Verification of a timed multitask system with UPPAAL

Houda Bel Mokadem, Béatrice Berard, Vincent Gourcuff, Olivier de Smet, Jean-Marc Roussel

Verification of a timed multitask system with UPPAAL

Houda Bel mokadem, Béatrice Bérard, Vincent Gourcuff, Olivier De Smet and Jean-Marc Roussel

***Abstract*—System and program verification has been a large area of research since the introduction of computers in industrial systems. It is an especially important issue for critical systems, where errors can cause human and financial damages. Programmable Logic Controllers (PLCs) are now widely used in many industrial systems and verification of the corresponding programs has already been studied in various contexts for a few years, for the benefit of users and system designers. First restricted to an untimed setting, verification was recently extended to systems where quantitative constraints are needed, possibly related to time elapsing. For instance, timed features like TON (Timers ON delay), used in PLC programs, were modeled with timed automata, thus increasing the size of the verification problems addressed.**

In this framework, we propose the modeling and verification of a particular timed multitask PLC program, which is part of the so-called MSS (Mecatronic Standard System) platform from Bosch Group. In this case study, time aspects are combined with multitask programming, which raises questions related to the reaction time between the detection of a signal and the resulting event. Our model for station 2 of the MSS platform is a network of timed automata, including automata for the operative part and for the control program, which is first described in SFC then translated in *Ladder Diagram*.

This model is constrained with atomicity hypotheses concerning program execution, and model checking of a reaction time property is performed with the tool UPPAAL.

***Index Terms*—Programmable Logic Controllers, Timed Automata, Model Checking.**

I. INTRODUCTION

General context. Verification of safety properties for programs is a very important issue for users and system designers, particularly when those programs are to control critical applications for reactive systems. Indeed, many accidents have been discovered to be the result of programming errors, which lead to the need of formal methods for system verification. Among these methods, model-checking is a rather successful one. Basically, the model-checking technique consists in (i) modeling the program and its environment as a transition system, (ii) modeling some property as a logical formula, and (iii) using an algorithm to test if the model satisfies the formula. This algorithm is based on a symbolic exploration of the system state space, thus providing an exhaustive search for counter-examples for the property to be tested. It is implemented in a tool called *model-checker*, many of them having been developed for untimed systems.

Since the nineties, the attention focused on systems where quantitative properties related to time are involved. For such systems, additional components must be introduced, for instance clocks, thus increasing the size of the model and making the verification step harder. Nevertheless, the model of timed automata, introduced in 1990 by Alur and Dill [2], [3], has proved very fruitful: some positive decidability results were obtained for this model, as well as for some extensions, and analysis algorithms were implemented in efficient tools called *timed model-checkers*, like HYTECH [10], KRONOS [6] or UPPAAL [12], which were then applied to industrial case studies.

Comparison with other techniques [9] [22] such as discret event simultation (POOSL, *http://www.es.ele.tue.nl/poosl/*, SHESIM), symbolic timing analysis (SymTA/S, *http://www.symtavision.com*), modular performance analysis (MPA, *http://www.mpa.ethz.ch*) showed similar cost in time and resources for less accurate results. This comforts us in the choice of the timed verification approach.

The particular framework of PLC programs also attracted increasing interest in the past few years. In this area, work was mostly devoted to the untimed setting [18], [4], [8], even when function blocks for timers were included [19], although the model of timed automata has already been used for the modeling of timed features in PLC programming [7], [15], [16].

Contribution. In this work, we are interested in the combination of these time aspects with multitask PLC programming. Our case study concerns a part (called station 2) of the MSS (Mecatronic Standard System) platform from Bosch Group, in which multitask programming can be used to reduce the reaction time of the control program to an external signal. The program is written in *Ladder Diagram*, one of the languages most commonly used in this area, which is part of the IEC-61131-3 standard [11]. We give semantics for a subclass of Ladder Diagram programs including timer function blocks, in terms of timed automata, and we also provide a timed automata based model for the operative part of the system. These timed automata are described in UPPAAL syntax. While a similar approach was introduced in [15], we propose here additional restrictions which significantly reduce the size of the complete model, obtained from its components by a synchronized product. These restrictions consist of atomicity hypotheses, compacting sequences of actions from the control program into a single one, and lead to reasonable verification

Manuscript received March 23, 2009

This work was supported by the Pluri Formation Project VSMT of ENS Cachan.

H. Bel mokadem is with LSV, Ens de Cachan, mokadem@lsv.ens-cachan.fr

B. Bérard is with LIP6, Université P. et M. Curie, beatrice.berard@lip6.fr

V. Gourcuff is with LURPA, Ens de Cachan, vincent.gourcuff@lurpa.ens-cachan.fr

O. De Smet is with LURPA, Ens de Cachan, olivier.de_smet@lurpa.ens-cachan.fr

J.-M.. Roussel is with LURPA, Ens de Cachan, jean-marc.roussel@lurpa.ens-cachan.fr

times for the response property to be checked. We also give a simpler model for timers, using particular features of UPPAAL.

Outline. Section 2 of the paper explains the context of this study: the problem of reaction times in PLC programs, and includes a description of timed automata and a short presentation of UPPAAL. In Section 3, we give more details on Bosch MSS platform and in Section 4, we give the semantics of the control program. Section 5 presents the UPPAAL timed automata which form the components of the network, while Section 6 gives the results of the verification step.

II. PROGRAMMABLE LOGIC CONTROLLERS AND TIMED AUTOMATA

In this section, we describe the general context of our study and recall the main features of timed automata.

A. Programmable Logic Controllers with multi-task programming

Programmable Logic Controllers (PLCs) execute programs for the control of an operative part, to which they are connected via an input/output system. The control programs can be written in several languages described in the IEC-61131-3 standard. The execution of such a program consists in iterating a cycle with three main steps (Fig. 1): first, input variables are read and their values are stored in memory. Then a computation step is performed using these values, producing output values which are also stored. The last step is an activation using the output values. The cycle duration P is called the PLC scan.

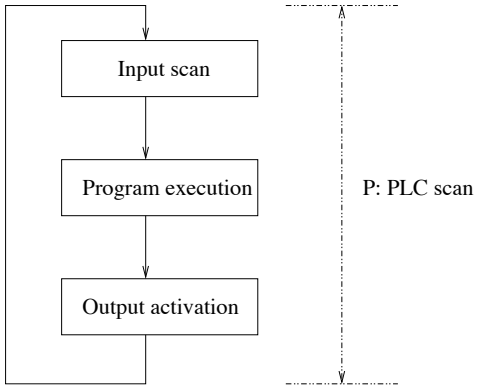

Figure 1. The cyclic execution of a PLC program

The programming design may be either mono-task or multitask. In the first case, a single program executes sequentially, while in the second case, the main task can be interrupted by additional parts of code, either with a fixed period or triggered by some events. These two execution models result in different reaction times to changes of values. In the mono-task case, if the change of value occurs at the input scan, the corresponding output is emitted at the end of the PLC cycle. If the change occurs later, this output may be emitted at as far as the end of the next cycle. This results in a reaction time in the interval $[P, 2P]$ (Fig. 2). This reaction time can be reduced with multi-task programming, which is available in most PLCs (with the use of so-called Organization Blocks [20]): consider an event-driven task interrupting the main task when some event occurs. In turn, the interrupting task reads its input and computes its new output values. Depending on the configuration and type of the PLC, these values can be emitted either at the end of the event-driven task or at the end of the current main task. In this work, we investigate the second case where output values of the event-driven task are emitted by the main program, which yields a reaction time of at most P.

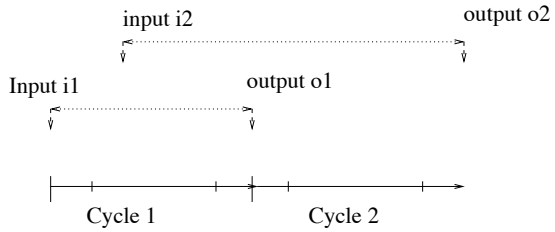

Figure 2. Reaction time with mono-task programming

B. Timed automata

The model of timed automata was introduced by Alur and Dill [2], [3]. A timed automaton is built from two main parts:

- a finite automaton in the usual sense, which describes the states also called locations and the discrete transitions of the system,
- a finite set of real variables called clocks, used for the specification of quantitative time constraints which may be associated with transitions. These variables evolve synchronously with time.

Example. The timed automaton on Fig. 3 describes a system which observes a sequence of events a. In location **ok**, the delay between two consecutive occurrences of a must belong to $[5, 6[$. If it is less than 5, an alarm is triggered (the automaton enters location **alarm**), and if the time reaches 6 before the next a occurs, the system enters the **error** location. To measure these delays, a clock x is reset at each occurrence of a inside the interval $[5, 6[$. It then increases as time elapses and its value can be compared to the constants 5 and 6, to see if a change of mode is required.

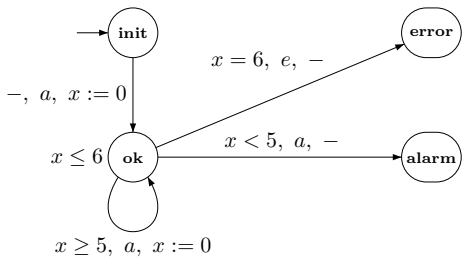

Figure 3. A timed automaton

Formally, a timed automaton is a tuple $A = (\Sigma, X, Q, q_0, I, E)$, where

- Σ is a finite set of actions,
- X is the finite set of clocks,

- Q is a finite set of locations, with $q_0 \in Q$ the initial location,
- I is a mapping associating with each location q an *invariant* $I(q)$,
- and E is the set of transitions.

In order to describe more precisely the components I and E, we denote by $P(X)$ the powerset of the set X (of clocks) and by $C(X)$ the set of boolean conjunctions of atomic formulas of the form $x \bowtie c$ for a clock x, a constant c in $\mathbb{N}$ (the set of natural numbers) and $\bowtie$ in $\{<, \leq, =, \geq, >\}$. Elements of $C(X)$ are called *clock constraints*.

- An invariant $I(q)$ for location q is a clock constraint, which contains only atomic formulas of the form $x \leq c$ or $x < c$. This constraint must hold as long as time elapses in this location. In the example of Fig. 3, the condition $x \leq 6$ is an invariant for location **ok**.
- The set E of transitions is a subset of $Q \times C(X) \times \Sigma \times P(X) \times Q$.
  A *transition* (q, g, a, r, q') of the automaton is written $q \xrightarrow{g,a,r} q' \in E$. Its label contains three parts (each one is optional):
  1) The first one g is a constraint in $C(X)$, called the *guard*, which must be satisfied for the transition to be fired. For instance, $x < 5$ is the guard for the transition from location **ok** to location **alarm**.
  2) The second one, called *action name*, is an element of Σ, like a in the example.
  3) The third one, $r \in P(X)$, called *clock reset*, contains the subset of X of clocks which must be reset to zero when the transition is fired. For $r = \{x\}$, a reset of x when reaching location **ok** is written $x := 0$ in Fig. 3.

In order to define the semantics of a timed automaton, we use the notion of *clock valuation*. Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. A valuation is a mapping from X to $\mathbb{R}_{\geq 0}$, and the set of valuations is written $\mathbb{R}_{\geq 0}^X$. For each $v \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$, we use $v + d$ to denote the valuation which maps each clock $x \in X$ to the value $v(x) + d$.
For a subset r of X, we write $v[r \leftarrow 0]$ for the valuation which maps each clock in r to the value 0 and agrees with v over $X \setminus r$.
Constraints of $C(X)$ are interpreted over clock valuations.

The semantics of a timed automaton is given in terms of transition systems. A *configuration* of the system is a pair (q, v), where q is a location of the automaton and v is a valuation of the variables, *i.e.* a mapping associating a real value with each clock. The initial configuration is (q_0, v_0) where all clock values are equal to 0 in v_0.
The system may change its configuration in two ways.

- Either by letting time elapse: as long as no invariant is violated in the current location, time may progress and the clock values increase by the amount of time elapsed. Such a move is written $(q, v) \xrightarrow{d} (q, v + d)$ for a delay of d time units, and it is possible only if $v + d$ satisfies the invariant $I(q)$ of location q.
- Or by carrying out a discrete transition of the automaton: the clocks to be reset take the value zero, while the values of the other clocks remain unchanged. This is written $(q, v) \xrightarrow{a} (q', v')$, when there is a transition $q \xrightarrow{g,a,r} q'$ in E such that v satisfies the constraint g, $v' = v[r \leftarrow 0]$ and v' satisfies the invariant of q'.

In the example above, the initial configuration of the system is $(\mathbf{init}, 0)$. A particular execution of the system can be described by the following sequence of configurations:
$(\mathbf{init}, 0) \xrightarrow{7.2} (\mathbf{init}, 7.2) \xrightarrow{a} (\mathbf{ok}, 0) \xrightarrow{5.3} (\mathbf{ok}, 5.3) \xrightarrow{a} (\mathbf{ok}, 0) \xrightarrow{6} (\mathbf{ok}, 6) \xrightarrow{e} (\mathbf{error}, 6)$

C. The tool Uppaal

The tool UPPAAL (see [5] for the most recent developments) offers a compact description language, a simulation module and a model-checker. In this paragraph we present a subset of UPPAAL functionalities which are sufficient for our purpose. A system is represented in UPPAAL by a collection of timed automata, which communicate through binary synchronization: a channel c can be defined for two automata. Sending a message is denoted by the discrete action $c!$ while receiving the message is denoted by $c?$. These automata also handle a set of (discrete) integer variables, a feature which makes easier the modeling of complex systems. Then, a guard is a conjunction of atomic clock conditions and similar conditions on the integer variables. Moreover, a clock reset may be augmented by an update of the integer variables, of the form $z := c$ for some constant c.

A global *configuration* is a triple (ℓ, v, w) where ℓ is a location vector (indicating the current state in each component of the timed automata network), v is a valuation of the clocks and w is a valuation for discrete variables: as before, they assign to each clock a real value, but they also assign to each discrete variable an integer value. An execution in the network starts in initial locations of the different components with all the clocks and variables set to zero. The semantics of this model is expressed by moves between configurations. Three types of moves can occur in the system: delay moves, internal moves and synchronized moves. Delay moves and internal moves have already been described above for a single automaton, so we simply describe now the global evolution.

1) Delaying.: Given a current location vector, time elapses for all automata synchronously, as long as no invariant is violated. All clock values increase by the amount of time elapsed. No changes occur for the locations or the integer variables. The move for a set of n automata can be described as above by $(\ell, v, w) \xrightarrow{d} (\ell, v + d, w)$, where $\ell = (q_1, \ldots, q_n)$ is the tuple of locations, v is the clock valuation and w is the valuation of discrete variables. It is possible only if $v + d$ satisfies the conjunction of invariants $I(q_i)$, for all $1 \leq i \leq n$.

2) Performing an internal action.: An internal action is an action which corresponds to neither $c!$ (sending a message), nor $c?$ (receiving a message). If such an action is enabled (the variable values satisfy the guard condition), the component can perform this action alone, while the others do nothing. Only the location of this component is changed, as well as its variables, according to the transition. If $q_i \xrightarrow{g_i,a_i,r_i} q_i'$ is the transition possible from component i, the global move can be written $((q_1, \ldots, q_i, \ldots, q_n), v, w) \xrightarrow{a_i} ((q_1, \ldots, q_i', \ldots, q_n), v', w')$

with $v' = v[r_i \leftarrow 0]$ and w' is obtained by the update of discrete variables from component i.

3) Synchronizing.: If, in the network, some complementary actions $c!$ and $c?$ are enabled in two components (in particular, guards must be satisfied by the current valuation), then these components can synchronize. Note that if a component can execute only an action $c!$ while no other component can perform a complementary action $c?$, then this component is blocked. A carefree programing can lead to a deadlock, for instance if the system consists of two components, the first one executing $c!$ followed by $d?$ when the second executes $d!$ followed by $c?$. In the synchronizing transition, the location vector is changed for both components and the clock and variable values are changed according to the clock reset and updates of variables for the two transitions. If components i and j synchronize on some channel c, with transitions $q_i \xrightarrow{g_i, c!, r_i} q_i'$ and $q_j \xrightarrow{g_j, c?, r_j} q_j'$, the move is described by $((q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n), v, w) \xrightarrow{c} ((q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n), v', w')$, with $v' = v[(r_i \cup r_j) \leftarrow 0]$ and w' is obtained by the update of discrete variables from components i and j. Multiple assignment of the same variable can occur, but it generally indicates bad programming in the control program.

To illustrate this composition operation, consider the example from [21], where three components $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{M}$ communicate with two channels in and out (Fig. 4). The first one produces an $in!$ message every 7 time units, the second one, $\mathcal{B}$, emits $out?$ messages, with at least 4 time units between them, and the third one, $\mathcal{M}$, performs the communication by transmitting to $\mathcal{B}$ ($out!$) the messages received from $\mathcal{A}$ ($in?$), with a delay between 3 and 8 time units.

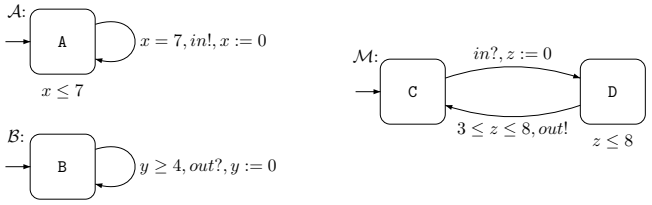

Figure 4. Three components with channels in and out

Fig. 5 shows how to obtain the composition $\mathcal{A} \mid \mathcal{B} \mid \mathcal{M}$, by synchronizing on the channels in and out.

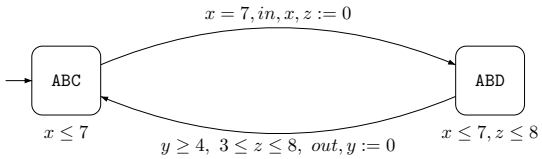

Figure 5. Composition $\mathcal{A} \mid \mathcal{B} \mid \mathcal{M}$

Finally, we introduce two additional features of UPPAAL which will be very useful in our modeling.

- A *committed* location (decorated by the special label C) in an automaton, corresponds to a location in which no progress of time is possible and no transition from a non-committed location is allowed.
- A *broadcast* channel is a channel where more than two automata may communicate: emission of a message $c!$ can be synchronized with several (possibly zero) receptions $c?$ in other components. Note that this is a non-blocking synchronization, since the sender is never blocked, although the receiver must synchronize if it can. Guards on clocks are not allowed on the receiving edge.

III. DESCRIPTION OF THE MSS (MECATRONIC STANDARD SYSTEM) PLATFORM

A. Presentation.

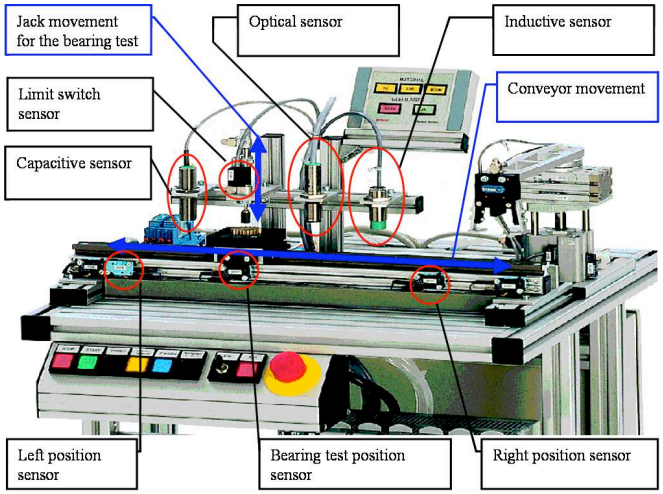

Figure 6. Presentation of station 2 of the MSS platform

Platform MSS (from Bosch Group) provides a function for sorting a stock of pinions of different materials and for adding or withdrawing a press-fit bushing to a given pinion [17]. The platform contains four stations.

Our study is centered on station 2. Fig. 6 shows the various components of this station, which are schematized in Fig. 7. The work-pieces are transported by a linear conveyor put into motion by a belt, and driven by an engine with two direction moves. They first reach a scanning position, where the presence or absence of a press-fit bushing is detected. They are then tested by three sensors (respectively inductive, capacitive and optical sensor) to determine their material (steel, copper or black PVC). The detected information is forwarded to the next stations. A rotary/lift gripper performs the transfer to a follow-on station if applicable.

The function performed by the controller for the linear conveyor and the detection of the press-fit and material is presented in Fig. 8 by the SFC [11] specification, where no interruptive task is used. In step 1 the motor starts to move the conveyor to the right. Steps 2 and 3 are used to detect the presence of a pinion and arrival of the conveyor at the test position. Step 4 stops the motor to halt the conveyor, steps 5 and 6 correspond to the test by the jack. At the end of the test, step 7 restarts the motor to move the conveyor further to the right. Steps 8 to 11 test the material of the pinion as it passes under the various sensors. Step 12 is activated when the conveyor is at the rightmost position (motor stops), the

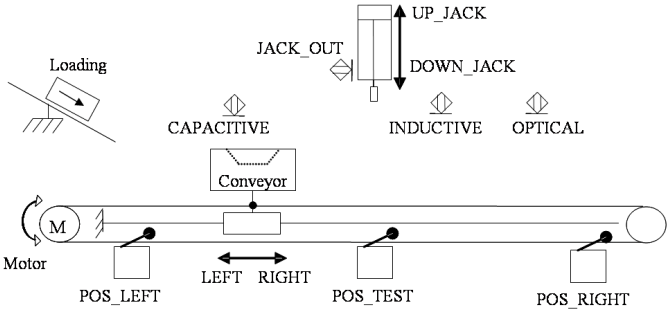

Figure 7. Scheme of station 2

end of the transfer by the rotary/lift gripper is indicated by EVACUATED_PINION. Step 13 returns the conveyor to its initial position. It can be noted that the controller specification uses 6 timed functions as TON blocks (for instance `t1/X1`), in order to measure delays. The semantics of timers is explained in IV-C.

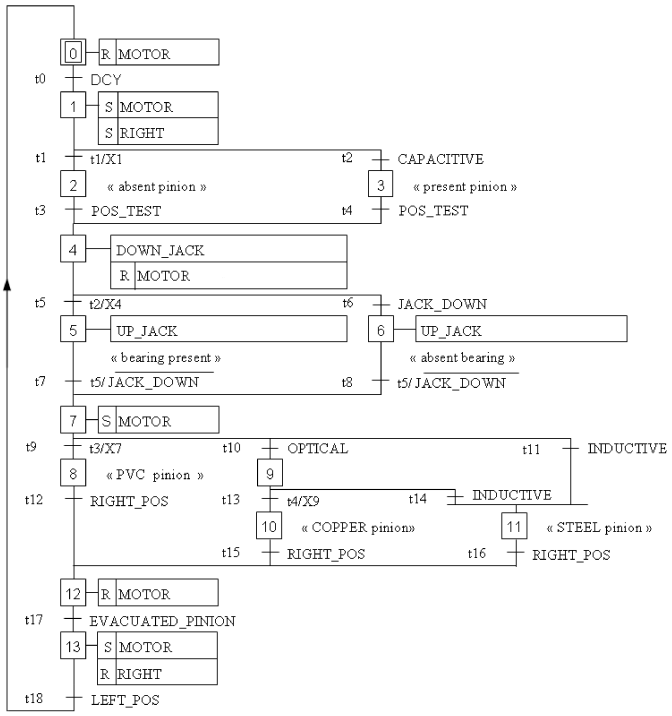

Figure 8. SFC specification of station 2

With this specification, a problem arises when the conveyor arrives at the bearing test position (POS_TEST sensor). At this time the conveyor moves at high speed (200 mm/s) and the variation of the reaction time of the control system, above 10 ms, is not negligible. Indeed the conveyor position should have a precision of 1 mm for the tester (or jack) to be able to penetrate inside the pinion, in case the bearing is absent. So, we can deduce that the the variation of the reaction time of the control system must be less than 5 ms. In the rest of the paper, we study the case of a multitask controller, with an event-driven task, launched on the rising edge of the test position (POS_TEST) sensor, which stops the conveyor if it comes from the loading station.

B. Properties to check.

The multitask control program of this station must satisfy the following properties:

P1: to ensure safety, the conveyor must stop on its way out but not when it comes back from unloading.

P2: the time performance is accurate: the conveyor stops in less than 5 ms at the press-fit bushing test point.

In this work, we focus on the timed property **P2**, and show that the multitask solution reduces the reaction time (Fig. 2). For the first property, the model of the system will be built so that an erroneous control program can invalidate it. In our case, the proposed control program ensures **P1**.

IV. Modeling station 2

In this section, we briefly recall the timed automata based semantics proposed by [15] for a control program. Then we explain the structure of our model for (station 2 of) the MSS platform, with a particular attention to the question of timers. Finally we give the complete description of Uppaal timed automata for the system.

A. Modeling principles for the control program

Various models have already been proposed for the analysis of PLC programs. Our approach is based on the model introduced by [15], which disregards the exact execution times of elementary instructions.

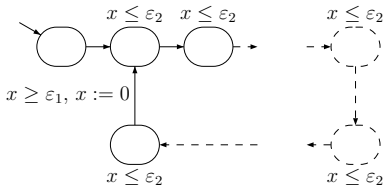

Figure 9. Mader-Wupper model

As depicted in Fig. 9, the model has a clock x to measure the cycle scan, which is thus reset after each cycle of the program. The duration of the two steps `input scan` and `output activation` (see Fig. 1) is at least ε_1 while the whole PLC cycle is at most ε_2. Therefore, the invariant $x \leq \varepsilon_2$ is associated with each location. The guard $x \geq \varepsilon_1$ appears on the last edge of the cycle which models the `output activation` and `input scan` steps. A dotted edge in the model describes a step of the control program.

Mader-Wupper also models each timer block as a timed automaton that runs in parallel with the control program. Synchronization is performed through operations on the timer variables and on the timer calls, which requires one extra clock and three synchronization channels for each timer.

B. An overview of the model

Our model is built in a compositional way from a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables (see Fig. 10). The two main parts are the environment

and the control program, which communicate through shared variables and synchronization messages. The modeling of the operative part (environment) is necessary for the verification of the safety and performance properties stated previously.

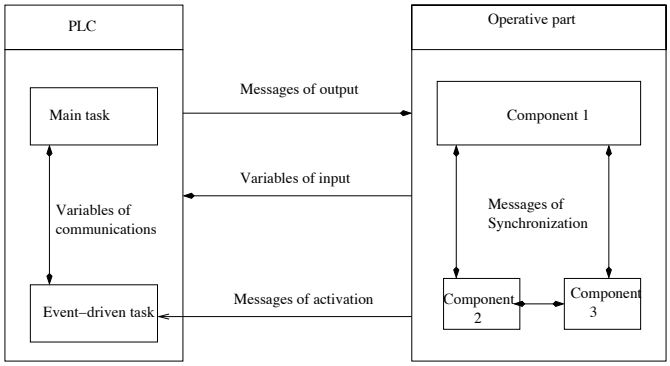

Figure 10. An overview of the model

The operative part initializes the input variables of the PLC, used to compute the output variables. According to these new values, messages are sent to the operative part, acting as orders given to start or stop the conveyor, or to get the jack down. When the conveyor is at the testing position, the operative part sends a message to activate the event-driven task. Details are explained in Section 5.

C. Modeling timers

The control program contains timing operations, described by functional blocks called TON (Timer On-delay). This mechanism has:

- two input variables: a boolean variable IN, to start or stop counting the time and a time parameter PT (Preset Time) which indicates the timing delay,
- two output variables: a boolean variable Q, with value 1 if the delay has expired and a time variable ET (Elapsed Time) which gives the time elapsed from the last rising edge of IN.

The IEC 61131-3 standard explains the behavior of a TON block by the diagram in Fig. 11. The time count starts when the variable IN changes from 0 to 1. Then variable ET increases until reaching the value PT. At that point, variable Q is set to 1. When IN changes to 0, then ET and Q are immediately set to 0. If IN changes from 1 to 0 before ET reaches the delay PT, then ET is immediately set to 0.

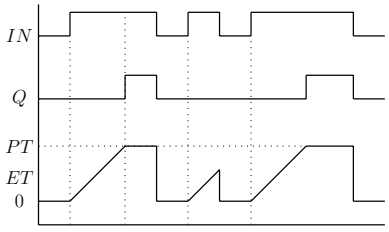

Figure 11. Time diagram of a TON block

Note that only the boolean variables of the timers IN and Q appear in the control program (Fig. 8). For the IEC 61131-3, it is not mandatory to connect all input and output variables of a functional block. Since the variable ET is not used in the program it will not appear explicitly in the model of the timer (see Fig. 12), but is instead modeled by a clock. The parameters PT associated with various instances of TON blocks are defined in a declaration part which is not represented here.

Six independent timers are used in station 2 control program. We now explain how our model of a TON function block differs from that of Mader-Wupper [15] and how we use broadcast channels in UPPAAL to avoid deadlocks. Each TON block is modeled by an automaton, described in Fig. 12, with three locations, one clock *x-Ton* and two discrete variables *Ton-Ine* (input) and *Ton-Qe* (output).

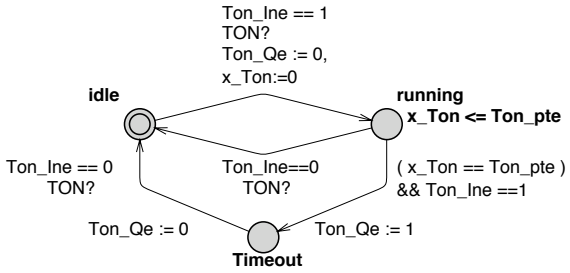

Figure 12. UPPAAL model of a TON block

Initially **idle**, the location becomes **running** when the timer has been switched on and **Timeout**, when some fixed preset delay (constant *Ton_pte*) has been reached. At each cycle of the main task, a synchronization message `TON!` is sent with a broadcast channel (as seen in Fig. 18). This message is synchronized with `TON?` for all timers. We choose this modeling technique because it allows us to use a single broadcast channel for all TON blocks instead of three ordinary channels per TON in Mader-Wupper's model. Note that in our case, no deadlock can occur. This choice is validated in practice since all TONs are used to measures delay of larger scale than the PLC cycle time. Another assumption in PLC languages is that the state of a TON block cannot change as the control program is processed.

D. Modeling the environment

In order to validate not only the PLC program but also its integration in the system it has to control, we also need to model the operative part. This implies a thorough knowledge of the system to control, particularly the behavior of each element and its reaction time. Modeling the environment makes it possible to speed-up the verification time, in particular by reducing the combinatorial aspects related to non-deterministic definition of all possible input values, including sometimes non relevant ones. Indeed, when the input values of the PLC program are emitted by a model instead of a non-deterministic process, the space of reachable states is reduced. Since our timed properties are significant only in nominal mode, we choose to restrict the model of the plant to the nominal mode without failure.

Each physical device is represented by a timed automaton. In such an automaton, a given location represents a particular configuration of the device. However, while some components of the operative part, like a sensor for example, behave as discrete event systems, this is not the case for all of them. Some devices like closed-loop control have pure continuous behaviors, and cylinders have hybrid behaviors, discrete in the end of their course but continuous during the move. In the models proposed here, clocks are the only continuous components, while physical continuous moves are discretised (for instance for the conveyor). Modeling with hybrid tools like HYTECH (instead of UPPAAL) would be a solution to this problem, but at the price of much larger verification times, and sometimes even no guarantee of termination.

1) The linear conveyor: The conveyor is the main element of the operative part, because several triggerings of sensors depend on its position. It is also the most delicate to model because of its continuous behavior along the belt, while our model can only provide a discrete abstraction of this behavior, leaving out the details which do not influence the properties to be checked. In order to obtain reasonable performances in terms of memory and automatic verification time, we model only the almost stable positions, *i.e.* the positions where the conveyor can stop, or trigger a sensor. These positions correspond to the six states: **inductive-sensor**, **capacitive-sensor**, **optical-sensor**, **test**, **left**, **right**. They are simply associated with a particular point on the conveyor trajectory, useful for detection, but the conveyor does not stop in these positions, except if it receives a *stop!* message. Between two given positions, the behavior of the conveyor is assumed to be a movement with constant speed, and is thus modeled by a clock. This clock evolves in only one state and is controlled by an invariant, the corresponding constant representing the time needed by the conveyor to cross the distance between these two positions. For example, the conveyor goes from the left position to the capacitive-sensor position in 490 to 500 ms. There is another abstraction imposed by the fact that no stopwatch exists in UPPAAL: between two almost stable positions, the conveyor cannot change direction. However, on a stable position, the direction is permitted to change, which allows property **P1** to be violated. In fact this property is satisfied with our control program as in [18]. The conveyor sends synchronization messages to the various sensors (like $optics$!) and the event-driven task ($postest$!) at the time of its arrival to the test position. It also modifies the input variables of the control program. The corresponding automaton is represented in Fig. 13.

Note that we did not model time non-determinism for the movement of the conveyor from right to left (lower part of the automaton). This is not relevant for the properties to check as this corresponds to step 13 of the SFC specification Fig. 8.

2) The external environment: In station 2, the leftmost position corresponds to the loading of pinions, while the rightmost position is used for unloading. However, the control of loading and unloading operations is not part of this station, which just waits for them to be done. Information about termination of one of these operations is obtained through changes of input values. Upon loading, the conveyor is provided an unspecified pinion. This is modeled by an automaton, presented in Fig. 14, which selects in a non-deterministic way the nature of the pinion (variable *ob*) when the conveyor is at the leftmost position.

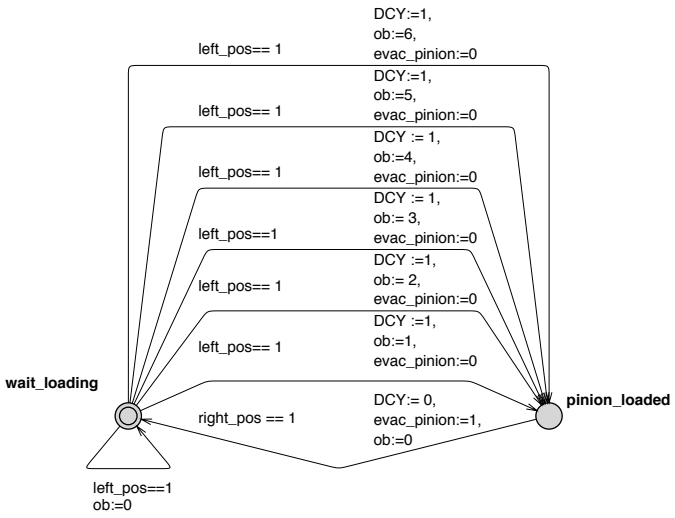

Figure 14. Model of the environment external to station 2

3) The jack: The jack detects the presence or absence of a press-fit bushing in a work-piece. This test is made by a vertical movement of the jack until a limiting position. The jack must go down until the limiting position is reached, in a given time, to conclude to the absence of the press-fit bushing. The time alloted for this movement is given in the specification (see Fig. 8) by the TON `t2/X4`. As it is larger than the PLC cycle duration, the *down_jack* order will be held during several PLC cycles. The model of this sensor (Fig. 15) depends on the characteristics of the work-pieces which are represented by the values of the variable *ob*. The automaton starts from location **top**. It moves to state **go-down** when it receives a message *down-jack?* from the PLC program. From this point on, there are two cases: if there is a press-fit bushing in the work-piece (represented in the model by the guard $ob == 1||ob == 3||ob == 5$) then the automaton waits in the state **go-down**, else it moves to state **limiting position**. This model of the jack needs at least two PLC cycles to reach a stable position, which is ensured by the `t2` duration.

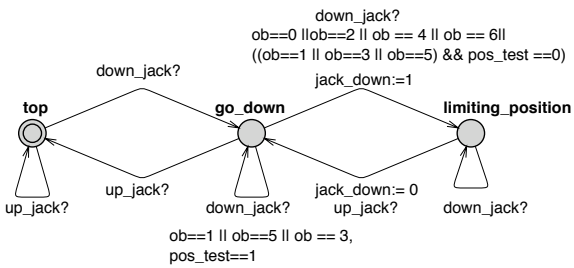

Figure 15. Timed automaton for the jack

4) The sensors: The optical, capacitive and inductive sensors are modeled by timed automata synchronized with the automaton of the conveyor. We only show here the model for

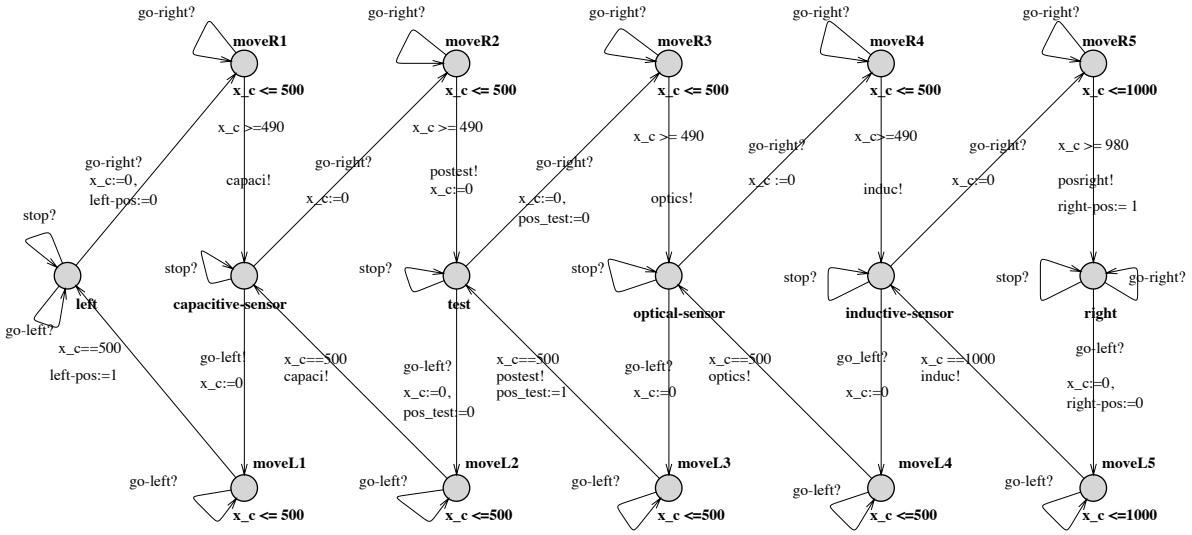

Figure 13. Timed automaton for the conveyor

the optical sensor, the others are very similar, only location names and signal names should be adapted. The conveyor sends the activation messages (for example *optics?* in Fig. 16) when it is under the corresponding sensor. According to the nature of the material, the sensor modifies the value of the corresponding variable (here *optical*) which is then used by the PLC program. No clock is used in this model as its value is used only during the input scan of the PLC cycle. The state of the sensor is used as a constant within a given PLC scan. The filtering delay for the sensors is assumed to be smaller than the input scan duration (Fig. 1).

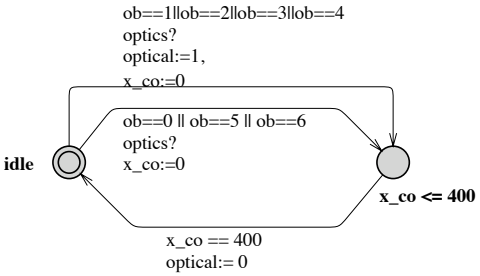

Figure 16. Timed automaton for the optical sensor

E. Modeling the control program

1) The main program: The functional specification of the global system is modified from Fig. 8 to use an interruptive task in Ladder language (see Fig. 17).

In this specification, the interruptive task is triggered by the `POS_TEST` sensor. To stop only from left to right, the action is conditioned by being in SFC step 2 or 3 (denoted by `x2` and `x3` in the interruptive task in Fig. 17.

This SFC specification needs to be translated into a PLC programming language. Recall that the execution of a PLC program is a cycle with three phases: input reading, program execution and output writing as in Fig. 1. This translation is done using the (classical) algebraic representation of SFC [13]

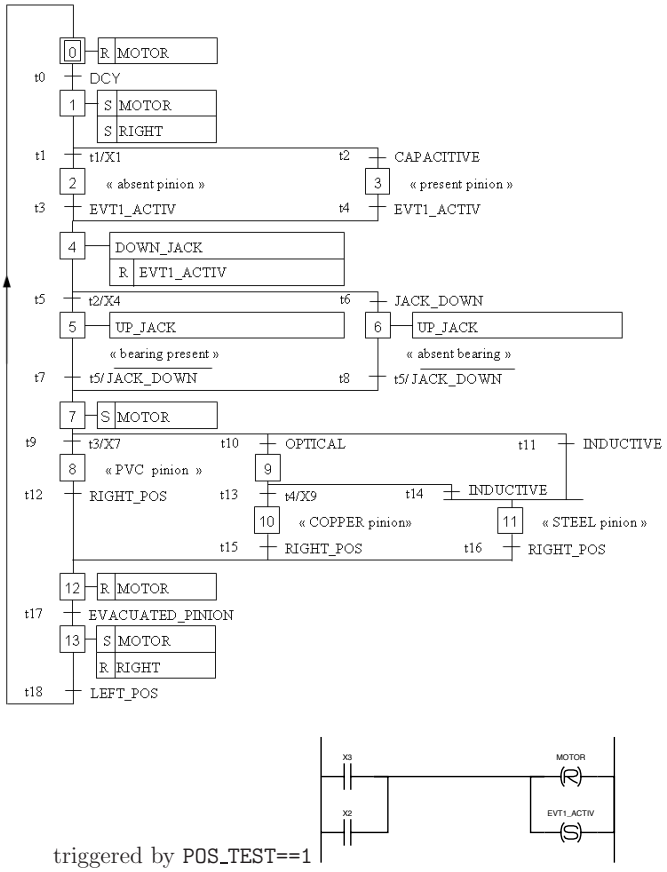

Figure 17. SFC specification of station 2 with LD interruptive task

with three sequential modules inside the program execution part.

- The first module is the computation of the clearing conditions of the transitions as boolean variables called `CFT0 ... CFT18` in Fig. 18. For instance, the `t0` transition in Fig. 17 is associated with `CFT0` variable

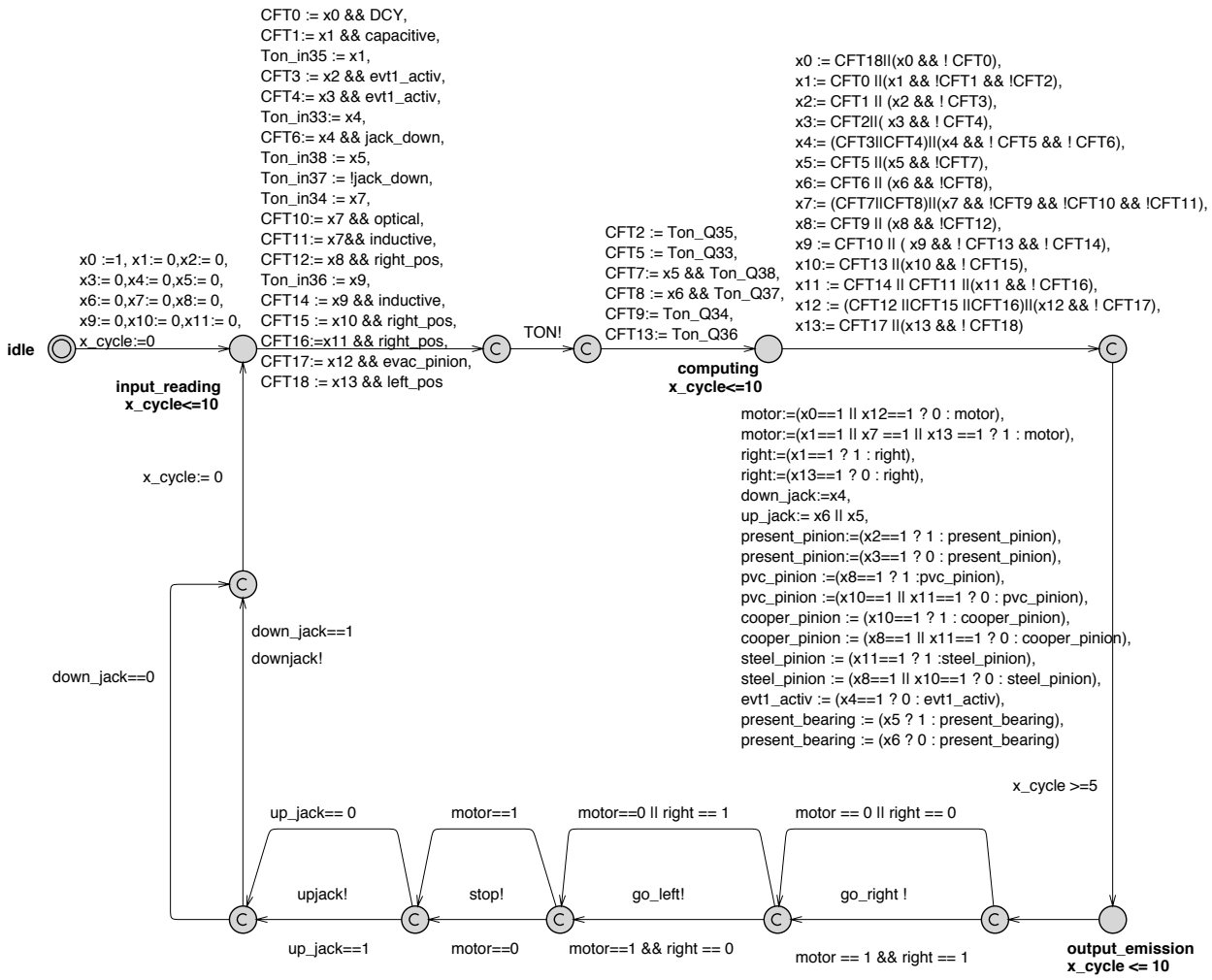

Figure 18. UPPAAL model of main program

defined by `CFT0=x0&&DCY` which means that state 0 must be active and the transition condition `DCY` must be true to allow activation of step 1 and deactivation of step 0 in the next module.

- The second module is the computation of the step variables called `x0 ... x13` in the same figure. For step 1, activation is computed by `x1=CFT0||(x1&&!CFT1&&!CFT2)`.
- The last module is the computation of actions using the step variables. As seen in Fig. 17, action `motor` is reset when step 0 or step 12 is activated and set when step 1 or step 7 or step 13 is activated, this is written as `R_motor=x0||x12, S_motor=x1||x7||x13, motor=S_motor&&!R_motor`.

Those algebraic equations are directly implemented in the PLC with Ladder diagram language, they are also used in the UPPAAL model with small syntactic variations as in Fig. 18.

The complete PLC cycle is modeled in UPPAAL by an automaton structured as a loop, which includes a clock to measure the cycle time (equal to 10 time units here). This loop consist of four steps:

1) input reading and computation of new values for the evolution conditions (`CFT`) of the SFC,
2) computation of other new values for SFC variables: step activation (`x`) and output computation (`motor ...`),
3) output writing, performed by a sequence of messages for synchronization with the operative part,
4) reset of the clock modeling the cycle time.

The atomicity hypothesis is the following: time can elapse only in the three states between these steps (**input_reading**, **computing**, **output_emission** in Fig. 18), to represent the duration of their execution. Note that the outputs are usually emitted simultaneously by the PLC, so that the duration of the ouput emission is negligible with respect to the cycle time. We ensure this atomicity property of step (3) with committed locations.

2) The event-driven program: Since it is run upon activation of the bushing-test position, the event-driven task is strongly dependent on the environment. This aspect is modeled by the emission of a message from the conveyor, received by the automaton of the event-driven task (see Fig. 19).

When the message *postest?* is received, the automaton executes the algebraic equations which represent the Ladder program and modifies the internal variable `motor` if the

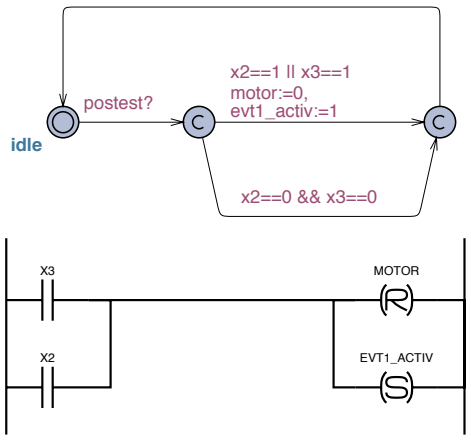

Figure 19. UPPAAL model for the event-driven task

condition holds. Note that the execution time of the even-driven task is taken as null due to its negligible duration. In practice, for a basic PLC, this execution time is about 1 microsecond while the cycle time of the main program is about 3 milliseconds.

The first *committed* location is used to model the priority of the event-driven task and the other ones to express the null duration.

The two programming designs are considered in order to determine the conditions under which the requirements are satisfied and to compare both models:

- the event-driven task only modifies the internal memory of the output as defined above (see Fig. 17),
- the event-driven task is not activated and the main task is designed to control the whole process (see Fig. 8).

V. Verification with Uppaal

A. The observer automaton.

In order to verify the timed property **P2**, we need to introduce an additional automaton (see Fig. 20) which is composed with the rest of the system. This automaton plays the role of an external observer and does not interfere with the model previously described.

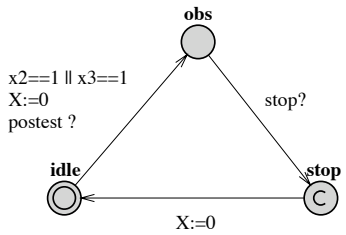

Figure 20. Observer automaton

This automaton contains a location **stop**, reached when the conveyor stops in testing position. It also contains a clock X to measure the reaction time. The observer automaton starts from state **idle** with X set to 0. When the message *postest?* is received from the conveyor, the automaton moves to state **obs** and resets the clock X. From this point on, the clock value again increases with time. When the message *stop?* is received from the main program, the automaton switches to state **stop** which must be committed to avoid time elapsing. Thus, the value of X in this last state corresponds to the time elapsed between the triggering of the event-driven task and the physical stop of the conveyor. To check the timed property **P2**, we express its negation (**C1** in the table I below): the observer automaton will eventually reach the state **stop** with the value of the clock X greater than 5 time units. Property **C1** is written as

$$\mathsf{E}\!<\!> (observer.stop \text{ and } X > 5)$$

in UPPAAL syntax, which is a fragment of the logic TCTL [1]. In this formula, the combination E<> means "for some path in the future" and *observer.stop* denotes location **stop** of the observer automaton. The formula above can thus be read as "for some paths in the future, the location **stop** of the observer is reached with the value of clock X greater than 5".

Note that if property **C1** is true in our abstracted model, it is also true in the real program. Indeed, if time can elapse in the intermediate locations of the main program, this will result in a possibly greater value for the clock X. On the contrary, if we obtained a negative answer, we would have to extend the model so that time could elapse in the location just before emission of the message *stop!*.

B. Experiments.

First note that the global model has about 30.10^6 configurations, which are explored in an on the fly computation of the set of reachable states.

Table I
RESULTS OF THE EXPERIMENTS

property	result	time	memory
with the event driven task			
C1:E<> observer.stop and $X > 5$	yes	15 s	30 Mb
C2:E<> observer.stop and $X \leq 5$	yes	15 s	30 Mb
C3:E<> observer.stop and $X > 10$	no	22 s	61 Mb
without the event driven task			
C5:E<> observer.stop and $X \geq 10$	yes	16 s	30 Mb
C6:E<> observer.stop and $X > 20$	no	22 s	70 Mb
C7:E<> observer.stop and $X < 10$	no	22 s	69 Mb
with Mader-Wupper model			
C8:E<> observer.stop and $X > 5$	-	-	-

Table I gives the time and memory used for verification (on a linux machine with a pentium4 at 2.4 GHz with 3 Gb RAM). The results provide a comparison of the reaction times between mono-task and multitask programming. Indeed, on the one hand, properties **C5**, **C6** and **C7** show that the conveyor stops between 10 and 20 time units after it reaches the test position. This is far from being a surprise because these values correspond respectively to one and two PLC cycle times. On the other hand, property **C3** shows that the conveyor stops in less than one PLC cycle time (which would also be the case if time could elapse in intermediate locations because the length of the cycle is preserved). Thus, multitask programming reduces the reaction time. However, property **C1** proves that

it is not sufficient to satisfy the requirement **P2**.
Note that, after 29 hours of computation, we stopped the verification process in the case of Mader-Wupper model.
These performances are due to two main reasons: the atomicity hypothesis for executions between some states of the main program and the enhanced model of the TON block.

- The atomicity hypothesis: we assume that each one of the four steps of the main program (section IV-E1) executes instantaneously. Recall that time can elapse only in three states (**input_reading**, **computing**, **output_emission**).
- The enhanced model of the TON block: we use one broadcast channel to synchronize all the TON blocks and the main program instead of three ordinary channels for each TON block as in Mader-Wupper model.

VI. Discusion and conclusion

In this work, we use a generic algebraic translation of SFC specification with TON blocks which leads to formal semantics for a subset of Ladder diagram language, with timed automata. The loop structure of the automaton is generic, and updates on the transitions result directly from the algebraic equations.

We also describe the operative part of station 2 of MSS platform with timed automata. This part is not generic in this paper, but some work has been devoted to this problem [14]. On this network of timed automata represented in UPPAAL syntax, we formally prove by model-checking that multitask programming reduces the reaction time of the conveyor, upon emission of an output order to stop. While this does not really come as a surprise, we obtain reasonable verification times (less than 30 s) on a global model with about 30.10^6 states, by adding an atomicity hypothesis to Mader-Wupper model and modifying the automata for timer blocks. In comparison, model-checking the same formula with the original model had to be stopped after several hours.

This experiment shows that timed model-checking is a useful technique for the verification of PLC programs. Attention must be focused on the following points:

- The semantics of the language must be formally defined. Here, due to the algebraic translation of SFC specifications and the loop structure of the PLC cycle, the construction of a timed automaton and a control program in Ladder diagram language could be generated in a systematic way. This construction process could be done automatically in the future.
- Timed aspects can be integrated with respect to some constraints (logical time for the control program, discrete time for plant model).
- The technique requires hypotheses on the behavior of the operative part, on the real-time kernel of the PLC, to build a compact enough model of the system.

Taking these conditions into account may lead to efficient formal verification of timed properties. We believe that design-based approaches (e.g. synchronous languages) would not fit this objective. While design-based approaches can also be used for formal verification purposes, we think that they are not so convenient to deal with the asynchronous context of PLC programming.

Scaling remains the main challenge for verification techniques due to the combinatorial explosion: combining asynchronous processes will result in a very high number of configurations even with untimed models. Here we choose to present a timed approach combined with multitask processes in one PLC. Adding more processing stations controlled by the same PLC would not be a real problem. However, adding a processing station controlled by another PLC leads to combinatorial explosion due to a fine description of the PLC behavior.

Although this study is not yet at industrial scale, our approach aims at techniques that can be applied in an industrial context, with commonly used languages as starting point. This can provide tools to enhance the quality of control programs with the usual industrial practices.

References

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP'90), Warwick University, England, July 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[3] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science (TCS)*, 126(2):183–235, 1994.

[4] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and Ph. Schnoebelen. Towards the automatic verification of PLC programs written in Instruction List. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics (SMC'2000), Nashville, TN, USA, Oct. 2000*, pages 2449–2454, 2000.

[5] A. David, G. Behrmann, K. G. Larsen, and W. Yi. A Tool Architecture for the Next Generation of UPPAAL. Technical Report 2003-011, Department of Information Technology, Uppsala University, Feb. 2003. 20 pages.

[6] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proc. Workshop Hybrid Systems III: Verification and Control, New Brunswick, NJ, USA, Oct. 1995*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[7] H. Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. *Theoretical Comput. Sci.*, 253(1):61–93, 2000.

[8] G. Frey and L. Litz. Formal methods in PLC-programming. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics (SMC'2000), Nashville, TN, USA, Oct. 2000*, pages 2431–2436, 2000.

[9] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures In *Parallel and Distributed Processing Symposium*, 2006 , 2006. DOI: 10.1109/IPDPS.2006.1639422

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In *Proc. 1st Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95), Aarhus, DK, May 1995*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[11] IEC (International Electrotechnical Commission). *IEC Standard 61131-3 : Programmable controllers - Part 3*, 1993.

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

[13] J. Machado, B. Denis, J.-J. Lesage, J.-M. Faure, J. Ferreira Da Silva. Logic controllers dependability verification using a plant model In *Proc. 3rd IFAC Workshop on Discrete-Event System Design, (DESDes'06), Rydzyna (Poland), Sept. 2006*, pages 37-42.

[14] J. Machado, B. Denis, J.-J. Lesage. A generic approach to build plant models for DES verification purposes In *8th International Workshop On Discrete Event Systems, WODES'06, Ann Arbor (USA), July 2006*, pages 407-412.

[15] A. Mader and H. Wupper. Timed automaton models for simple programmable logic controllers. In *Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS'99), York, UK, June 1999*, pages 114–122. IEEE Comp. Soc. Press, 1999.

[16] E. Olderog. Correct real-time software for programmable logic controllers. In *Correct System Design. Recent Insights and Advances*, volume 1710 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 1999.

[17] Rexroth Bosch Group. Mechatronik standard system. http://www.boschrexroth.com/country_units/europe/germany/sub_websites/brs_germany/de/didactic/lehrsysteme/mechatronik/mechatronik_standard_system_mss/index.jsp.

[18] O. Rossi, O. de Smet, S. Lampérière-Couffin, J.-J. Lesage, H. Papini, and H. Guennec. Formal verification: a tool to improve the safety of control systems. In *4th Symposium on Fault Detection, Supervision and Safety for Technical Processes (IFAC Safeprocess 2000), Budapest, Hungary*, pages 885–890, 2000.

[19] O. Rossi and Ph. Schnoebelen. Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs. In *Proc. 4th Int. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM'2000), Dortmund, Germany, Sept. 2000*, pages 177–182. Shaker Verlag, Aachen, Germany, 2000.

[20] Siemens. Programming with STEP 7 v5.4 ref: A5E00706944-01 http://www.automation.siemens.com

[21] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proc. 13th Annual Symposium on Theoretical Computer Science (STACS'96)* , volume 1046 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 1996.

[22] F.W. Vaandrager and A.L. de Groot. Analysis of a biphase mark protocol with Uppaal and PVS In *Formal Aspects of Computing*, Volume 18, number 4, pages 433–458. Springer, 2006.

[23] L. Waszniowski and Z. Hanzlek. Formal verication of multitasking applications based on timed automata model In *Real-Time Syst*, Volume 38, pages 39–65, 2008 DOI 10.1007/s11241-007-9036

Houda Bel mokadem received a PhD degree from École Normale Supérieure de Cachan (France). She is associate professor at ENSA de Tanger (Morocco). Her research area is the verification of temporal properties.

Béatrice Bérard is full professor at University Pierre et Marie Curie. Her research area is the modeling and verification of concurrent systems with quantitative constraints, in particular real time and hybrid systems.

Vincent Gourcuff received a PhD degree from École Normale Supérieure de Cachan (France). He is qualified teacher at Institute of Technology of Cachan (France). He carries out research on the verification of safety properties on PLC programs.

Olivier De Smet is currently associate professor of manufacturing engineering at Conservatoire National des Arts et Métier (Paris, France) and carries out research at the LURPA on the control of Discrete Event Systems with verification approaches.

Jean-Marc Roussel received a PhD degree in 1994. He is currently associate professor of automatic control at École Normale Supérieure de Cachan (France) and carries out research at the LURPA on the control of Discrete Event Systems with algebraic approaches.