Building Meaningful Timed Plant Models for Verification Purposes

Matthieu Perin, Jean-Marc Faure

Building Meaningful Timed Plant Models for Verification Purposes

Matthieu Perin [*,**] **Jean-Marc Faure** [*,***]

[*] *Laboratoire Universitaire de Recherche en Production Automatisée, École Normale Supérieure de Cachan, F-94230 Cachan (e-mail: {perin, faure}@lurpa.ens-cachan.fr)*

[**] *Dassault Systemes, Delmia R&D services, F-78140 Vélizy-Villacoublay (e-mail : matthieu.perin@3ds.com)*

[***] *Institut Supérieur de Mécanique de Paris, F-93400 Saint-Ouen (e-mail: jean-marc.faure@supmeca.fr)*

Abstract:
This paper presents a method to build a formal model of a plant, in the form of a network of timed automata, to be used for model-based verification of controllers. To ensure re-usability, this model is built by instantiation of generic components models. When the instantiated components models are assembled, spurious evolutions leading to states which do not represent the real behavior of the plant, can occur, owing to the rich semantics of the modeling formalism. Then a modeling strategy is proposed in order to discard these evolutions so as to reduce the state space of the plant model to the only meaningful states. The method is exemplified and discussed on a small case study.

Keywords: Plant modeling, Timed models, UPPAAL, Concurrent evolutions.

1. INTRODUCTION

Model checking is a promising formal verification method to design more dependable controllers. This paper focuses only on timed model-checking of controllers, i.e. formal verification of time-dependant formal properties. Two main approaches have been defined to check properties of a controller (Frey and Litz [2000]):

- Non-model-based model-checking.
- Model-based model-checking.

In the first approach, properties are checked on a formal model of the controller in isolation; in the second one (figure 1), they are checked on a model of the closed loop system which encompasses both the controller and the controlled system, termed plant. This work considers only the model-based approach, which in particular permits to verify some liveness properties that cannot be proved with the other approach (Machado et al. [2006]), and focuses on plant model construction.

To ease re-usability, the plant model can be built by assembling instances of generic components models which represent the behaviors of typical plant components, like electric and pneumatic actuators, sensors. The state space of the whole plant model is then obtained by composition of the state spaces of these components instances. The first aim of this paper is to show that, when this approach is used without any care, the overall state space of the plant model can include meaningless states, i.e. states which do not represent real behaviors of the plant. To prevent from erroneous proof results, the evolutions that lead to these meaningless states must be forbidden. Proposing modeling

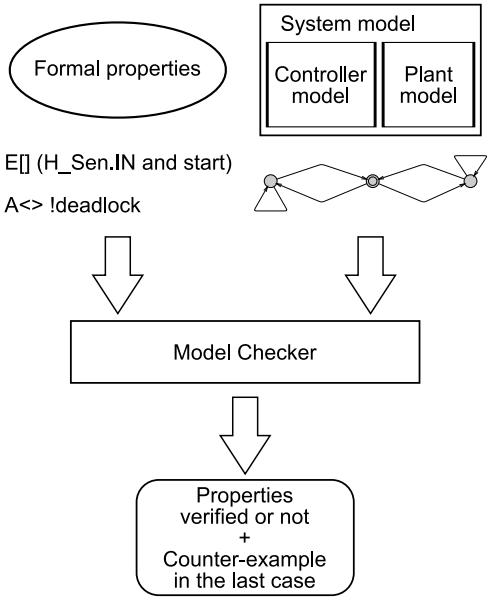

Fig. 1. Model-based model-checking principle

rules to forbid these spurious evolutions is the second aim of this work.

Hence, this paper presents a method to build, from generic components models, a plant model that includes only states representing real behaviors of the physical plant. The semantics of the communicating timed automata, the formal frame that underlies the selected model-checker, is recalled in the next section. Section 3 presents some generic components models and shows how these models are used to construct a plant model. The issue of spurious evolutions is illustrated and formalized in section 4 while section 5 proposes modeling rules to avoid these evolutions.

2. COMMUNICATING TIMED AUTOMATA SEMANTICS

One of the most used formalisms to describe the evolutions of timed discrete event systems is the formalism of timed automata proposed by Alur [1994]. In this work, the UPPAAL model-checker, whose formalism is based on the networks of timed automata, will be used as a widely recognized toolbox over timed systems design, simulation and verification. The reader can view Larsen et al. [1997] and Behrmann et al. [2002] for more details on UPPAAL.

In order to explain in section 4 why the semantic possibilities of the used formalism may give rise to spurious evolutions in a plant model, the semantics of timed automata and of network of timed automata are reminded below.

2.1 Timed automaton formalism

Definition of a timed automaton: A timed automaton is a tuple (L, l_0, C, A, E, I), where:

- L is a set of locations.
- $l_0 \in L$ is the initial location.
- C is the set of clocks.
- A is a set of actions.
- $E \subseteq L \times A \times N(C) \times 2^C \times L$ is a set of edges between two locations l,l'∈L, with an action $a \in A$, a guard $g \in N(C)$ and a set of clocks to be reset $r \subseteq C$.
- $I: L \rightarrow N(C)$ assigns invariants to locations.

$N(C)$ is the notation for the set of conjunctions over simple conditions of the form x # c or x - y # c, where x, y $\in C$, c $\in \mathbb{N}$ and #$\in \{<, \leq, =, \geq, >\}$.

The semantics of a timed automaton is given below, according to the following definitions:

- $v: C \rightarrow \mathbb{R}^+$ is a clock valuation.
- V is the set of clock valuations.
- $v_0(x) = 0$ for all $x \in C$.
- $v \models I(l)$ means that v satisfies the invariant of location l.

Timed automaton semantics: Let (L, L_0, C, A, E, I) be a timed automaton, the semantics is defined as a labeled transition system $\langle S, S_0, \rightarrow \rangle$ where $S \subseteq L \times V$ is the set of states, $s_0 = (l_0, v_0)$ is the initial state, and $\rightarrow \subseteq S \times \{\mathbb{R}^+ \cup A\} \times S$ is the set of transition relations such that :

$$(l, v) \xrightarrow{d} (l, v + d) \text{ if } \forall d', \quad 0 \leq d' \leq d \Rightarrow v + d' \models I(l) \tag{1}$$

$$(l, v) \xrightarrow{a} (l', v') \text{ if } \exists\, e = (l, a, g, r, l') \in E, \quad v \models g,\ v' = [r \mapsto 0]v, \text{ and } v' \models I(l') \tag{2}$$

Where, for d $\in \mathbb{R}^+$, $v + d$ maps each clock x $\in C$ to $v(x) + d$, and $[r \mapsto 0]v$ stands for the clock valuation mapping each clock of r to 0 and agrees with v over $C \backslash r$ where $\backslash$ stands for the set difference.

Hence, two evolutions of a timed automaton are possible: an increase of the clocks so that the active location does not change, *or* the firing of an edge with a new active location.

2.2 Network of timed automata formalism

A network of timed automata is a set of n timed automata with common actions and clocks, each one defined as $A_i = (L_i, l_i^0, C, A, E_i, I_i)$ with $i \in \mathbb{N}$ and $1 \leq i \leq n$; interactions between these automata are limited to shared variables and communication channels. The following precisions are added:

- $\bar{l} = (l_1, l_2, \ldots, l_n)$ 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.
- $I(\bar{l}) = \bigwedge_i I_i(l_i)$ is defined as the common invariant function.
- $\bar{l}[l_i \mapsto l_i']$ stands for the change of the i$^{\text{th}}$ value of vector $\bar{l}$ from l_i to l_i'.
- $\tau \in A$ is an action on shared variables.
- $\chi!, \chi? \in A$ are actions (emission and reception) over the communication channel χ.

Network of timed automata semantics: Let $\{A_i\}$, with $i \in \mathbb{N}$ and $1 \leq i \leq n$, be a network of n timed automata. The semantics is defined as a transition system $\langle S, S_0, \rightarrow \rangle$ where $S = (L_1, L_2, \ldots, L_n) \times V$ is the set of states, $s_0 = (\bar{l}_0, v_0)$ is the initial state, and $\rightarrow \subseteq S \times S$ is the set of transition relations such that:

$$(\bar{l}, v) \rightarrow (\bar{l}, v + d) \text{ if } \forall d' : \quad 0 \leq d' \leq d \Rightarrow v + d' \models I(\bar{l}) \tag{3}$$

$$(\bar{l}, v) \rightarrow (\bar{l}[l_i \mapsto l_i'], v') \text{ if } \exists\, l_i \xrightarrow{\tau gr} l_i', \quad v \models g,\ v' = [r \mapsto 0]v, \text{ and } v' \models I(\bar{l}') \tag{4}$$

$$(\bar{l}, v) \rightarrow (\bar{l}[l_i \mapsto l_i', l_j \mapsto l_j'], v') \text{ if } \exists\, l_i \xrightarrow{\chi! g_i r_i} l_i' \text{ and } l_j \xrightarrow{\chi? g_j r_j} l_j', \quad v \models (g_i \wedge g_j),\ v' = [r_i \cup r_j \mapsto 0]v, \textit{ and } v' \models I(\bar{l}') \tag{5}$$

Then there are three possible evolutions for a network of automata: the active locations remain the same and the clocks values are increasing, *one* edge of one automaton is fired *or two* edges of two automata are firing simultaneously using a communication channel (also called synchronization channel).

Figure 2 is an example of a network of timed automata. It shows two automata linked by the communication channel COM, and sharing one clock t. The two initial locations are double circled locations Loc_A and Loc_1. At the initial state (initial locations active and t=0), this network can evolve according to any of the three semantics:

semantics (3) makes the clock increase within the limitation imposed by the invariant, such as t<=2, written in bold style near Loc_1.

semantics (4) makes the edge from Loc_A to Loc_B fire, because the guard of this edge is always true.

semantics (5) makes the edges from Loc_A to Loc_C *and* from Loc_1 to Loc_3 fire using communication channel COM, the emitting action uses symbol "!", symbol "?" standing for the receiving action.

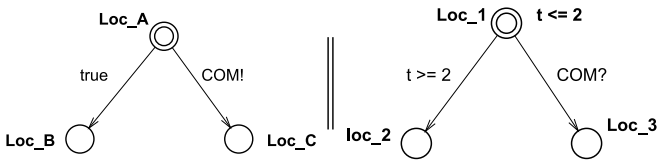

Fig. 2. Network of 2 timed automata.

2.3 Specific UPPAAL semantics

The main addition to the network of timed automata formalism made in UPPAAL is the notion of "urgency" which is used in two ways: a specification of locations as urgent or committed and a specification of urgent communication channels for synchronization.

- *Urgent locations* (U-decorated locations) have the following specific behavior: when at least one of such locations is active, all the clocks of the network are frozen. This can bring the network to a deadlock state. No invariant over clocks is allowed in an urgent location.
- *Committed locations* (C-decorated locations) bring a much stronger behavior than urgent locations. They have a similar effect on clocks by freezing them, and add another behavior: when at least one of the active locations is committed, the next edge fired must start from a committed location. This can also bring the network to a deadlock state, and no clock invariant is allowed in such locations.
- *Urgent communication channels* are synchronizations with time priority. If a transition involving urgent channel synchronization can be fired, all the clocks of the network are frozen. No clock guard is allowed with urgent synchronization.

Taking into account the following additional definitions, the semantics of these three specific behaviors can be defined by referring to the above limitations applied to semantics (3), (4) and (5):

- $L^u \subseteq L$ is the subset of urgent locations.
- $L^c \subseteq L$, $L^c \bigcap L^u = \emptyset$ is the subset of committed locations.
- $A^u \subseteq A$ is the subset of actions related to an urgent communication channel.

Urgent location behavior: if $\exists l_k \in \bar{l}, l_k \in L^u$, then semantics (3) is forbidden.

Committed location behavior: if $\exists l_k \in \bar{l}, l_k \in L^c$, then semantics (3) is forbidden *and* semantics (4) and (5) are limited to $l_i, l_j \in L^c$.

Urgent channel behavior: if, among the fireable edges, there exist two edges linked through a communication channel χ -as defined in semantic (5)- so that $\chi!, \chi? \in A^u$ then semantics (3) is forbidden and only semantics (4) and (5) are possible.

If the network of Figure 2 is an UPPAAL model and COM is declared as an *urgent* communication channel, then the only possible evolutions are the firing of the edge from Loc_A to Loc_B -because guard true does not need a clock increase to be validated- and the simultaneous firing of edges from Loc_A to Loc_C *and* from Loc_1 to Loc_3.

3. PLANT MODELING

To build the formal model of the plant, instantiation of formal generic models of plant components, such as actuators and sensors, was chosen. The benefits of this approach -modification ease and re-usability- are well-known.

Thus, the plant model will be composed of instances of generic component models, all these models being fault-free, i.e. they do not include faulty behaviors such as unexpected stops or untimely movements.

This section aims at presenting some generic models of typical plant components, as well as instances of these models for a small case study. The last sub-section addresses communications between the components models within the plant model and between these models and the controller model.

3.1 Generic components models description

Figure 3 shows the generic model of a pneumatic actuator plugged to a 5/3 normally closed preactuator where G_IN (respectively G_OUT) stands for Go In (resp. Go Out) and is the order of the ingoing (resp. outgoing) movement of the rod. Integer X represents the length of the rod which is out. It is modified by D_X or -D_X every time step during the movement. The limit values X_max and X_min are standing for the physical limits of the rod movement. The left (respectively right) location is modeling the ingoing (resp. outgoing) movement of the rod and the central location is modeling the rod motionlessness. As the initial state of the actuator is supposed motionless, the initial location of the model is the central one.

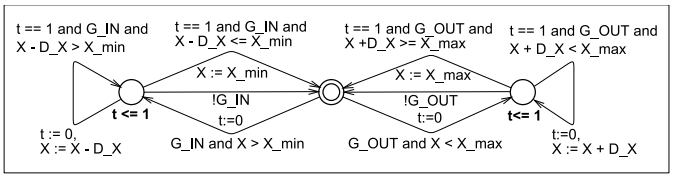

Fig. 3. Generic model of a pneumatic actuator plugged to a 5/3 normally closed preactuator where:
- G_IN and G_OUT are read (inputs of the model),
- X is written and read.

In order to reduce the number of models, all the sensors related to an actuator will be modeled by an unique model called a *sensor pack model*. In this model, each location represents a position which can be obtained from sensors values. Figure 4 represents a model of a pack of sensors composed of three logic sensors, so capable of detecting five positions of the rod of a pneumatic actuator.

The three signals issued from this sensor pack model are IN, OUT and MID for middle (outputs of the model), then the five positions are: IN, between IN and MID, MID, between MID and OUT and OUT. The model is sensitive to the value of integer X, which models a physical variable: the length of a rod which is out. X_IN_max is a configuration value standing for the maximum value of X needed to detect position IN. X_MID_min, X_MID_max and X_OUT_min are also configuration values used to settle the detection limits of middle and out positions. A particular location is added for initialization purpose: as no assumptions have been made over the value of X, the initialization of the model has to allow every possibilities. The committed status of this location ensures that the initialization of the model will be done *before* any standard evolution of the other models, exception made of the initialization of other models that will also use committed locations as initial ones.

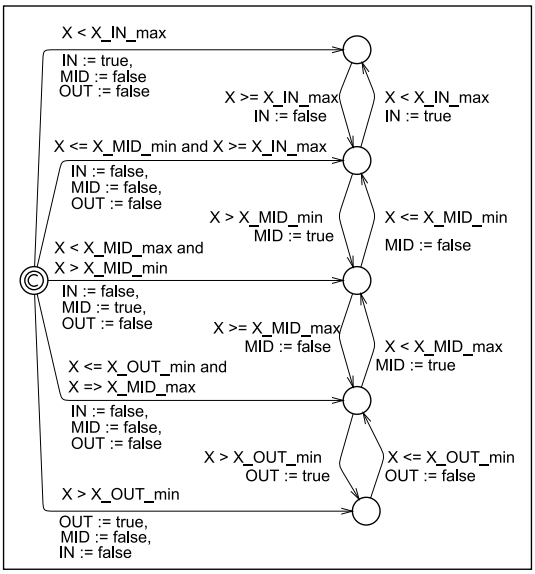

Fig. 4. Generic model of a 3-sensor pack where:
- X is read (input of the model),
- IN, MID and OUT are written.

3.2 Example

Construction of a plant model from generic components models will be exemplified on the case study of figure 5 that represents a synthetic view of the layout of a pneumatic manipulator composed by two pneumatic actuators coupled with their respective sensors and a sucker to grab pieces.

The formal model of this pneumatic manipulator includes:

- One instance of a generic 5/3 pneumatic actuator model (figure 3), named H_Act, for modeling the horizontal actuator, where X, G_OUT and G_IN are instantiated respectively by H, H_G_IN and H_G_OUT and where X_min, X_max and D_X are settled to 0, 20 and 2, respectively.
- One instance of a generic 5/3 pneumatic actuator model (figure 3), named V_Act, for modeling the vertical actuator, where X, G_OUT and G_IN are instantiated respectively by V, V_G_IN and V_G_OUT and where X_min, X_max and D_X are settled to 0, 10 and 1, respectively.
- One instance of a generic 3-sensor pack model (figure 4), named H_Sen, for modeling the set of sensors detecting the passage of the horizontal rod, where X, OUT, MID and IN are instantiated respectively by H, H_IN, H_MID and H_OUT and where X_IN_max, X_MID_min, X_MID_max and X_OUT_min are settled to 2, 9, 11 and 18, respectively.
- One instance of a generic 2-sensor pack model, 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, for modeling the set of sensors detecting the passage 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 settled to 1 and 9, respectively.

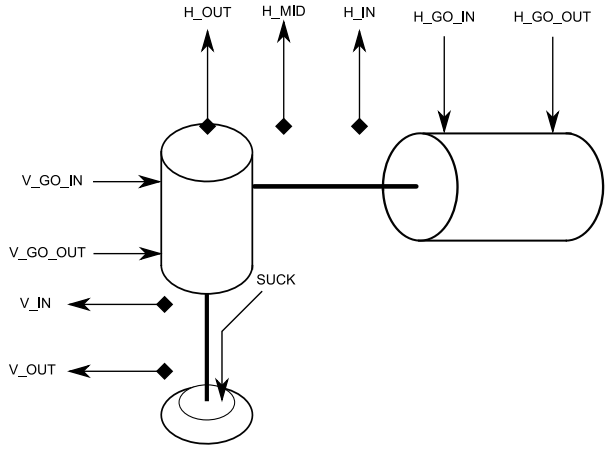

Fig. 5. Subpart of a plant: a pneumatic manipulator.

3.3 Communications between models

Therefore, the plant is modeled as a set of communicating timed automata which are obtained by instantiation of generic components models. The controller must also be modeled in this formalism, e.g. from a controller model in a standardized language, as presented in Mader and Wupper [1999], or from generic control blocks, as explained in Vyatkin and Hanisch [2000]. Whatever the solution selected to build the controller model, communication mode between timed automata is to be defined.

In this work, communications between two or several timed automata, which represent parts of the plant model or of the controller model, are implemented by using exclusively shared variables. To avoid multiple assignments, each shared variable must be assigned by one and only one model; multiple readings are possible, however. In the model of the above example, for instance, variable H is assigned only by the H_Act model but is read by both this model and H_Sen; in a similar way, variable H_GO_IN is assigned only by the controller model, is read by H_Act and may be read by other models of plant components which ensure operator's safety when the horizontal cylinder is moving.

4. PLANT MODEL EVOLUTIONS

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

4.1 Issue illustration on the example model

The following simple scenario 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 command to move to the middle position (variable H_G_OUT is set by the controller and will remain true until variable H_MID becomes true).

At the beginning of this scenario, the active location of H_Sen is its upper location (H_IN true, H_MID and H_OUT false) and that of H_Act its initial one; 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 to 1 and the loop edge of H_Act is fired, setting the value of H to 2 and resetting t. The state described on figure 6 (a) is then reached.

Here, two concurrent evolutions are possible according to the selected semantics:

- If semantics (3) is selected, clock t increases to 1 and then permits another evolution of H_Act (the invariant must be satisfied), using the 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 6 (b), where the dotted edge represents the fired edge.
- If semantics (4) is selected (figure 6 (c)), H_Sen evolves. The dotted edge is fired, which means that the sensor has detected that the rod is no more in its rightmost position (H_IN is reset). Clock t remains equal to 0 and H_Act does not evolve.

No evolution on the basis of semantics (5) is possible because communications between timed automata are based on shared variables and not on communication channels, as pointed out in the previous section.

This discussion shows clearly that the evolution with semantics (3) leads to a meaningless state that does not represent correctly the physical behavior of the plant composents: the model of the actuator assigns H value to 4 while the sensor model always yields the information H_IN true. This evolution is spurious. On the opposite, when semantics (4) is used, the sensor model evolves at the right time. This evolution corresponds to the real behavior and then is meaningful.

4.2 Generalization

The issue that was exemplified above is the general issue of concurrent evolutions in a network of timed automata. In such a network, none of the three semantics has priority over the other ones indeed. Then, edges can be concurrent with respect to only one semantics or to different ones. The first case is not really an issue, because it models often a real concurrency problem, like concurrent access to shared resources. On the other hand, the second case can lead to modeling flaws. Networks designers often think indeed that the time is spent -semantics (3)- after that all other possible evolutions have been made -semantics (4) & (5)-. This is not true 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.

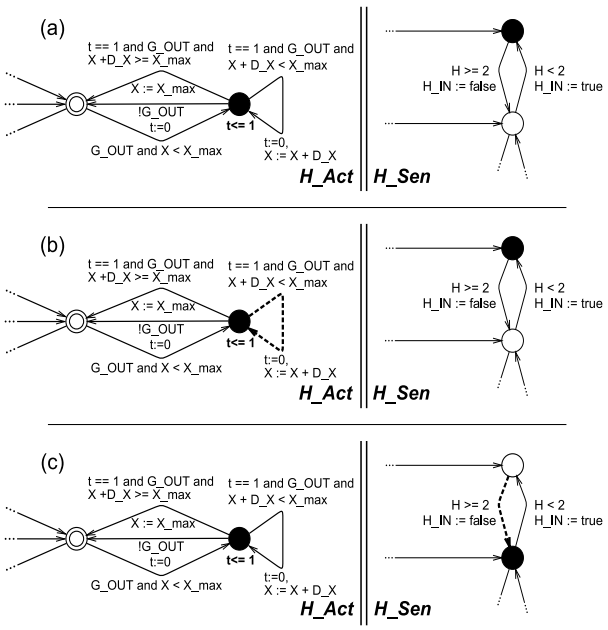

Fig. 6. Possible evolutions of H_Act and H_Sen, where the active locations are blacked.

5. MODIFICATION PROPOSED

5.1 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 evolutions of the plant.

These later 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 what follows, it will be assumed that any edge of the network of timed automata models either an instantaneous evolution or a timed evolution, but not both. If this is not the case, the edge must be split.

Extensions to standard formalism of timed automata are proposed in Barbuti and Tesei [2004] to model the first kind of evolution and to prioritize them over timed evo-

lutions. This valuable contribution was not retained for this work whose objective is not to extend an existing semantics, but to use all possibilities of the UPPAAL semantics to provide a solution avoiding concurrency between the two kinds of evolutions by always giving priority to instantaneous evolutions over timed evolutions. This solution is presented below.

5.2 Prioritizing instantaneous evolutions

UPPAAL proposes three ways to model urgency (subsection 2.3) but in this paper only urgent communication channels were kept to model instantaneous evolutions. Models of plant components contain indeed often edges that start from the same location but correspond to instantaneous and timed evolutions (see for instance the edges starting from the right or left locations in the actuator model); in that case, it is absolutely impossible to give priority to one evolution by defining urgent or committed locations.

An urgent communication channel describes synchronization between at least two automata. To modify as little as possible the plant model, the solution proposed is:

- to introduce an automaton, termed prioritizing automaton (figure 7), with only one location and one loop edge labeled by an urgent communication channel. The evolution of this automaton has priority over all other evolutions, provided that synchronization with another automaton with the same communication channel is possible.
- to add to each edge which models an instantaneous evolution an urgent synchronization with the prioritizing automaton.

With these modifications, clocks values cannot increase (semantics (3)) when instantaneous evolutions are possible;there is no more concurrency between instantaneous and timed evolutions.

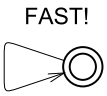

Fig. 7. Prioritizing automaton with urgent communication channel FAST.

5.3 Application to the example

Once the prioritizing automaton introduced and the two models H_Act and H_Sen modified (figure 8 (a)), only one evolution can occur when H reaches the value 2: instantaneous evolution of H_Sen (figure 8 (b)). The spurious evolution which was discussed in 4.1 is no more possible, because semantics (5) has priority over semantics (3) owing to the FAST urgent communication channel.

6. CONCLUSION

Instantiation of generic component models is an effective method to build plant models. This paper has shown that unexpected and erroneous behaviors -spurious evolutions and meaningless states- can appear in a so built plant model, when timed systems are dealt with, however.

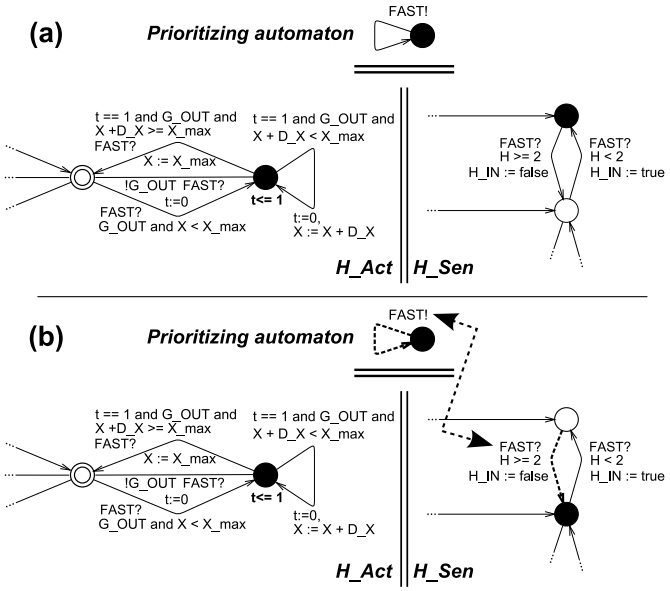

Fig. 8. Evolution of the modified example.

A simple solution based on urgent communication channel has been proposed to remove these erroneous behaviors. Further works are aiming at automating the plant model construction from a library of generic components models and at proposing sound abstractions to improve verification scalability.

REFERENCES

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

R. Barbuti and L. Tesei. Timed automata with urgent transitions. *Acta Inf.*, 40(5):317–347, 2004.

G. Behrmann, J. Bengtsson, A. David, K. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *Proc. of 7*th *International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, Oldenburg, Germany, September 2002.

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.

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

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

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

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