked to steps. The links from steps to transitions and from transitions to steps are directed *links*. The default orientation is from top to bottom and it is not necessary in this case to put an arrow on the link. An arrow must be put on a link if this link goes from bottom to top or may be put on any link to ease understanding.

A step defines a partial state of the system and can be active or inactive; hence, a Boolean variable, named step activity variable can be defined for each step. The set of all active steps define the *situation* of the GRAFCET.

Actions may be associated to a step; an action associated to a step is performed only when this step is active and then acts upon an output variable.

A *transition condition* must be associated to each transition; this condition is a Boolean expression which may include input variables, steps activity variables and conditions on time.

5.1.2. Evolution rules

The detailed behavior of any GRAFCET model can be obtained by applying the following five evolution rules:

1. At the initial time, all the initial steps, defined by the model designer and double-squared, are active; all the other steps are inactive.
2. A transition is enabled when all the steps that immediately precede this transition (upstream steps of the transition) are active. A transition is fireable when it is enabled and when the associated transition condition is true. A fireable transition must be immediately fired.
3. Firing a transition provokes simultaneously the activation of all the immediately succeeding steps and the deactivation of all the immediately preceding steps.
4. When several transitions are simultaneously fireable, they are simultaneously fired.
5. When a step shall be both activated and deactivated, by applying the above previous evolution rules, it is activated if it was inactive, or remains active if it was previously active.

5.2. GRAFCET specification of the controller of the example

The expected behavior of the controller of the example has been described textually in section 3. The GRAFCET of Figure 15 represents this behavior.

The inputs and outputs of this model are respectively the outputs and inputs of the plant. Moreover, the following notations are used in the transition conditions:

- $\cdot$ is the disjunction operator,
- $+$ is the conjunction operator,

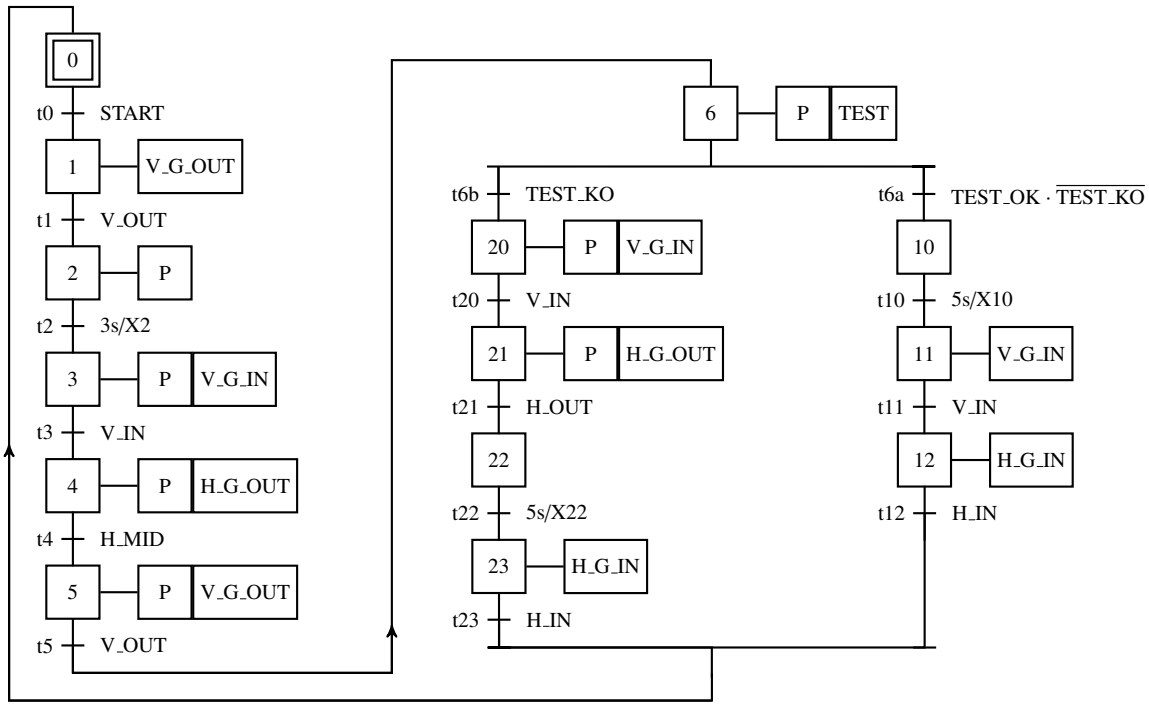


Figure 15: GRAFCET specification of the controller of the example.

- $\overline{x}$ is the complement of x,
- Ns/Xi means that the condition becomes true N seconds after the activation of the step i.

5.3. Algebraic representation of a GRAFCET specification

The behavior of a GRAFCET model can be represented by a set of algebraic equations, as detailed in Machado et al. (2006). This requires to introduce the following definitions.

Let $Step$ be the set of steps of the GRAFCET, let $s \in Step$ be a step:

- X_s is the activity variable associated to step s. It is set to *true* if the step is active, *false* otherwise.
- Pre_s is the set of preceding transitions (using directed links) of step s.
- $Post_s$ is the set of following transitions (using directed links) of step s.

Let $Trans$ be the set of transitions of the GRAFCET, let $t \in Trans$ be a transition:

- TC_t is the transition condition associated to transition t.
- $Up_t \subset Step$ is the set of upstream steps of transition t.

- $Dn_t \subset Step$ is the set of downstream steps of transition t.
- FC_t is the firing condition of transition t. It is set to *true* if the transition can be fired.

Let *Out* be the set of output variables of the GRAFCET, let $o \in Out$ be an output variable.

- $AS_o \subset Step$ is the set of steps where the output variable o is set through actions.

The following three sets of equations defines then the behavior of a GRAFCET:

Firing conditions of transitions:

$$FC_t = (\bigwedge X_s) \wedge TC_t \tag{11}$$

where X_s belongs to Up_t.

Steps activities evolutions: :

$$X_s = (\bigvee FC_t) \vee (X_s \wedge (\bigwedge (\neg FC_u))) \tag{12}$$

where $FC_t \in Pre_s$ and $FC_u \in Post_s$.

Outputs values definitions:

$$o = (\bigvee X_s), \tag{13}$$

where X_s belongs to AS_o

At the initial situation, the step activity variables of the initial steps are *true*, the other steps activity variables are *false*. The output variables associated through actions to the initial steps are also *true*, the other output variables false.

5.4. Controller model

The I/O scanning cycle of a PLC is reminded in Figure 16; this cycle comprises three phases (input values reading, execution of the control application and output values updating) that are performed sequentially. If the behavior of the controller has been specified in GRAFCET, the control application can be described by the three sets of equations which have been previously defined and executing this application consists in evaluating sequentially these sets :

1. The firing conditions of transitions are first computed.
2. Then, the values of the step activity variables are updated.
3. Last, the outputs values are computed.

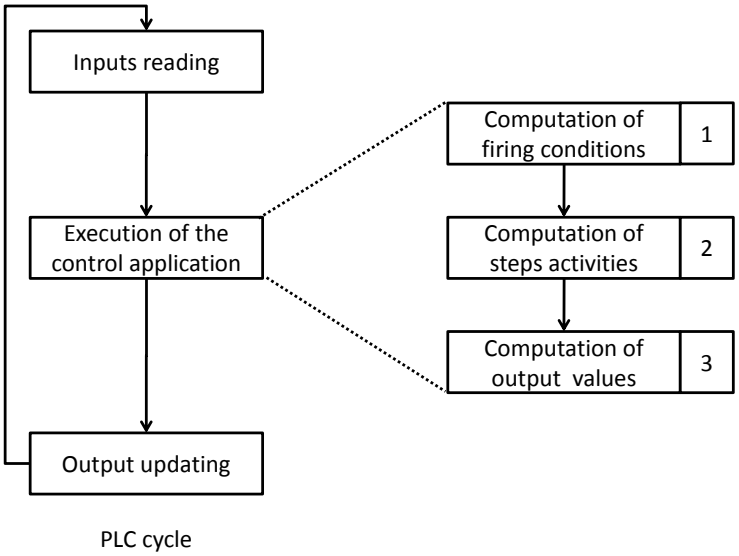


Figure 16: Implementation of the algebraic model of GRAFCET in a PLC.

It is then possible to build a first version of the model of the controller in TADD (Figure 17) using the three sets of equations. This model does not represent the first and third phases of the PLC cycle because, in the model of the closed-loop system, the inputs and outputs of the PLC model are variables shared with the model of the plant. The inputs reading phase is merely performed by reading the values of variables written by the plant model and the outputs updating phase consists in writing variables. Hence, only the second phase of the PLC cycle is modeled. To take into account the order of the three computations, three locations, then edges, are introduced; the three sets of equations are actions associated to the edges. Last, it matters to underline that all edges are urgent because it is assumed that the duration of the PLC cycle is far smaller than that of the timed evolutions of the plant components; the consequence of this assumption on the behavior of the closed-loop system will be discussed in the following section.

As some transition conditions depend on time and specifically on durations of steps (conditions associated to transitions t2, t10 and t22 in Figure 15), models of timers are to be added to the previous model of the PLC. A solution to model a timer with timed automata is given in Mader and Wupper (1999). Figure 18 shows how this solution can be adapted to a timer which measures the time t1 elapsed from the activation of a step i:

- In the initial location, the timer is disabled. When the variable T_Xi_t1 which represents the activation of the step becomes *true*, the active location becomes RUN; the timer is launched.
- From this new location, two evolutions are possible. The timer can be disabled if T_Xi_t1 becomes false. Otherwise, the active location becomes OK when the duration defined by the parameter t1 is elapsed; the variable T_Xi_t1_Q becomes *true*. This variable is the transition condition of the transition which follows the step and can be used in the firing condition of this transition.

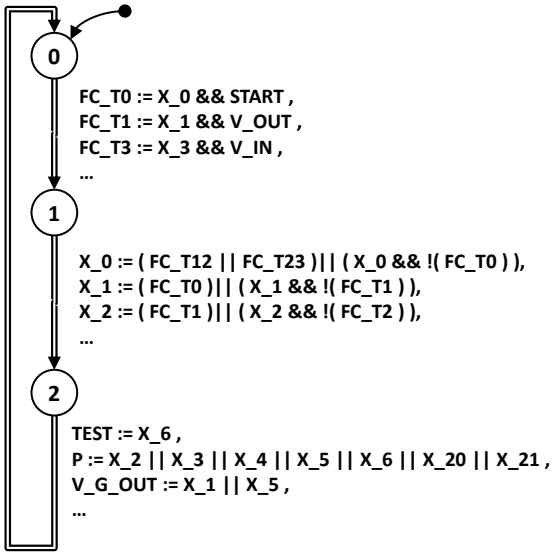


Figure 17: First version of the PLC model for the example of Figure 15.

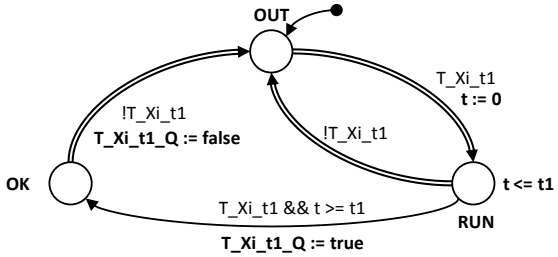


Figure 18: Model of a timer associated to step *i*.

Once the models of timers defined, the complete set of algebraic equations for the GRAFCET of Figure 15 can be stated (Figure 19). Some firing conditions (CF_T2, CF_T10, and CF_T22) depend on timer output variables and three GRAFCET outputs must be defined to launch the timers (first three equations of 19c).

The complete model of the controller specified by the GRAFCET of Figure 15 and implemented in a PLC with cyclic I/O scanning is the network of TADD given in Figure 20:

- Figure 20a models the I/O scanning and controller behavior;
- Figures 20b, 20c, and 20d model the timers;
- Communication between these models is performed by the variables T_Xi_ti (timer launch) and T_Xi_t1_Q (end of timer).

$$CF_{T0} = X_0 \wedge START$$
$$CF_{T1} = X_1 \wedge V\_OUT$$
$$CF_{T2} = X_2 \wedge T\_X2\_3s\_Q$$
$$CF_{T3} = X_3 \wedge V\_IN$$
$$CF_{T4} = X_4 \wedge H\_MID$$
$$CF_{T5} = X_5 \wedge V\_OUT$$
$$CF_{T6a} = X_6 \wedge TEST\_OK$$
$$CF_{T6b} = X_6 \wedge TEST\_KO$$
$$CF_{T10} = X_10 \wedge T\_X10\_5s\_Q$$
$$CF_{T11} = X_11 \wedge V\_IN$$
$$CF_{T12} = X_12 \wedge H\_IN$$
$$CF_{T20} = X_20 \wedge V\_IN$$
$$CF_{T21} = X_21 \wedge H\_OUT$$
$$CF_{T22} = X_22 \wedge T\_X22\_5s\_Q$$
$$CF_{T23} = X_23 \wedge H\_IN$$

(a) Firing conditions

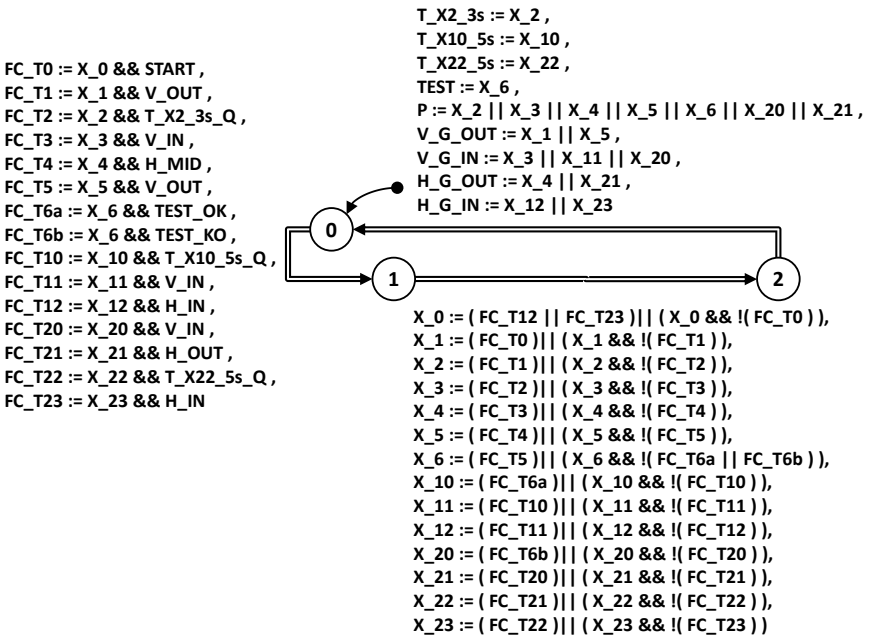
$$X_0 = (CF_{T12} \vee CF_{T23}) \vee (X_0 \wedge \neg(CF_{T0}))$$
$$X_1 = (CF_{T0}) \vee (X_1 \wedge \neg(CF_{T1}))$$
$$X_2 = (CF_{T1}) \vee (X_2 \wedge \neg(CF_{T2}))$$
$$X_3 = (CF_{T2}) \vee (X_3 \wedge \neg(CF_{T3}))$$
$$X_4 = (CF_{T3}) \vee (X_4 \wedge \neg(CF_{T4}))$$
$$X_5 = (CF_{T4}) \vee (X_5 \wedge \neg(CF_{T5}))$$
$$X_6 = (CF_{T5}) \vee (X_6 \wedge \neg(CF_{T6a} \vee CF_{T6b}))$$
$$X_{10} = (CF_{T6a}) \vee (X_{10} \wedge \neg(CF_{T10}))$$
$$X_{11} = (CF_{T10}) \vee (X_{11} \wedge \neg(CF_{T11}))$$
$$X_{12} = (CF_{T11}) \vee (X_{12} \wedge \neg(CF_{T12}))$$
$$X_{20} = (CF_{T6b}) \vee (X_{20} \wedge \neg(CF_{T20}))$$
$$X_{21} = (CF_{T20}) \vee (X_{21} \wedge \neg(CF_{T21}))$$
$$X_{22} = (CF_{T21}) \vee (X_{22} \wedge \neg(CF_{T22}))$$
$$X_{23} = (CF_{T22}) \vee (X_{23} \wedge \neg(CF_{T23}))$$

(b) Steps activities

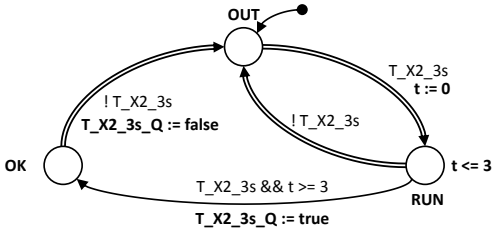
$$T\_X2\_3s = X_2$$
$$T\_X10\_5s = X_{10}$$
$$\mathrm{T}\_X22\_5s = X_{22}$$
$$TEST = X_6$$
$$P = X\_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_{20} \vee X_{21}$$
$$V\_G\_OUT = X_1 \vee X_5$$
$$V\_G\_IN = X_3 \vee X_{11} \vee X_{20}$$
$$H\_G\_OUT = X_4 \vee X_{21}$$
$$H\_G\_IN = X_{12} \vee X_{23}$$

(c) Outputs values

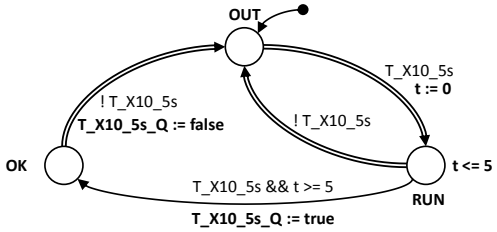
Figure 19: Algebraic representation of the GRAFCET specification of the example.



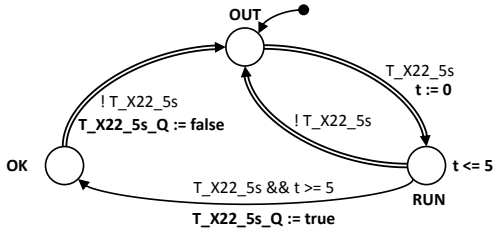
(a) PLC cycle model.



(b) Model of the timer associated to step 2.



(c) Model of the timer associated to step 10.



(d) Model of the timer associated to step 22.

Figure 20: Complete controller model of the example.

6. Building meaningful models of closed-loop systems

The model of the closed-loop system, termed in what follows closed-loop model, is a network of TADD composed by the plant model and the controller model.

6.1. Closed-loop model evolutions

Figure 21 shows a possible evolutions sequence of a part of the closed-loop model. For readability purposes, only a partial representation of the vertical pneumatic cylinder (V_Act model) and of the PLC cycle model is presented. As a reminder, a partial view of the GRAFCET specification is given, with the active steps grayed. The transitions which must be fired at a given moment are also grayed.

At the initial date, Figure 21a, the active locations of both V_Act and PLC cycle model are the initial ones (X_0 is *true* for instance). It will be assumed that the Boolean variable START is *true*, all other Boolean variables are *false* and V, position on the vertical axis, equals 0.

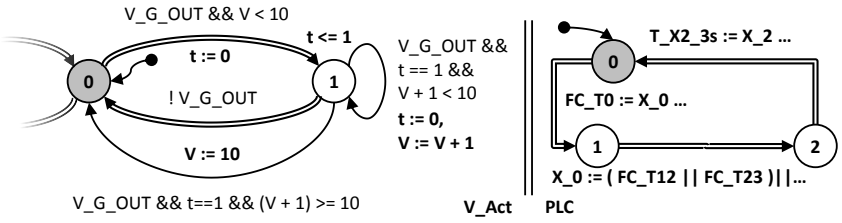
- The first evolution is an evolution of the PLC cycle model (Figure 21c), because V_G_OUT is *false*. The firing conditions (FC) are computed: as START is *true* and step 0 active (X_0 is *true*), the firing condition FC_0 becomes *true*.
- The PLC cycle model continues to evolve from location 1 to 2 (Figure 21e), calculating the new active steps. As FC_0 is *true*, step 0 is deactivated and step 1 activated (X_0 reset, X_1 set).
- During the third evolution of the PLC cycle from 2 to 0 (Figure 21g), the values of the output variables of the controller are computed. V_G_OUT becomes *true* because step 1 is active.
- Then, two urgent edges, one in the plant model and one in the controller model are concurrent. It will be assumed that the edge of the plant model is fired (otherwise the previous evolutions may occur leading another time to a concurrency situation), as shown in Figure 21i. The V_Act model evolves from location 0 to the location 1 modeling the outgoing movement of the rod, because V_G_OUT is *true*.

This leads to a deadlock in the plant model. As the urgent edges of the PLC cycle model have priority over the only fireable edge of the plant model (timed self-loop from/to location 1), this edge will never be fired, the clock t and the variable V will never be increased. This would mean that the rod of the cylinder stays motionless while this actuator has received the order to move out; this state is really a meaningless state for a faultless actuator.

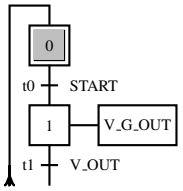
6.2. Removing deadlock of the plant model

The solution to solve this issue is to allow the evolutions of the PLC cycle model *only when an input variable of the controller has been modified*. This solution must be also applied to the variables which represent the end of timers (T_Xi_t1_Q variables) because the models of timers include also timed evolutions which could be superseded by urgent edges of the PLC cycle. This solution is implemented by introducing new variables:

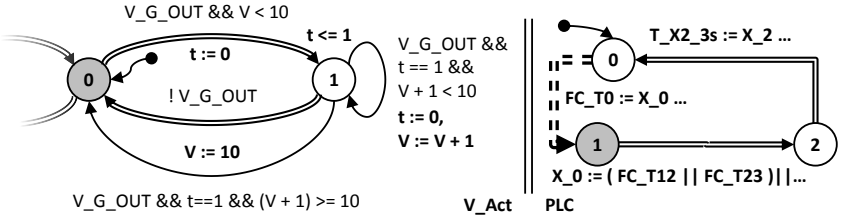
- EVOL_PL (for evolution of the plant), Boolean variable representing the assignation of an input variable of the controller. This variable becomes *true* each time an input variable of the controller is assigned by the plant model. An action -EVOL PL := true- is then added to every edge of the plant model where an input variable



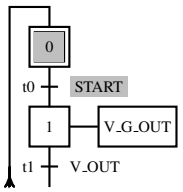
(a) Initial state.



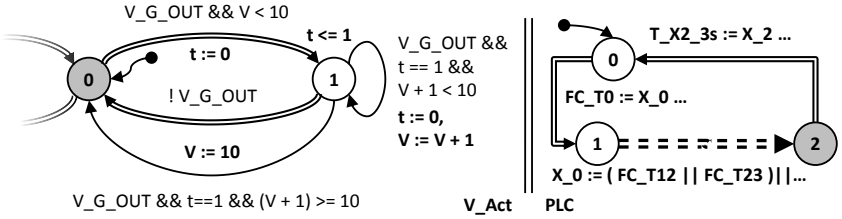
(b) Specification.



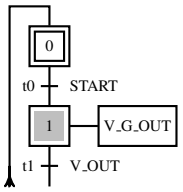
(c) First evolution: firing conditions.



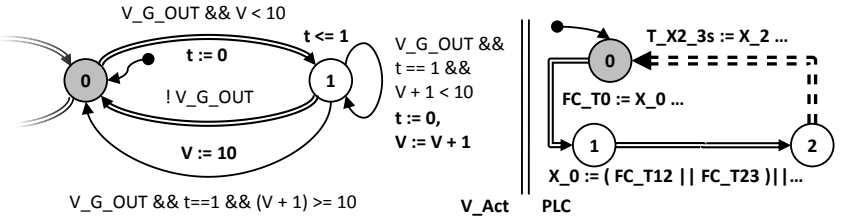
(d) Specification.



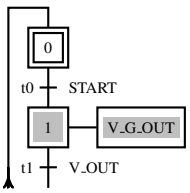
(e) Second evolution: active steps.



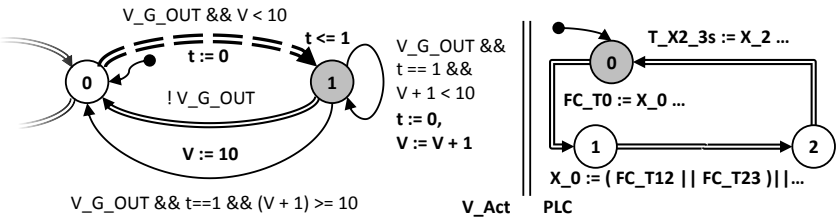
(f) Specification.



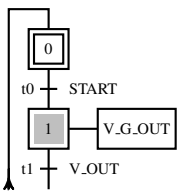
(g) Third evolution: set of output variables.



(h) Specification.



(i) Fourth evolution: evolution of the actuator model.



(j) Specification.

Figure 21: Evolutions of the closed-loop model.

of the controller is assigned. This variable will be reset at the beginning of the PLC cycle as explained below. Figure 22 shows the modified model of the set of vertical sensors; four actions are introduced because input variables of the controller are assigned in every edge of this model.

- END_TI (for end of a timer), Boolean variable which represents the end of a timer. This variable becomes *true* when the duration of the timer is elapsed. An action -END_TI := true- is then added to the edges of timers which model this end (Figure 23).

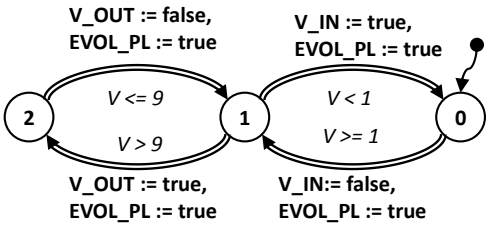


Figure 22: Modified version of the model of the set of vertical sensors.

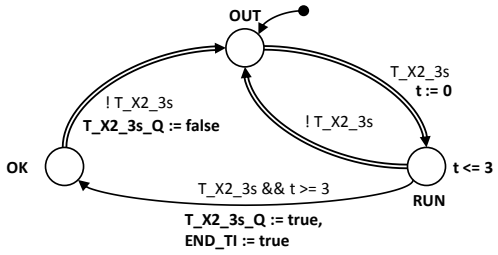


Figure 23: Modified version of the model of the timer associated to step 2.

Once these variables introduced in the plant model and in the timer models, the PLC cycle model must be modified as shown in Figure 24. A new initial location W (for waiting) is defined. The guard of the edge W $\rightarrow$ 0 is true if an input variable of the controller has changed or a timer has ended. In that case, the model may evolve (but is not forced to) and EVOL_PL and END_TI are both reset to avoid two consecutive cycles without change of the value of an input variable or end of a timer.

If focus is now put on the behavior of the closed-loop model, it can be noted that:

- When W is the active location of the PLC cycle model and the guard of the edge starting from this location false, only the plant model or timer models can evolve.
- When W is active and this guard true, the PLC cycle model may perform one (and only one) cycle. During this evolution, concurrencies between every edge of the model and urgent edges of other models may be found and solved, as usually, in a non-deterministic fashion (evolution of the controller or the plant model); this implies that several inputs of the controller may be changed by the plant model during the cycle.

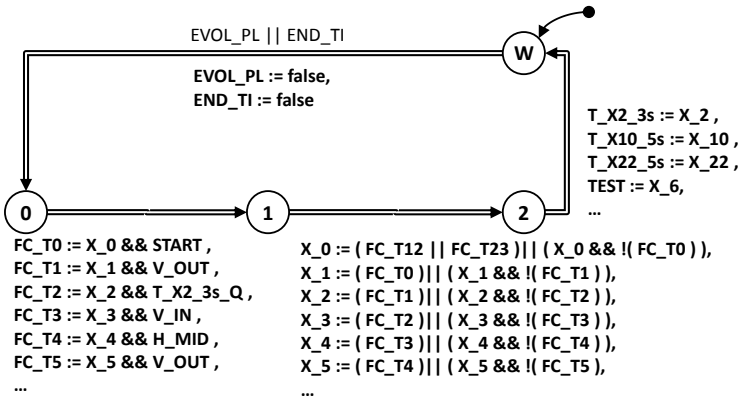


Figure 24: Modified model of the PLC cycle.

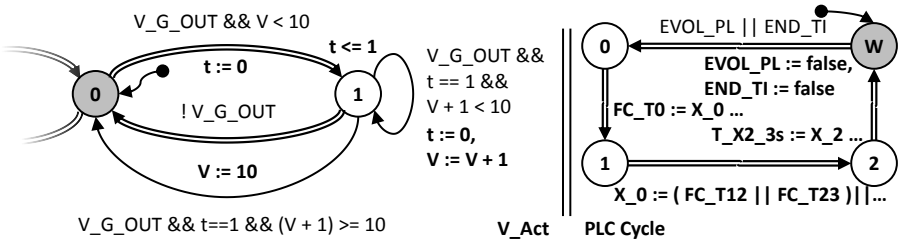
Hence, from one cycle to the following one, at least one input variable is changed -the controller is reactive enough, no input change is missed- but several input variables may change as this is the case when a real PLC is connected to a real plant. The closed-loop model behaves in a realistic manner what is mandatory to obtain trustworthy verification results.

With these modifications, the previous sequence of evolutions is changed as shown in Figure 25. The initial state is the same (Figure 25a), except that EVOL_PL is *true* because it is assumed that START has changed (no evolution is possible without this assumption). This allows the PLC cycle model to start a cycle as shown in Figure 25b; EVOL_PL becomes false. The next three evolutions are unchanged and have been skipped. At the end of this PLC cycle, V_G_OUT has been set to *true* and, as neither EVOL_PL nor END_TI are true (no input of the controller has changed), the PLC model is stopped. The only evolution is then that of the V_Act model shown in Figure 25c. At this point, as no urgent edge is fireable, the self-loop edge modeling the movement of the rod can be fired after 1 time unit, as shown is Figure 25d.

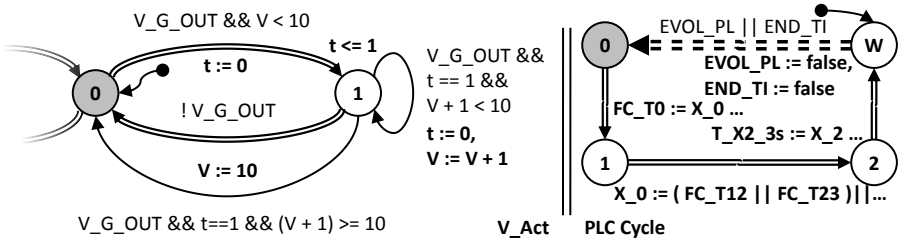
There is no more deadlock in the plant model. Another PLC cycle is possible when the next input change occurs, and so on; in the case of this example, this change is generated by the vertical sensors model, when V becomes equal to 1.

6.3. Removing deadlocks in the controller model

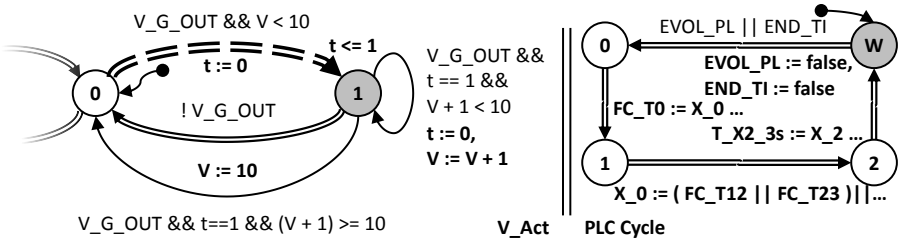
The modifications of the plant and controller models which have been introduced in the previous sub-section remove deadlocks of the plant model but may introduce a deadlock of the controller model when the situation of the GRAFCET is *transient*. It is reminded that a situation of a GRAFCET is the set of active steps at a given date; a situation is *stable* iff no transition of the GRAFCET is fireable from this situation without the change of an input value and *transient* iff at least one transition can be fired without inputs values change (Provost et al. (2011)). Hence, with the modification proposed in 6.2, the PLC cycle model will reach a deadlock state if the current situation is transient (the GRAFCET can evolve even if the values of the inputs remain the same) as illustrated in figures 26 and 27.



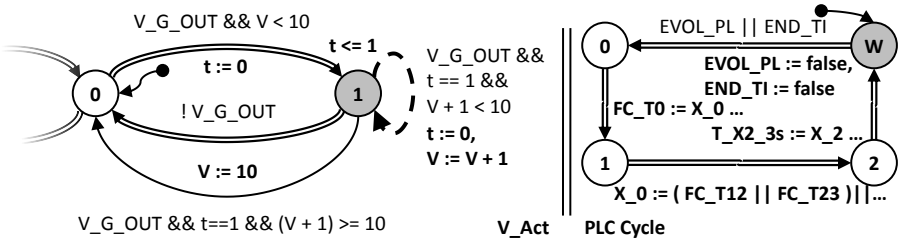
(a) Initial state.



(b) First evolution: starting of the PLC cycle.



(c) Fifth evolution: Evolution of the V_Act model.



(d) Sixth evolution: Timed evolution of the V_Act model.

Figure 25: Evolutions with the modified models.

In those figures, the table on the left gives the values of three variables, the PLC cycle model is given in the center (dotted edges represent edges which are fired) and a part of the GRAFCET specification is represented on the right.

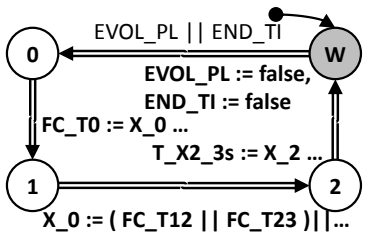
- The initial situation of the GRAFCET is step 23 active (Figure 26); then H_G_IN is *true*. The variable H_IN delivered by the horizontal sensor is assumed *false* as well as EVOL_PL and END_TI.
- During the activity of this step, the push-button is pressed and START becomes *true* as well as EVOL_PL (Figure 27a).
- This provokes one cycle of evolutions of the PLC cycle model (Figure 27b). However, no transition is fireable: t0 is not fireable because step 0 is not active and t23 is not fireable because H_IN is *false*. Hence, no step activity variable changes.
- When H_IN becomes *true* (Figure 27c), EVOL_PL becomes *true* too. Another cycle is possible.
- During this cycle, the step activity variables of steps 23 and 0 are respectively reset and set: this models the firing of transition t23 (27d).

From this situation no more evolution is possible because, as EVOL_PL has been reset during the firing of the first edge in the previous cycle of evolutions, the PLC cycle model is in a deadlock state, except if another input variable changes. This is not the case because, at this moment, the plant model is waiting for the order V_G_OUT (action associated to step 1) from the controller; the model of the closed-loop system has reached a deadlock. This state is meaningless because a real system would evolve as the transition t0 can be fired and consequently must be fired.

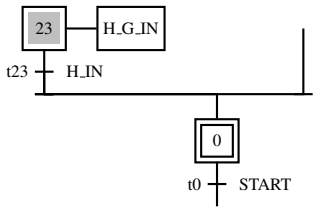
This issue is solved by introducing a new Boolean variable, named stable, *false* if at least one firing condition is true, *true* otherwise. This variable is assigned and used only in the PLC cycle model (Figure 28). More precisely, it is computed in an action of the second edge of the cycle($0 \rightarrow 1$), once all firing conditions are computed. It is used twice to control the following evolutions:

- if stable is not true, the outputs values are not computed and a new cycle is performed (guard of the first edge from the initial state is true),

Variables	Values
START	False
H_IN	False
EVOL_PL	False



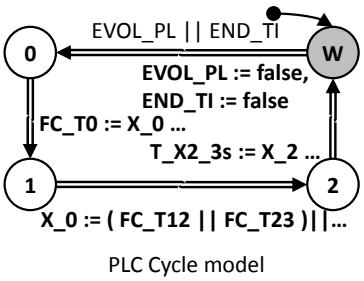
PLC Cycle model

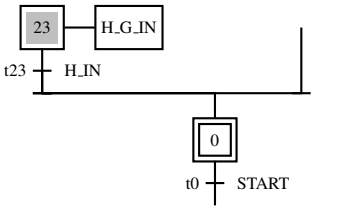


GRAFCET specification

Figure 26: Sequence of evolutions of the closed-loop model with the modified cycle model: initial state.

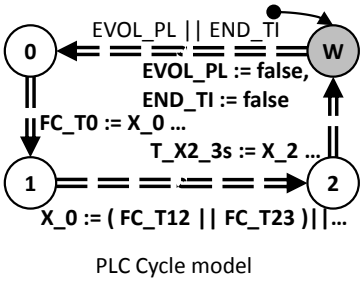
Variables	Values
START	True
H_IN	False
EVOL_PL	True

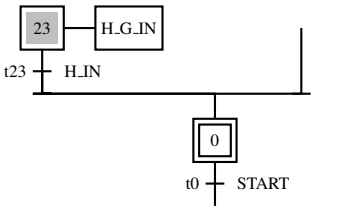




(a) First evolution: START set.

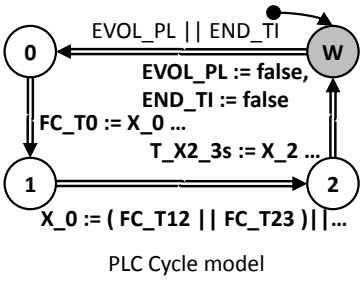
Variables	Values
START	True
H_IN	False
EVOL_PL	False

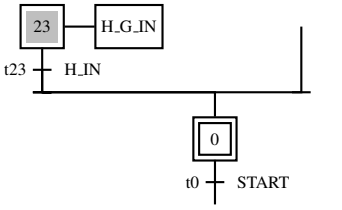




(b) Second evolution: PLC model reacts.

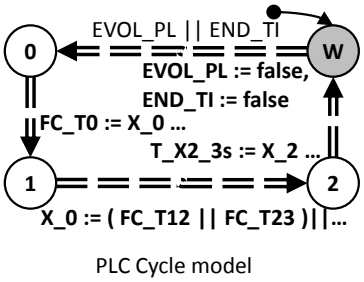
Variables	Values
START	True
H_IN	True
EVOL_PL	True

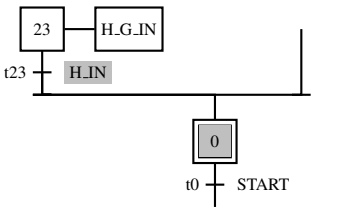




(c) Third evolution: H_IN set.

Variables	Values
START	True
H_IN	True
EVOL_PL	False





(d) Fourth evolution: PLC model reacts.

Figure 27: Sequence of evolutions of the closed-loop model with the modified cycle model.

- otherwise, the outputs values are computed and the evolutions stop.

This behavior is conforming to that described in the GRAFCET standard; outputs are updated only when the situation is stable. Last, it matters to note that an infinite sequence of transient situations may cause an unexpected behavior: the cycle of evolutions never stops. It is assumed in this study that the behavior of the GRAFCET model does not contain such sequence; this assumption can be checked by model-checking the GRAFCET previously, as described in Roussel and Lesage (1996) or Lamperiere-Couffin and Lesage (2000).

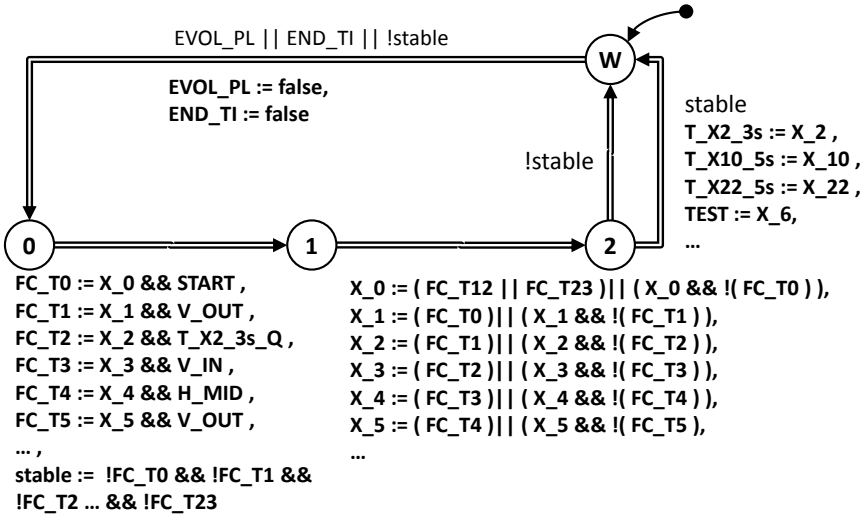
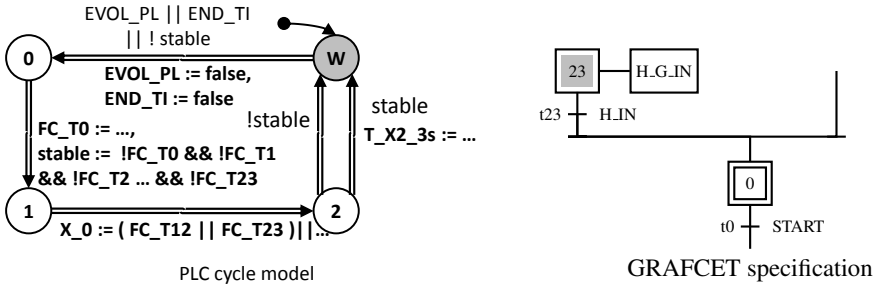


Figure 28: Final version of the PLC cycle model.

Figures 29 and 30 show that this final version of the PLC cycle model (Figure 28) removes the deadlock in case of transient situations. The initial situation is the same that in Figure 27 and stable is *true* because this situation is stable.

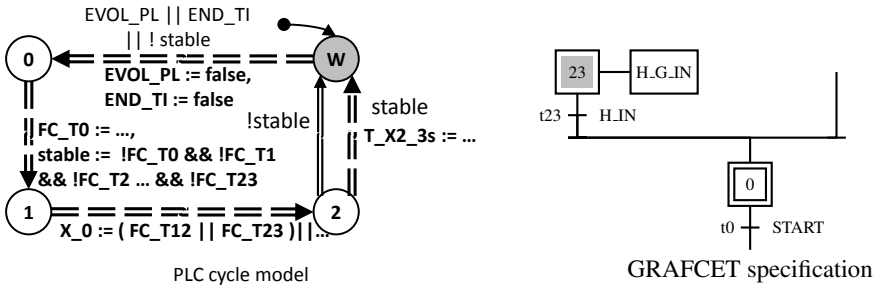
- When START becomes *true*, EVOL_PL becomes *true* too (Figure 29a).
- The PLC cycle model evolves (Figure 29b) and, as no firing condition is true, stable remains *true*. The outputs values are assigned at their previous values and the PLC cycle stops in its initial state.
- When H_IN becomes *true*, EVOL_PL becomes *true* too (Figure 29c) and the PLC cycle model evolves (Figure 29d). As the firing condition FC_23 is now true, transition t23 is fired and stable becomes *false*. No output is assigned and a new cycle begins.
- During this second cycle (Figure 30), FC_0 is computed true and stable *false*; the active step becomes step 1 (referring to Figure 15). No output is assigned and a new cycle begins.
- This third cycle (not represented) will show that the situation is stable (FC_1 requires V_OUT to be *true*). The outputs are assigned and, in particular, H_G_IN becomes *false* and V_G_OUT *true* (action linked to step 1).

Variables	Values
START	True
H_IN	False
EVOL_PL	True
stable	True



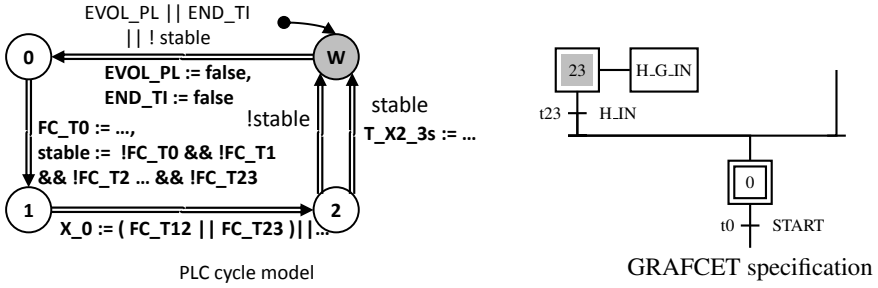
(a) First evolution: START set.

Variables	Values
START	True
H_IN	False
EVOL_PL	False
stable	True



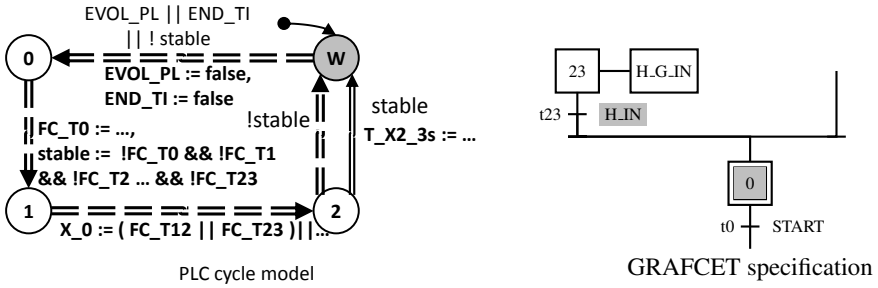
(b) Second evolution: PLC model reacts.

Variables	Values
START	True
H_IN	True
EVOL_PL	True
stable	True



(c) Third evolution: H_IN is set.

Variables	Values
START	True
H_IN	True
EVOL_PL	False
stable	False



(d) Fourth evolution: PLC model reacts.

Figure 29: Sequence of evolutions of the closed-loop model with the final version of the PLC cycle model (part 1).

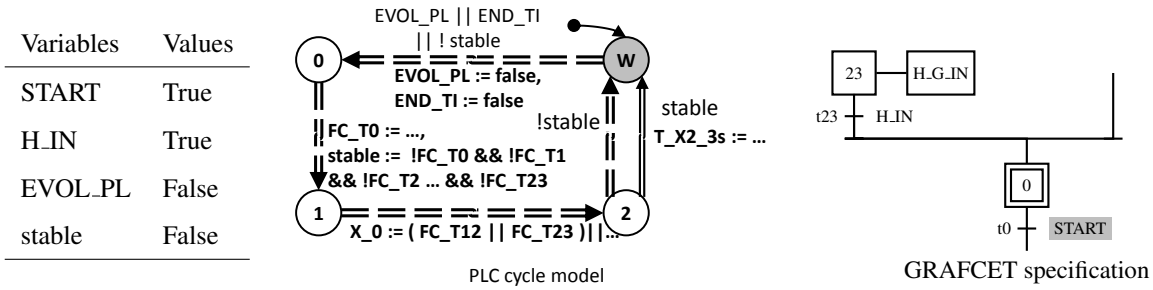


Figure 30: Sequence of evolutions of the closed-loop model with the final version of the PLC cycle model (part 2).

6.4. Final generic models

Figure 31 depicts the two successive modifications of the closed-loop model that have been proposed in this section to remove the deadlocks:

- Two variables EVOL_PL and EVOL_TI have been first defined. The first one is set in every edge of a sensor model where an input variable of the controller is modified; the second one is set in the edge of a timer that represents the end of this timer. These two variables are used in the guard of the first edge of the PLC cycle model; they are reset when this edge is crossed.
- The variable stable has been then defined and a new edge introduced in the PLC cycle model. This variable is used only in this model; it is computed at every cycle, after all firing conditions have been computed, and used in the guards of three edges.

Then, the meaningful model of a given closed-loop system is composed of:

- Instances of generic models of plant components (Figures 13a for the actuators and 32 for the sensors);
- One controller model that is a network of TADD with one model of PLC cycle, instance of the model of Figure 33a, and as many instances of the model of Figure 33b as there are timers in the GRAFCET specification.

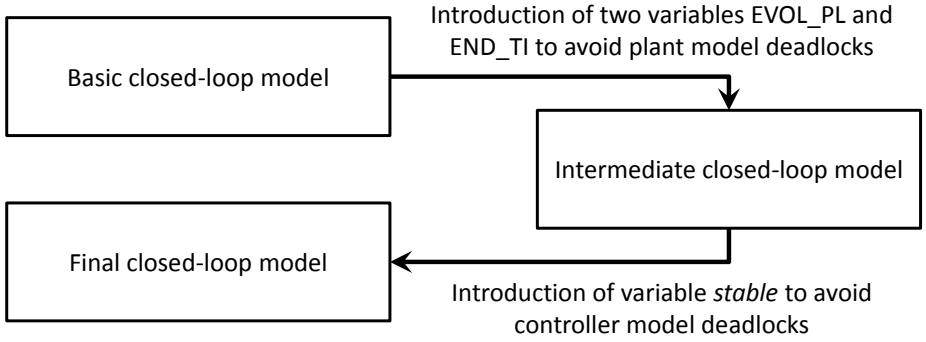


Figure 31: Modifications of the closed-loop model.

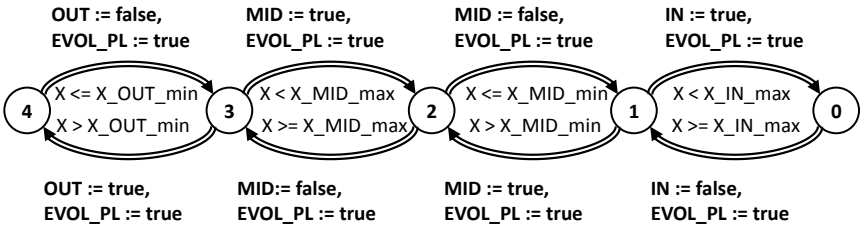
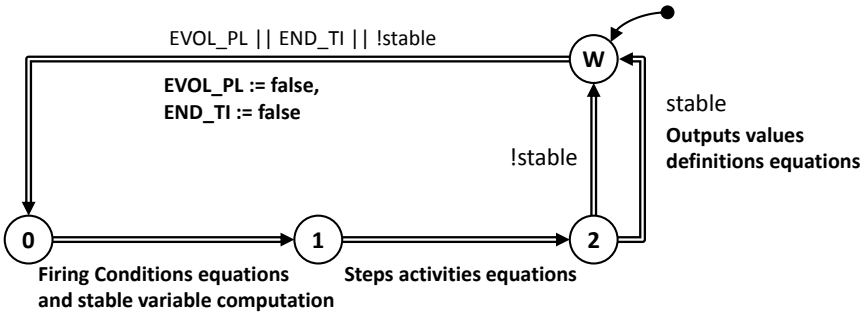
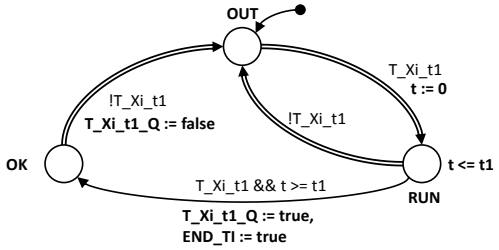


Figure 32: Final version of a generic model for a set of sensors.



(a) Final version of the generic model of the PLC cycle.



(b) Final version of a generic model for a timer.

Figure 33: Final generic controller model.

7. Verification of formal properties

The aim of this section is to show, on the basis of the example of Figure 6, that the closed-loop models developed are tractable, i.e. formal properties can be checked in a short time by an existing timed model-checker. As the TADD formalism is not tool-supported, UPPAAL (Larsen et al. (1997)) has been selected for this study. This tool is widely used for formal verification of timed models and its efficiency, compared to other similar tools , as well as its scalability have been already demonstrated (Larsen et al. (1995); Lindahl et al. (1998)). This choice implies that the models in TADD are to be translated into UPPAAL timed automata prior to verification. The main rules for this translation are only sketched below, for room reasons:

- 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 (Behrmann et al. (2004), what is not the case for the non-urgent edges.

7.1. Environment modeling

Properties verification on the closed-loop model requires the environment (START push-button and Testing device) be modeled because three inputs of the controller (START, TEST OK, TEST KO) are issued from the environment. These models have been built according to the modeling rules that have been defined and are presented at Figures 34 and 35.

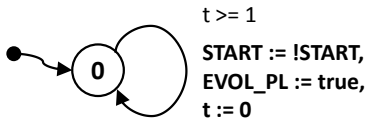


Figure 34: START push button model.

The model of the START push-button is composed of a unique location and a self-loop; this edge is timed but no invariant is associated to the location. Hence, firing of the edge is never forced. The value of the START logical variable is changed when the edge is fired; as there exists concurrency between this edge and the other timed edges, the variable START may change at any moment provided that at least one time unit be elapsed since the previous change. Hence, these changes are uncontrollable events, a feature that is mandatory for verification.

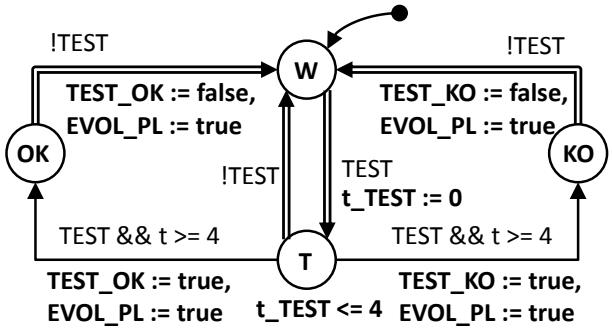


Figure 35: Testing device model.

Figure 35 describes a testing procedure that always lasts 4 time units and provides a Boolean test result (TEST_OK or TEST_KO).

7.2. Properties checking with a faultless plant model

Several kinds of properties can be defined and checked on a formal model (Bérard et al. (2001)). Only one liveness property and one safety property will be considered in this paper which focuses mainly on construction of meaningful models. The selected liveness property is the ability to come back to the initial state of the closed-loop system (step 0 of the GRAFCET active and cylinders at their initial positions (V_IN and H_IN true)), whatever the current state; it is written in the UPPAAL formalism:

$$!(X\_0 \,\&\&\, V\_IN \,\&\&\, H\_IN) \; -->\; (X\_0 \,\&\&\, V\_IN \,\&\&\, H\_IN) \tag{14}$$

where && is the conjunction operator, ! the complement one and $-->$ represents the *Leads to* operator[6].

The selected safety property means that the rod of the vertical cylinder is at its uppermost position during the horizontal movement of the cup; this property is mandatory to avoid collisions with other parts or tools and is stated as follows:

$$A[]\; !((H\_G\_OUT \,||\, H\_G\_IN) \,\&\&\, !V\_IN) \tag{15}$$

Which means that there does not exist any state of the closed-loop model where an order to the horizontal cylinder (H_G_OUT or H_G_IN) is true and the variable V_IN (vertical cylinder at its uppermost position) is false.

Both properties are verified on the presented model. Each verification result is obtained is approximately one second; this shows tractability of the closed-loop model.

7.3. Properties checking with a faulty plant model

7.3.1. Introducing faults in plant components models

Building meaningful models of closed-loop systems where the plant components may fail in different manners is a significant issue and would deserve a full paper. The aim of this section is simply to show on an example that the proposed method can be extended to integrate faults of the plant components and detection of these faults by the controller. This claim will be illustrated by focusing only on one fault: the vertical cylinder gets stuck and one fault detection method: watchdog during the cylinder movements.

Figure 36 shows the new model of the vertical cylinder. A Boolean variable V_S (Vertical cylinder Stuck) is introduced in the guards. This variable is controlled either by the user who can then verify properties in absence or presence of the fault or randomly. When this variable is false (faultless behavior), the model behaves as explained previously; when it is true, some edges (the self-loops that represent the movements of the rod for instance) cannot be fired because their guards are false and other ones (urgent edges from locations 1 or 2) are forced if their source location is active. Hence, assuming that the active location is 1 or 2, the urgent edge from this location is immediately

[6] $p --> q$ is equivalent to $A[]\,(p \; imply \; A <> \; q)$

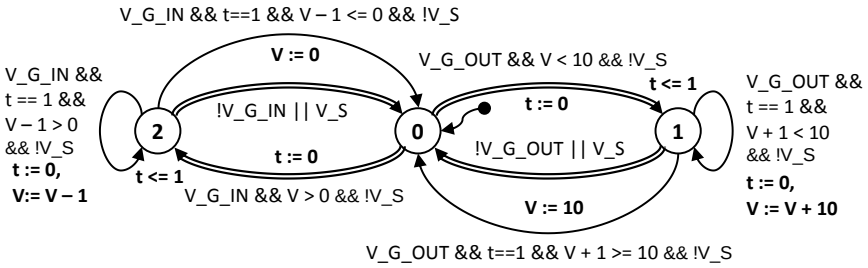
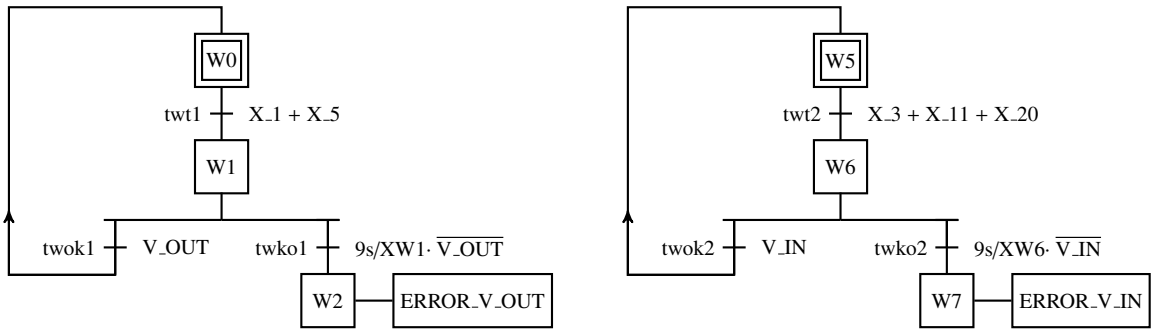


Figure 36: Model of the vertical cylinder with a possible fault V_Act_F.



(a) Watchdog for moving_out.

(b) Watchdog for moving_in.

Figure 37: GRAFCET specification of the watchdogs.

fired when V_S becomes true; the movement is stopped because the cylinder gets stuck. If the active location is 0 (motionless cylinder), no movement can begin.

7.3.2. Detection by the controller

In order to detect the fault of the cylinder, the controller specification must be enhanced with two watchdogs (one for each direction: moving_out, moving_in) that monitor the duration of the movements. These watchdogs can be easily represented in GRAFCET (Figure 37). When a movement is launched (the transition from the initial step is fired), a timer is launched (the active step becomes W1 or W6). From this situation, two evolutions are possible:

- the uppermost or lowermost position is detected before the end of timer, the transition to the initial step is fired and the timer reset;
- or the timer is elapsed before the extreme position is detected, the transition to the step which represents the detection of the fault is fired.

The algebraic representation of these GRAFCET watchdogs is obtained as explained in section 5; an enhanced version of the PLC cycle model, including fault detection, is obtained by adding these equations to those derived from

the GRAFCET of Figure 15.

7.3.3. Checking fault detection ability

Once the enhanced model of plant components, including the possible fault, and the controller, including detection of the fault, constructed, it is possible to formally check whether the controller is able to detect the fault whatever the date it occurs. The formal property to check is then:

$$(V\_S \;\&\&\; (V\_G\_OUT \,\|\, V\_G\_IN)) \; -\!-\!> \; (ERROR\_V\_OUT \,\|\, ERROR\_V\_G\_IN) \tag{16}$$

This means that each time the stuck fault is present with a movement order set, the watchdog must detect their fault.

This property holds on the enhanced model. As previously, this result is obtained in approximately one second; enhancement of the closed-loop model does not increase the verification time.

8. Conclusion

This paper has proposed a novel method to build a timed model of a closed-loop system where all evolutions are meaningful, i.e. correspond to real evolutions of the closed-loop. This model is a network of TADD that communicate through shared variables. The model of the plant is obtained by instantiation of generic components models; meaningless evolutions in this model are avoided by modeling all instantaneous evolutions with urgent edges. On the other side, the model of the controller is composed of a model of the PLC cycle and models of timers; the specified behavior of the controller, in an algebraic form, is included inside the first model. A cycle starts when at least one input of the controller has changed or a timer is finished and stops when no more evolution of the control specification is possible without another input change or end of timer (stability condition); this guarantees the absence of deadlocks and that all evolutions are meaningful. Verification of formal properties with the UPPAAL model-checker has shown that the models built with this approach are tractable.

The first prospect for further work has been sketched when properties verification was addressed. The objective of this contribution was indeed to propose a generic meaningful model of a closed-loop system where the plant is faultless. Such a model is a mandatory prerequisite to address modeling of a closed-loop where the plant can include faults; if the faultless model contain indeed meaningless evolutions, it will not then be possible to distinguish evolutions due to faults and evolutions due to modeling errors in the model with faults. Once this objective met, construction of a plant model that includes not only one type of actuator fault but different types of faults of actuators and sensors must be considered; this will lead to extend the controller model by adding solutions to detect and handle these new faults.

The second prospect is to focus on properties. Trustworthy verification results are obtained only when both properties and the model on which the properties are to be verified are meaningful. A proposal to solve the second

issue has been presented in this paper, assuming a faultless plant. Further work to obtain meaningful properties from the requirements of the closed-loop system is a challenging issue.

References

Alur, R., 1994. A theory of timed automata. Theoretical Computer Science 126, 183–235.

Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., Stursberg, O., 2004. Verification of PLC programs given as sequential function charts. In: LNCS. Vol. 3147. pp. 517–540.

Behrmann, G., David, R., Larsen, K., 2004. A tutorial on UPPAAL. LNCS, Formal Methods for the Design of Real-Time Systems (SFM-RT) 3185, 200–236.

Bel Mokadem, H., Berard, B., Gourcuff, V., De Smet, O., Roussel, J.-M., Oct. 2010. Verification of a timed multitask system with UPPAAL. IEEE Transactions on Automation Science and Engineering 7 (4), 921 – 932.

Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W., 1996. UPPAAL – a tool suite for automatic verification of real-time systems. LNCS 1066, 232–243.

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

Bierel, E., Douchin, O., Lhoste, P., 05 1997. Grafcet : from theory to implementation. European journal of automation (JESA) 31 (3), 543–559.

Campos, J. C., Machado, J., 2009. Pattern-based analysis of automated production systems. In: Proc. of the 13*th* IFAC Symposium on Information Control Problems in Manufacturing, INCOM'09.

David, R., 1995. Grafcet: a powerful tool for specification of logic controllers. Control Systems Technology, IEEE Transactions on 3 (3), 253 –268.

Frey, G., Litz, L., October 2000. Formal methods in PLC programming. In: Proc. of IEEE conference on Systems, Man and Cybernetics. Nashville, USA, pp. 2431–2436.

Gourcuff, V., de Smet, O., Faure, J.-M., 2008. Improving large-sized PLC programs verification using abstractions. In: IFAC World congress.

Hanisch, H.-M., Lobov, A., Lastra, J. L. M., Tuokko, R., Vyatkin, V., 2006. Formal validation of intelligent-automated production systems: towards industrial applications. International Journal of Manufacturing Technology and Management (IJMTM) 8 (1/2/3), 75 – 106.

Hanisch, H.-M., Thieme, J., Luder, A. ajnd Wienhold, O., 1997. Modeling of PLC behavior by means of timed net condition/event systems. In: Proc. of the 6*th* International Conference on Emerging Technologies and Factory Automation Proceedings, ETFA'97. pp. 391 – 396.

IEC 60848, 2002. International Electrotechnical Committee, Grafcet specification language for sequential function charts.

Janowska, A., Janowski, P., 2006. Slicing of timed automata with discrete data. Fundamenta Informaticae 72 (1-3), 181–195.

Johnson, T. L., 2007. Improving automation software dependability : A role for formal methods? Control engineering practice 15 (11), 1403 – 1415.

Lamperiere-Couffin, S., Lesage, J.-J., 2000. Formal verification of the sequential part of plc programs. In: Proc. of the 5*th* Workshop on Discrete Event Systems, WODES'2000. pp. 247–254.

Larsen, K., Pettersson, P., Yi, W., 1997. UPPAAL in a nutshell. Journal on Software Tools for Technology Transfer 1 (1–2), 134–152.

Larsen, K. G., Pettersson, P., Yi, W., Aug. 1995. Model-Checking for Real-Time Systems. LNCS, Proceedings of of Fundamentals of Computation Theory (965), 62–88.

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

Lobov, A., Lastra, J., Tuokko, R., 2005a. Application of UML in plant modeling for model-based verification: UML translation to TNCES. In: proc. of the 3*rd* IEEE International Conference on Industrial Informatics, INDIN'05. pp. 495 – 501.

Lobov, A., Lastra, J., Tuokko, R., 2005b. On controller and plant modeling for model-based formal verification. In: proc. of the 10*th* IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'05. pp. 121–128.

Machado, J., Denis, B., Lesage, J.-J., 2006. Formal verification of industrial controllers: with or without a plant model? In: Proc. of 7*th* Portuguese Conference on Automatic Control, CONTROLO'06. Lisboa Portugal.

Machado, J., Denis, B., Lesage, J.-J., Faure., J.-M., Silva, J. F. D., September 2006. Logic controllers dependability verification using a plant model. In: Proc. of 3*rd* IFAC Workshop on Discrete-Event System Design (DESDes). Rydzyna, Poland, pp. 37–42.

Mader, A., Wupper, H., June 1999. Timed automaton models for simple programmable logic controllers. In: Proc. of 11*th* Euromicro Conference on Real-Time Systems. York, England, pp. 114–122.

Pavlovic, O., Ehrich, H.-D., 2010. Model checking PLC software written in function block diagram. In: Proc. of 3*rd* International Conference on Software Testing, Verification and Validation (ICST). pp. 439 – 448.

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

Philippot, A., Sayed Mouchaweh, M., Carré-Ménétrier, V., jun 2009. Modelling of a discrete manufacturing system by parts of plant. In: 13*th* IFAC Symposium on Information Control Problem in Manufacturing. Moscow.

Provost, J., Roussel, J.-M., Faure, J.-M., 2011. Translating grafcet specifications into mealy machines for conformance test purposes. Control Engineering Practice 19 (9), 947 – 957.

Roussel, J.-M., Lesage, J.-J., 07 1996. Validation and verification of grafcets using state machine. In: Proc. of IMACS-IEEE "CESA'96" IMACS-IEEE "CESA'96" - Lille, France. pp. pp. 758–764.

Schlich, B., Brauer, J., Wernerus, J., Kowalewski, S., 2009. Direct model checking of PLC programs in IL. In: Proc. of the International Workshop on Dependable Control of Discrete Systems, DCDS'09.

Vyatkin, V., Hanisch, H.-M., 1999. A modeling approach for verification of IEC 1499 function blocks using net condition/event systems. In: Proc. of the 7*th* International Conference on Emerging Technologies and Factory Automation Proceedings, ETFA'99. pp. 261 – 270.