Coupling timed plant and controller models with urgent transitions without introducing deadlocks

Matthieu Perin, Jean-Marc Faure

Coupling timed plant and controller models with urgent transitions without introducing deadlocks

Matthieu Perin
LURPA
École Normale Supérieure de Cachan
F-94230 Cachan
perin@lurpa.ens-cachan.fr

Jean-Marc Faure
LURPA
École Normale Supérieure de Cachan
F-94230 Cachan
faure@lurpa.ens-cachan.fr

Abstract

This paper focuses on timed models which represent closed-loop systems composed of a plant and a logic controller. Both the plant and controller models are described in a formalism where urgent transitions are possible, to avoid non-realistic evolutions, and without deadlock when they are separated.

It is first shown that deadlocks may occur in the plant model as soon as both models are coupled. To solve this issue, shared variables which model the changes of the inputs of the controller are defined and introduced in some actions of the plant model as well as in some guards of the controller model; the aim of these variables is to authorize evolutions of the controller only when at least one input has changed. This solution removes the previously pinpointed deadlocks and guarantees the reactivity of the controller.

1 Introduction

Building a controller model to check formal properties is an important concern of the community which has been widely studied [5, 4, 3]. However, as explained in [9], the special need for a plant model to check liveness properties has encouraged works on formal modeling of the plant [17, 15, 13].

However, most of these works have considered untimed models only, whereas the present work use timed models, for both the plant and controller models. In order to build meaningful timed plant models, the need for an urgency semantics has been presented in a previous work [13] and will be re-used. It is also assumed that both the plant and controller models have been built *separately*. This is a possible result of an industrial design process in which two experts –one in control theory, another one in plant modeling– have been requested to build the closed-loop system model.

Unfortunately, when coupling the timed controller and plant models to build the model of the closed-loop system, unexpected deadlocks may appear. The aim of this paper thus is to show the occurrence of these deadlocks on a case study, and then to propose a solution to avoid such a behavior, keeping in mind that both the controller and plant models should be as less modified as possible, by scheduling the evolutions of these models.

The formalism used to model both the plant and the controller is presented in the next section. An example that supports the analysis troughout this paper is introduced in section 3. Sections 4 and 5 focus respectively on the construction of the plant and the controller model. The deadlock issue when coupling the two models is shown in section 6, while a solution to this issue is proposed in section 7 and exemplified in section 8 and 9. Prospects for further works are given in the last section.

2 Formalism used

The models used in this paper will be described by using the formalism of Timed Automata with Discrete Data (TADD) fully described in [8]. This formalism is an extension of the timed automata formalism presented in [1] and is particularly suitable to model urgency and discrete variable evolutions in time. Only the elements if the semantics of TADD that have been used in this work are described below for room saving.

A network of TADD is a set of automata $A_{\mathrm{i}}^{\mathrm{TADD}} = (L_i, l_i^0, V, C, E_i, I_i)$ with $i \in 1, \ldots, n, n \in \mathbb{N}^*$; the sets of clocks ($C$) and variables ($V$) are shared between 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 the active locations vector.
- $\bar{l}^0 = (l_1^0, l_2^0, \ldots, l_n^0)$ is the initial locations vector.
- V is a set of common discrete variables; let $v \in V$ be a variable, $\tilde{v}$ is the variable valuation and $\tilde{V}$ is the set of all the variables valuations.

- C is a set of common 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.
- $E = \bigcup_i E_i$ is the set of the edges of the network. Each edge $e = (l, g, t, u, a, 2^C, l')$ is going from a source location $l \in L_i$ to a target location $l' \in L_i$, with a guard g as a boolean expression over the set of variables V, a timed constraint t over the set of clocks C, and an action a setting potential new values for variables of V. The term 2^C is introduced for the reset of clocks: each clock may be reset or not. $r \subseteq C$ is defined as the set of the clocks to be reset in a specific edge. $u \in \{true, false\}$ is the *urgent attribute* that defines the edge as an *urgent edge*; no time constraint is allowed (i.e. t = true, the always satisfied time constraint) when the urgent attribute is true.
- $I(\bar{l}) = \bigwedge_j I_j(l_i)$ is defined as the common invariant function over the set of clocks C.

A network of TADD may evolve according to two semantics:

The edge firing semantics is possible for each edge if the source location is active, the guard is validated (i.e. the variables valuations of $\tilde{V}$ satisfy the guard), the time constraint is satisfied (i.e. $\tilde{C}$ satisfies the time constraint) and the invariant of the target location is validated by the new clocks valuations $\tilde{C'}$ including the clocks resets of the edge. The firing changes the active location from the source location to the target one, the set $\tilde{V'}$ of the new variables valuations is deduced from the assignment action a of the edge. Clocks valuations remain unchanged except for clocks that belong to r.

The time evolution semantics makes all the clocks valuations increase by the same amount. The active location of every automaton of the network is unchanged and $\tilde{C}$ becomes $\tilde{C'}$ iff:

- The valuations $\tilde{C}$ and $\tilde{C'}$ satisfy the invariant of the current active locations.
- *No urgent edge may be fired from the current active locations.*

These two semantics are exemplified on the network of TADD given in figure 1. This figure depicts a network of two TADD using one clock T, one Boolean variable a, one integer variable X, and composed of:

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

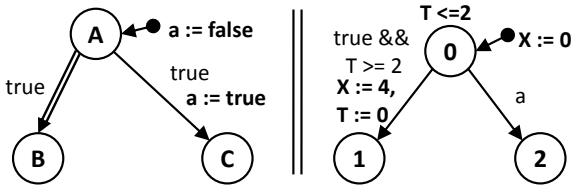

Figure 1: A Network of Timed Automata with Discrete Data (TADD).

The initial state (locations A and 0 are active, variables and clocks have their initial valuations), two evolutions are possible:

(A,0) → (B,0), as the source locations vector is active and the guard (true) is always satisfied. The next active locations vector will be (B,0), and the valuations of both variables and clocks remain unchanged (no assignment and no clock reset).

(A,0) → (C,0), as the source locations vector is active and the guard is satisfied. The next active locations vector will be (C,0), and the valuations of X and T remain unchanged but the Boolean variable a will become true.

No evolution of the right-hand side model is possible because the guard of the edge 0 → 2 is not satisfied and the time constraint of the edge 0 → 1 cannot be satisfied as the edge A → B may be fired and is urgent (thus preventing any evolution based on the time semantics. If the edge A → C is fired, two other evolutions are thus possible:

(C,0) → (C,1), by using the time evolution semantics to rise the valuation of the clock T from 0 to 2. The next active locations vector will be (C,1), and the valuation of a remains unchanged (true), the variable X is set to 4 and the clock T is reset.

(C,0) $\rightarrow$ **(C,2)**, as a is true and the guard is hence satisfied. The new active locations vector will be (C,2) and all the valuations remain unchanged (neither variable assignment nor clock reset on the fired edge).

It matters to highlight that urgent edges have no priority over non-urgent edges whose firing does not depend on time evolution. Similarly, these latter edges have no priority over non-urgent edges whose firing depends on time evolution.

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

3 Example used

The issue of this paper will be exemplified on an automatic door system for an oven presented in figure 2. A PLC[1] executing a code specified in Grafcet [6] is controlling, through its outputs, a plant composed of:

- A man-machine interface with:
  - A start push-button to set/reset the input variable START.
  - A warning light controlled by the output variable LIGHT.
  - A security buzzer controlled by the output variable BUZZER.
- An hydraulic cylinder coupled with its dedicated 5/3 preactuator, used to move the door.
- A set of two sensors detecting two positions of the door (door open and door closed) and sending the input variables FREE and SEAL.
- Hydraulic locking devices used to ensure the sealing of the door, and controlled by the output variable LOCK.

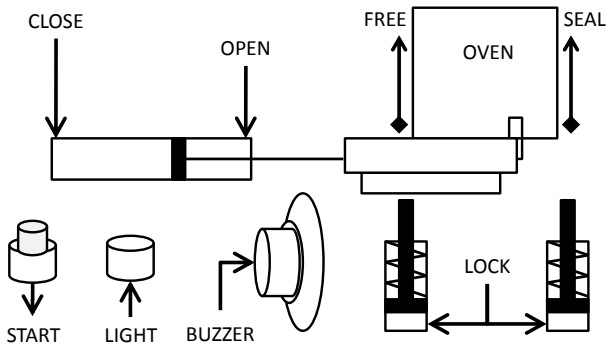

Figure 2: Plant of the example with the I/O of the controller.

[1] Programmable Logic Controller

The desired behavior is fully described in the Grafcet specification of the controller presented in section 5; it is yet sketched here: the user pushes the start button, which launches the automatic cycle, and the light turns on as an acknowledgment. Depending on the door position (FREE or SEAL), the door is moved (CLOSE or OPEN). If the door was in the SEAL position before being moved, the locking is removed (LOCK becomes false); if not, the locking is set at the end of the movement (door in the SEAL position). During the whole cycle, the buzzer and the light are active for security purposes.

4 Plant model

Several methods have been proposed to build untimed [10, 17] or timed [15, 13] plant models. The timed plant model presented in this paper has been built according to [13]; in this approach, a plant model is a network of TADD where every automaton describes the behavior of a component of the real plant (actuator, sensor). Moreover, this modular method permits to build plant models whose all evolutions are realistic, i.e. where no evolution describes a behavior that cannot be observed on the real plant, assuming that it is faultless.

To meet this objective, two kinds of evolutions of the real plant are to be defined, according to a time consumption criterion:

- *Instantaneous* evolutions, such as the detection of a part by a sensor, are evolutions whose duration is assumed to be negligible.
- *Timed* evolutions, such as the movement of a mechanical part, consume time.

Instantaneous evolutions will be modeled by urgent edges whilst non-urgent edges will be selected to model timed evolutions. It has been shown in [13] that this modeling choice ensures that all evolutions of the plant model are realistic.

The application of this method to the example provides the following models of plant components:

- The model of the hydraulic actuator is given in figure 3. The initial location models a stop state; location 1, resp. 2, models the outgoing, resp. ingoing, movement of the actuator rod. The self-loop edge on location 1, resp. 2, models the increase, resp. decrease, of the variable X (position of the extremity of the rod), until the physical limit (X = 50, resp. X = 0, in this example) is reached; the active location becomes the initial one once any of these limits is reached. The urgent edges represent the modifications of the actuator state when the orders OPEN and CLOSE are set/reset.
- The model of the set of two sensors that compute the input variables FREE and SEAL is given in figure 4. In the initial location, only the first variable is true

(the door is detected open); the other locations correspond to a door detected between its left and right positions (location 1) or totally closed (location 2). Only urgent edges have been selected because any change of the state of a sensor has been assumed instantaneous.

The plant model is then a network of TADD that includes these two automata as well as other automata which model the behavior of the other components of the plant (locking devices, buzzer,etc.). These latter models have been built in a similar manner but will not be presented in this paper for room reasons.

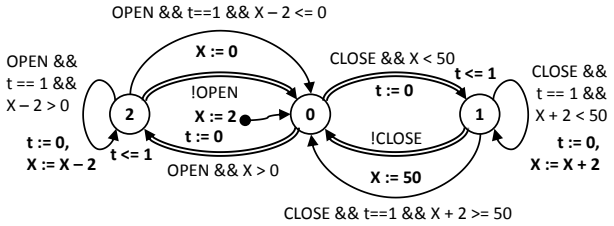

Figure 3: Model of the hydraulic actuator.

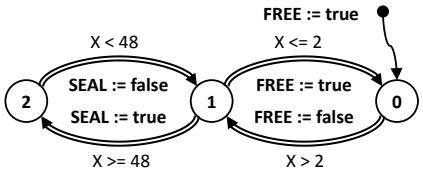

Figure 4: Model of the set of sensors related to the position of the door.

5 Controller model

The plant is assumed to be controlled by a PLC that executes cyclically a code whose specification is described in the standardized language termed Grafcet [6]. This specification (Figure 5) includes timers launched by the activation of some steps (see the transition conditions of t40 and t42 in figure 5).

Several methods have been proposed to translate a control model in a standardized language into a formal model. Some of them ([2] and [19] for example) start from a model in SFC[2] language [7]; however this language is an implementation and not a specification language (the interested reader is referred to [16] for further details on the semantic differences between Grafcet and SFC); this explains why these approaches were not selected for this work. On the opposite, [11] presents a method to translate an *untimed* Grafcet specification into three sets of algebraic equations:

- The first set is termed firing conditions computation. For each transition, the value of a Boolean variable, named firing condition, is calculated from the values of the activity variables of the steps that precede this transition and the value of the transition condition. If this variable is true, the transition must be fired.
- The second set is aiming at computing the values of the step activity variables once the transitions that must be fired have been fired. A step activity variable is a Boolean variable noted X_i, i being the number of the step, in what follows.
- The third set computes the new values of the output variables from the values of the step activity variables.

This method is the starting point to construct the controller model. However, to be able to deal with timed specifications, formal models of timers are to be introduced,

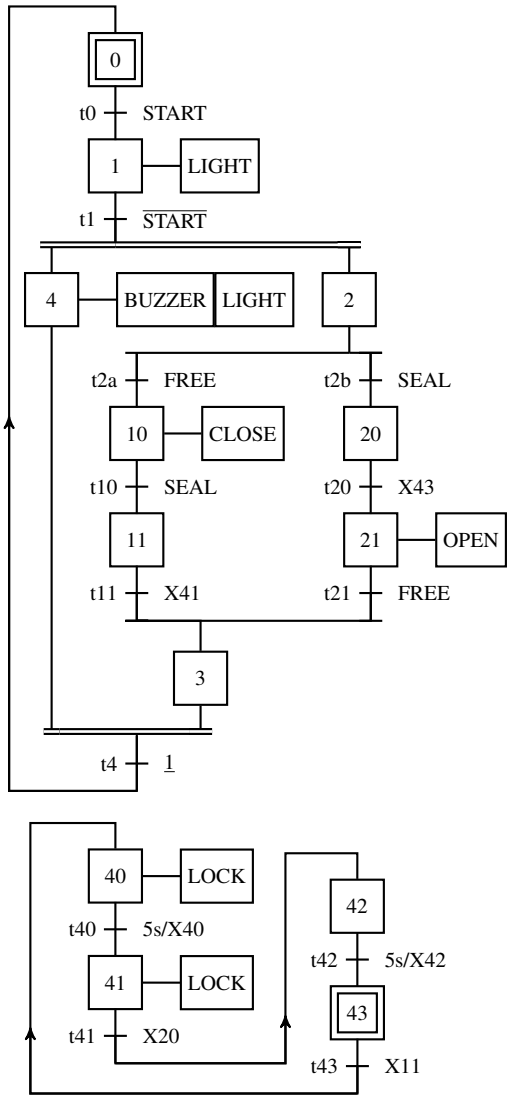

Figure 5: Grafcet specification of the automatic door.

[2]Sequential Function Chart

as proposed in [12] and [18]. Figure 6 shows the formal model of the timer launched by the activation of step 40 of the Grafcet of figure 5. The communication between this model and the Grafcet is ensured by two variables:

T_X40_5s is a Boolean variable set by the Grafcet in order to launch the timer. When it is reset, the timer model comes back to its initial location.

T_X40_5s_Q is a Boolean variable set by the timer model when the waiting time is elapsed. It is used in the transition condition of t40. Once set, it is reset only when T_X40_5s is reset.

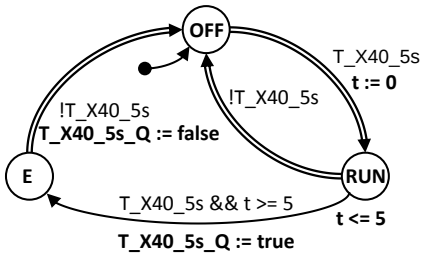

Figure 6: Timer model launched by the activation of the step 40.

The model of the controller can then be stated on the basis of the PLC cycle (Figure 7). This cycle includes three phases: inputs reading, execution of the control code and outputs updating. The execution of the control code is itself decomposed in three phases where the three sets of algebraic equations are computed sequentially. It is not necessary to represent the first and third phases of the PLC cycle (inputs reading and outputs updating) in the formal model of the controller because the inputs/outputs of the Grafcet are modeled as Boolean variables that are shared by the controller and plant models.

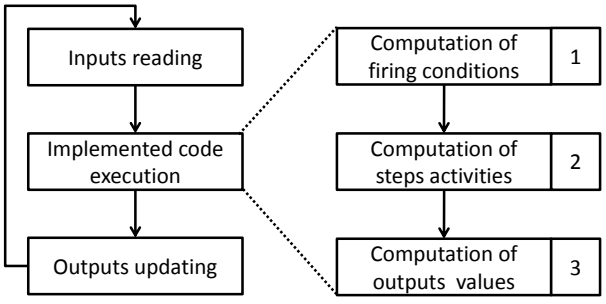

Figure 7: A PLC cycle.

The possibility of *transient* evolutions ([6], [16]) in the Grafcet, due for instance to the always satisfied transition condition of transition t4, imposes that the output values must not be updated as long as the Grafcet has not reached a stable situation. The computation of the output values must then be skipped if the situation is not stable. A Boolean variable noted *stable* is thus introduced and evaluated once all firing conditions have been computed; it is reset if at least one firing condition is true, and set otherwise. If *stable* is true, no more transitions may be fired; the situation is detected stable and the outputs can be updated. If *stable* is false, at least one transition can be fired; the computation of the output values has to be skipped, i.e. the output values remain unchanged.

The figure 8 presents the model of the controller built to describe this behavior:

- Three locations are used.
- For each set of equations of the Grafcet algebraic model, an edge is created with this set of equations as its actions.
- The computation of the *stable* variable is added to the actions of the second edge.
- The edge which corresponds to the output values computation has a guard *stable* to prevent these values be updated when the situation of the Grafcet is not stable.
- An edge with no action allows to skip the output values computation as long as *stable* is false.
- This model is initialized with only the activity variables of the initial steps true, only the values of the outputs set by the actions associated with the initial steps true and *stable* false because the initial situation may be transient for some values of the inputs.
- All edges are urgent because the time cycle of the PLC is supposed to be negligible compared to the duration of the movements of the components of the plant.

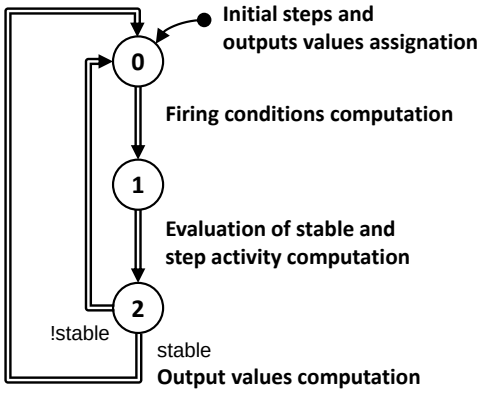

Figure 8: Generic model of controller.

Figure 9 represents the particular model of the controller for the example.

6 Deadlock issue when coupling the two models

A simple approach to build the model of the closed-loop system consists in putting in parallel the plant model

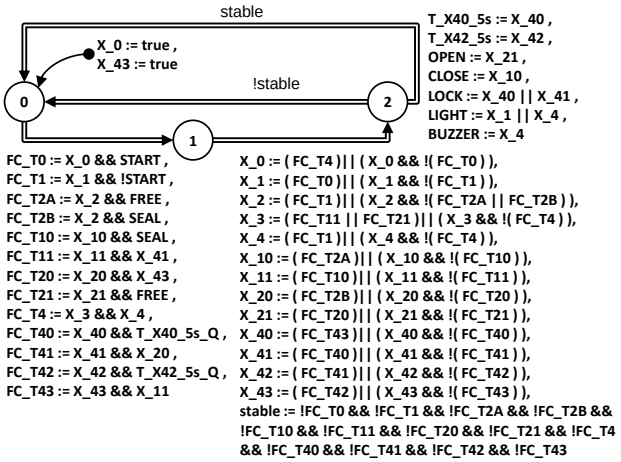

Figure 9: Controller model of the example

and the controller model in a network of TADD. The obtained model can then be simulated to check its behavior. A part of the evolutions of this simulation is described in figures 10, 11, 12 and 13, the oven door is supposed to be initially open (the FREE variable is true, SEAL is false). Only a reduced version of the controller model, a partial representation of the Grafcet specification, and the actuator model are represented. The active locations and the active steps of the Grafcet are grayed, the output which are set are also grayed and the fired edges are dotted for readability purposes.

Figure 10 shows the state once the operator has pushed the start button; step 1 of the Grafcet is active. The controller model is supposed to have reacted (setting LIGHT to true) and returned to its initial location, the actuator model has not evolved yet and is in its initial location.

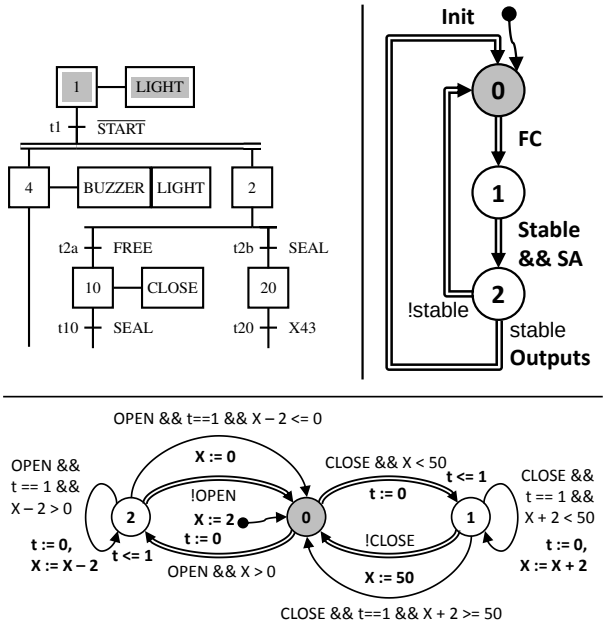

Figure 10: State once the operator has pushed the start button.

Figure 11 represents the state once the operator has released the button, resetting the variable START. The transition condition of t1 is satisfied; then this transition must be fired and steps 2 and 4 activated. Then, the controller model evolves three times:

- The firing conditions are computed; FC_T1 becomes true.
- The step activity variables are computed; X_1 becomes false, X_2 and X_4 true. The variable *stable* is computed false as FC_T1 is true.
- As *stable* is false, the output variables values are not updated and the controller model comes back to its initial location.

Moreover, as both output variables CLOSE and OPEN have not become true, the actuator model cannot evolve and remains in its initial location. As FREE is true, the situation in which the steps 2 and 4 are active is not stable. The controller evolves until the stable situation (steps 10 and 4 active) is reached (**Figure 12**). The outputs BUZZER, LIGHT and CLOSE are set.

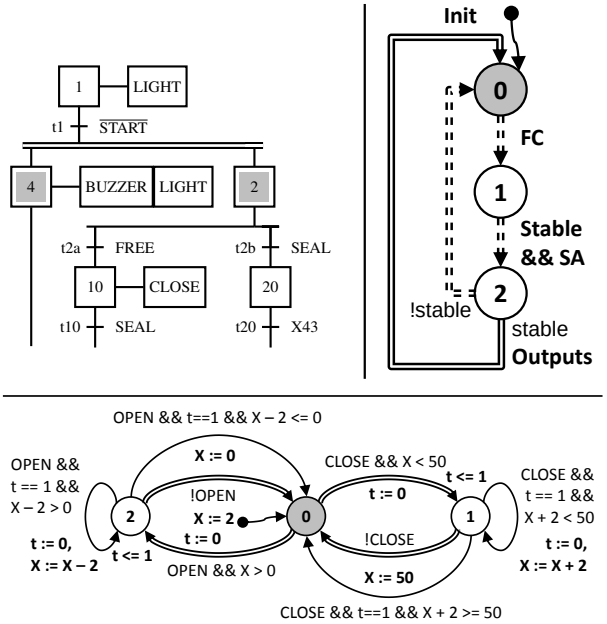

Figure 11: The operator releases the start button, the controller reacts.

Figure 13 shows the last evolution. As CLOSE is true, the model of the actuator can evolve from location 0 to location 1. More precisely, two urgent transitions may be fired, one in the controller model and one in the actuator model. It is assumed that this latter is fired; then the clock t is reset.

This state is a deadlock for the plant model. Two edges are starting from the active location of the actuator model but none can be fired. The self-loop on location 1 requires an evolution of the clock, which is not possible because at

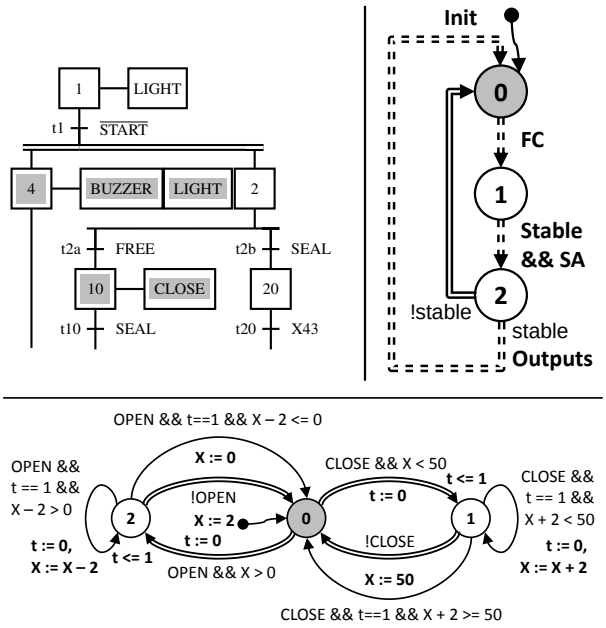

Figure 12: The controller model evolves again until *stable* is set and outputs updated.

least one urgent edge can be fired in the controller model. The urgent edge from location 1 to location 0 cannot be fired either, because CLOSE is true, and will remain true because the Grafcet will not change the value of this variable -step 10 will stay active- if SEAL remains false (the door remains open).

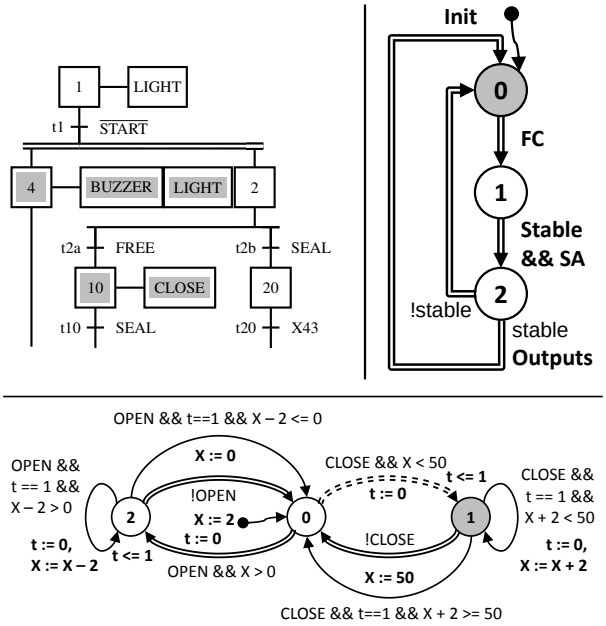

Figure 13: Last evolution: the actuator model evolves.

7 Analysis and solution proposed

All the edges of the controller model are urgent because the time cycle of the PLC (a few milliseconds) was considered to be negligible compared to the typical times of the plant (a few seconds or at least tenths of seconds). However, this modeling provokes deadlocks in the plant model when an edge whose firing depends on time evolution has to be fired.

As it is not possible to remove time-constrained edges in the plant model if a realistic timed model is expected, a solution to this issue might be to introduce such edges in the controller to model the processing time. Nevertheless, as the time step of all clocks is the same in the modeling formalism, this solution would imply that a lot of clocks steps of the controller clock would occur before two successive input changes and would surely lead to combinatory explosion during the formal analysis of non-trivial models. This is the reason why this solution was not selected.

The urgent edges of the controller model are then to be kept but the evolutions of this model must be limited to only three cases:

- The value of an input variable has changed.
- The variable that represents the end of a timer has been set.
- The situation of the Grafcet is not yet stable.

8 Application of the solution

To model the first two cases, two Boolean variables EVOL_PL (evolution of the plant) and EVOL_TI (evolution of a timer) which act as flags must be introduced:

EVOL_PL is set in every edge of the plant model where an input variable of the controller is assigned. As the input variables are assigned in edges of sensors models, EVOL_PL is set in such edges. It is reset by the controller model once the firing conditions have been computed.

EVOL_TI is set in every timer model edge where the variable that represents the end of this timer is set. It is reset by the controller model once the firing conditions have been computed.

The modified versions of the models of the set of sensors of figure 4 and of the timer of figure 6 are given in figures 14 and 15.

The controller model is modified as follows (Figure 16):

- The guard of the edge from location 0 to location 1 becomes !stable || EVOL_PL || EVOL_TI; hence a cycle of this model is possible only when at least one of these three conditions is true, which is expected.

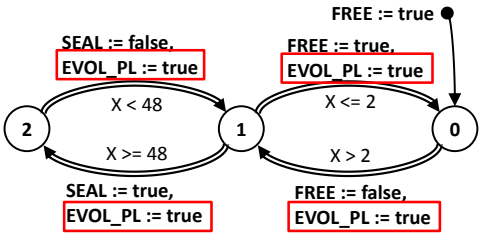

Figure 14: Modified version of the model of the set of sensors.

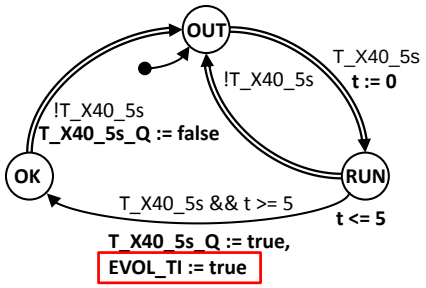

Figure 15: Modified version of the timer model related to the activity of the step 40.

- Two actions that reset both variables EVOL_PL and EVOL_TI are added to this edge; then, two successive cycles are possible iff one of these variables becomes true during the current cycle.

This modeling guarantees that every input change and end of timer will be detected while avoiding deadlocks of the plant model, as shown in the next section.

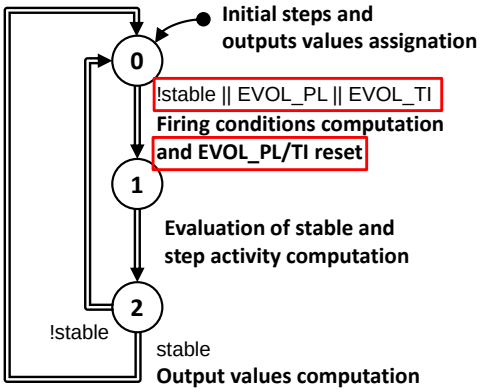

Figure 16: Final version of the model of the controller.

9 Evolutions with the modified models

Figure 17 shows the state once the operator has released the start button (previously described in Figure 11). As the input variable START has been reset, EVOL_PL becomes true and a cycle of the controller is possible. The new situation of the Grafcet (steps 2 and 4 are active) is not stable; hence a new cycle is possible.

Figure 18 shows the last evolution of the controller model and the following evolutions of the actuator model (evolutions previously described in figures 12 and 13). As the new situation (steps 4 and 10 are active) is stable, the cycle of the controller stops in location 0 (EVOL_PL and EVOL_TI are false and *stable* is true). Then the actuator model evolves to its location 1. It matters to underline that only this evolution is possible; there is no concurrency between the evolutions of the plant and the controller. As no more urgent edges can be fired, the time semantics is used and the actuator evolves again by using the timed-constrained self-loop edge.

The previous deadlock is then removed and the plant model will evolve normally (several time-constrained self-loops on location 1 before coming back to location 0 when the door is closed) until one of the inputs of the controller, e.g. the variable SEAL that models the end of the movement of the door, has changed.

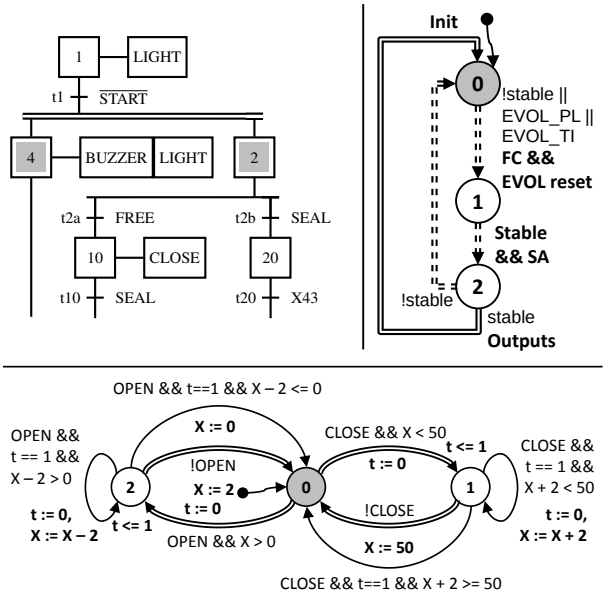

Figure 17: The operator releases the start button, the controller reacts.

10 Conclusion

This paper has shown that coupling timed plant and controller models, that have been built independently and that include urgent transitions, may lead to a deadlock state. To solve this issue, the evolutions of the two models need to be scheduled by means of two variables that model the changes of the input variables of the controller and the end of timers.

As the TADD formalism is not tool-supported, the models presented in this paper have been translated into the formalism of the UPPAAL suite, according to the translation rules presented in [14]. Formal verification of these models by the model-checker of this suite has per-

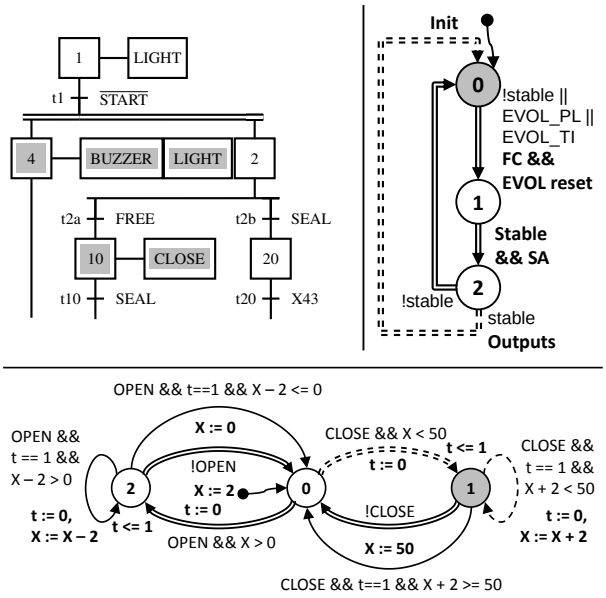

Figure 18: The controller model has reached a stable situation and stops, the actuator model evolves twice.

mitted to validate this proposal; no deadlock state was detected.

Future work is aiming at introducing faulty behaviors in the plant model and fault detection mechanisms in the controller model in order to analyze more complex models of closed-loop systems.

References

[1] R. Alur. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] N. Bauer, S. Engell, R. Huuck, S. Lohmann, B. Lukoschus, M. Remelhe, and O. Stursberg. Verification of PLC programs given as sequential function charts. In *LNCS*, volume 3147, pages 517–540, 2004.

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

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

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

[6] IEC 60848. *International Electrotechnical Committee, Grafcet specification language for sequential function charts*, 2002.

[7] IEC 61131. *International Electrotechnical Committee, Programmable controllers - Programming languages*, 1999.

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

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

[10] J. Machado, B. Denis, and J.-J. Lesage. A generic approach to build plant models for DES verification purposes,. In *Proc. of the 8th International Workshop on Discrete Event Systems (WODES)*, Ann Arbor (Michigan), USA, July 2006.

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

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

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

[14] M. Perin and J.-M. Faure. Building meaningful timed models of closed-loop DES for verification purposes. *Control Engineering Practice*, *pending for publication with doi: 10.1016/j.conengprac.2012.05.002*.

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

[16] J. Provost, J.-M. Roussel, and J.-M. Faure. Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9):947 – 957, 2011.

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

[18] O. Rossi and P. Schnoebelen. Formal modeling of timed function blocks for the automatic verification of ladder diagram programs. In *Proc. of the 4th International Conference Automation of Mixed Processes (ADPM'00)*, pages pp. 177–182, 2000.

[19] O. Stursberg and S. Lohmann. Analysis of logic controllers by transformation of SFC into timed automata. In *Proc of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, pages 7720–7725, 2005.