Applying REMES behavioral modeling to PLC systems

Aneta Vulgarakis
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University
Västerås, Sweden
aneta.vulgarakis@mdh.se

Aida Čaušević
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University
Västerås, Sweden
aida.delic@mdh.se

Abstract—**Programmable logic controllers (PLCs), as a specialized type of embedded systems, have been introduced to increase system flexibility and reliability, but at the same time to give faster response time and lower cost of implementation. In the beginning, their use brought a revolutionary change, but with the constant growth of system complexity, it became harder to guarantee both functional and extra functional properties, as early as possible in the development process. In this paper, we show how formal methods can be applied to describe PLC-based systems and illustrate it on an example of a car wash system. First, we show how the existing behavioral modeling language REMES (REsource Model for Embedded Systems) can be extended to model the behavior of such systems. Second, we show how REMES can be translated into networks of timed automata and priced timed automata in order to support safety and resource-wise reasoning about PLC systems. The formal verification of PLC systems is carried out in the UPPAAL and UPPAAL CORA tools.**

Keywords - behavioral model; model checking; PLC systems; timed automata; priced timed automata

I. INTRODUCTION

Programmable logic controllers (PLCs) are a specialized type of embedded systems used to control machines and processes. They have been introduced in the early 1970s to replace the existing relay control logic that became obsolete and expensive for implementing systems at that time. On the other hand, PLCs have offered flexibility, higher reliability, better communication possibilities, faster response time, easier troubleshooting, and lower cost [14]. So far, PLCs have been mainly of interest for industrial control engineers that introduced, developed, and standardized their own design methods and programming languages.

Formal methods and in particular model checking of PLC-based systems have not received much attention and it is important to emphasize that they represent important means to analyze and ensure accuracy and reliability of such complex systems. Nevertheless, some efforts have been devoted to formal modeling and analysis of PLC systems [7, 8, 9, 10, 11], but without any emphasis on resource usage. In this paper we show in a small car wash system example how REMES can be extended for modeling and analyzing of PLC systems.

This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and by the European Union under the ICT priority of the 7th Research Framework Programme in the context of the Q-ImPrESS research project (www.q-impress.eu).

REMES (**RE**source **M**odel for **E**mbedded **S**ystems) [1] is a behavioral modeling language that provides formal modeling and analysis of resource-wise behavior of embedded systems. It can be seen as a powerful way to model both functional and extra-functional behavior of PLCs and to reason about their critical properties (e.g., resource consumption). The possibility to be translated into networks of timed automata [2] and priced timed automata [3], depending on what type of the analysis is required (i.e., functional and timing properties or cost analysis, respectively) distinguishes REMES from similar modeling languages. REMES may be used from both researchers and practitioners, since it is easy to be used and it abstracts away from unnecessary details.

In brief, our contribution is threefold:

- Showing how PLC systems can be formally described, modeled, and verified by using an extended version of REMES for PLC systems.
- Demonstrating on a small example how the extended version of REMES can be used to describe behavior of PLC-based systems.
- Showing how critical PLC system resources can be annotated and reason about them in priced timed automata environment. The intention is to show how number of experiments can be done prior to PLC system implementation without increasing development cost.

The remainder of the paper is organized as follows. Section 2 presents a brief introduction to PLC systems, the language REMES, Computation Tree Logic, Timed Automata, and Priced Timed Automata. Section 3 presents how REMES may be extended to model behavior of PLC systems. Section 4 models and analysis a car wash PLC-based system in REMES, and Section 5 concludes the paper.

II. PRELIMINARIES

A. PLC systems

A PLC uses a programmable memory to store instructions and specific functions that include on/off control, counting, timing, sequencing, arithmetic and data handling. It can

communicate with other controllers and computer environment. The components of a PLC system can be synchronized by passing signals via channels.

The PLCs considered here have PLC architecture and operational structure as depicted in Fig.1 and Fig.2, respectively. During each operating cycle a PLC reads the status of sensors from the environment and places their actual state in a memory location accessible to a PLC program. Subsequently the instructions that are programmed into the user program of the PLC are calculated and executed and the results of that computation are written to other memory locations. In the last step of the cycle, these values are mapped to the output devices controlled by the PLC. After this output is produced, a new cycle may start.

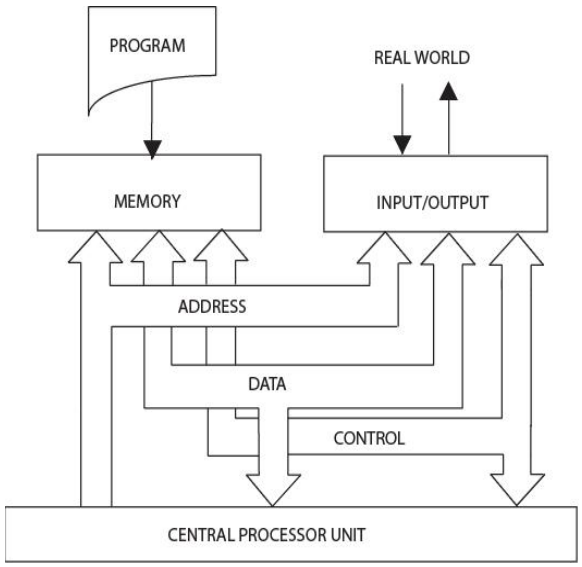

Figure1. A PLC architecture

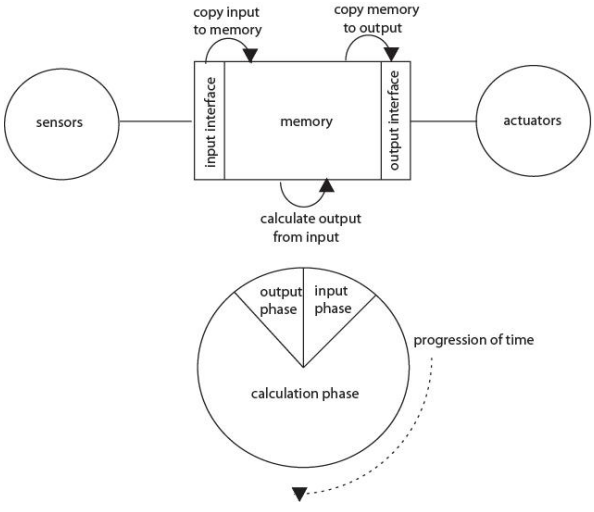

Figure2. The operational structure of a PLC

B. *REMES*

REsource **M**odel for **E**mbedded **S**ystems (REMES) is introduced in [1] in order to describe resource-wise behavior of embedded systems. REMES supports modeling of both continuous (e.g., energy) and discrete resources (e.g., memory, access to external devices, etc.). The behavior of embedded components in REMES is described by a *mode*. A mode can be either *atomic* or *composite* depending on whether it contains number of submodes or not. Well-defined *data interfaces* are used for data transfer. Every REMES mode has a *control interface* in terms of *entry* and *exit* points. Similar to other languages, for each mode one can define both *local* and *global* variables. Additionally, in REMES an extra variable evolving at rate 1, annotated as *clock,* is introduced.

A resource-wise continuous behavior can be annotated for each (sub)mode assuming that the corresponding component consumes resources. The consumption of a resource *c* represents an accumulated resource usage up to some point of time, while the derivative of c, denoted as *c'*, is a rate of consumption over time. Seceleanu *et al.* [1] give classification of resources due to their nature of being referable and non-referable and whether they are continuous or discrete. For example, memory is a discrete referable resource since it can be dynamically allocated/deallocated, addressed, and manipulated at run-time.

The control flow is defined by *edges* (i.e., set of directed lines) that connect control points of (sub)modes. REMES supports *delay/timed* actions and *discrete* actions. A delay/timed action describes the continuous behavior of a mode, and its execution does not change the mode, while a discrete action represents instantaneous action, which execution changes the mode. A discrete action can be executed if the corresponding *guard g* annotated for an action holds. An *Invariant Inv* specifies for how long a (sub)mode is executed. If the invariant stops to hold, mode is immediately exited through one of its exit edges. A mode can be defined as *urgent*. Urgency means that mode is exited right-away after its activation preventing a delay to happen. It is possible to define a *conditional connector C* that enables the selection of an outgoing edge out of two or more existing ones, based on the guarding boolean conditions of the discrete actions that correspond to the exiting edges. An action can be taken only when its guard evaluates to true.

C. *Computation Tree Logic*

Computation tree logic (CTL) [5] is a specification language for finite state systems. With use of CTL it is possible to reason about sequence of events.

Let AP be a set of atomic propositions, $a \in AP$. A CTL formula ϕ is defined as follows:

$$\phi ::= true \mid false \mid a \mid \neg a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid EX\phi \mid$$
$$AX\phi \mid EF\phi \mid AF\phi \mid EG\phi \mid AG\phi \mid E\phi_1 U \phi_2 \mid A\phi_1 U \phi_2$$

Each CTL operator is a pair of symbols. The first one is either A ("for All paths"), or E ("there Exists a path"). The second one is one of X ("neXt state"), F ("in a Future state"), G ("Globally in the future") or U ("Until"). For example $EX\phi$ means that there exists an immediate successor state, reachable by executing one step, in which ϕ holds.

D. Timed Automata

A timed automaton [2] is a finite-state machine enriched with a set of clocks. All clocks in one system are synchronized and they are assumed to be real-valued functions of time elapsed between events.

Formally, let assume a finite set of real-valued variables C ranged over by x, y, etc. standing for clocks and a finite alphabet Σ ranged over by a, b, etc. standing for actions.

A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in C, \sim \in \{\leq, <, =, >, \geq\}$ and $n \in N$. Clock constraints are used as guards for timed automata. $\chi(C)$ is used to denote the set of clock constraints, ranged over by g.

Definition 1 (Timed Automaton) *A timed automaton A is a tuple $\langle N, l_0, E, I \rangle$ where:*

- *N is a finite set of locations,*
- *$l_0 \in N$ is the initial location,*
- *$E \subseteq N \times \chi(C) \times \Sigma \times 2^C \times N$ is set of edges and*
- *$I : N \to \chi(C)$ assigns invariants to locations*

We will write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l' \rangle \in E$.

Timed automata semantics is defined as a transition system where each state consists of the current location and the current values of clock. A transition may be either a *delay transition* (the automaton may delay for some time), or an *action transition* (the automaton follow an enabled edge).

Let's assume that u is clock assignment mapping C to the non-negative reals R_+, $u \in g$ with meaning that the clock values denoted by u satisfy guard g. For each $d \in R_+$, $u + d$ denotes the clock assignment that maps all $x \in C$ to $u(x) + d$, and for $r \subseteq C$ let $[r \mapsto 0]u$ denote the clock assignment that maps all clocks in r to 0 and agree with u for the other clocks in $C \setminus r$.

Reachability analysis is one of the most useful analyses to perform on a given timed automaton. The reachability problem can be defined as follows: *Given two states of the system, is there an execution starting at one of them that reaches the other?* This state may be used to describe safety properties of the analyzed system.

Definition 3 *We shall write $\langle l, u \rangle \to \langle l', u' \rangle$ if $\langle l, u \rangle \xrightarrow{\sigma} \langle l', u' \rangle$ for some $\sigma \in \Sigma \cup R_+$. For an automaton with initial state $\langle l_0, u_0 \rangle, \langle l, u \rangle$, is reachable iff $\langle l_0, u_0 \rangle \to^* \langle l, u \rangle$. More generally, given constraint $\phi \in \chi(c)$ we say that the configuration $\langle l, \phi \rangle$ is reachable if $\langle l, u \rangle$ is reachable for some u satisfying ϕ.*

Properties of TA can be specified in the Timed Computation Tree Logic (TCTL), which is an extension of CTL with clocks. We refer the reader to [12] for details of TCTL.

E. Priced Timed Automata

In this subsection we will recall a formal definition of priced timed automata (PTA) [3, 4] and their semantics. Let's assume that C is set of clocks that are non-negative real valued variables. They can be reset to zero or have a fixed growing rate in time. A PTA over C is an annotated directed graph that consists of locations. One of them is marked as an initial location. An edge connects two locations and is decorated with a guard, an action, and a reset set. An edge is enabled only if its guard evaluates to true and the source location is active. A reset set is a set of clocks reset to zero whenever the edge is taken. Locations are decorated with invariants and invariant must evaluate to true whenever its location is active. A PTA has cost and cost annotation on both edges and locations, respectively.

Let C be a finite set of clocks and $\chi(C)$ the set of formulas obtained as conjunctions of atomic constraints of the form $c \bowtie n$ where $c \in C$, $n \in N$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. The elements of $\chi(C)$ are called clock constraints over C.

Definition 1 (Priced Timed Automata) *Let C be a set of clocks and Act a set of actions. A priced timed automata over C and Act is a tuple $A = (L, E, l_0, I, P)$, where L is a set of locations, $E \subseteq L \times \chi(C) \times Act \times 2^c \times L$ is a set of edges, $l_0 \in L$ is the initial location, $I : L \to \chi(C)$ assign invariants to locations, and $P : L \cup E \to N_0$ assigns cost rates and cost to locations and edges respectively.*

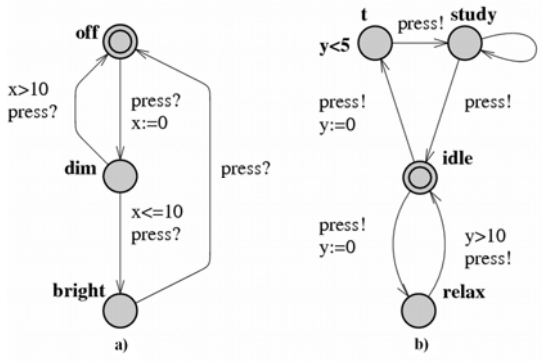

Figure3. An example of simple timed automata

In Fig.3 is depicted a simple network of timed automata. Each automaton consists of a set of locations (represented as circles) and edges (represented as directed lines). A location marked with a double circle is an initial location (e.g., locations off and idle in Fig.3). The timing behavior is controlled by two clocks x and y. For each location it is possible to assign an invariant value that must hold in order to stay in that location (e.g., in Fig.3 b) invariant y<5 must hold in order to stay in location t). Two automata communicate using synchronization channels, which are special types of global variables (e.g., in Fig.3 variable press is global variable of type channel). Further, for each edge it is possible to specify guards i.e., boolean expressions that must hold in order an edge to be taken. All variables defined for automata can be updated on edges (e.g., x: =0).

Priced timed automata semantics is defined as a priced transition system. It is a labeled transition system, where the transition relation is given by a partial function from transitions to the non-negative reals, intuitively being the cost of the transition. We write $s \xrightarrow{a}_{p} s'$ whenever the function is defined on the transition (s, a, s') and the cost is p [3, 4].

One can distinguish between discrete and delayed transitions. First change the control location of the automaton without time passing, while latter are transitions that pass time in a fixed control location.

To be able to specify properties of PTA, the Weighted Computation Tree Logic (WCTL) has been introduced [6]. WCTL is an extension of TCTL with possibility to reset and test cost variables. We interpret formulas of WCTL over labeled PTA, that is, PTA having a labeling function that associates with every location *l* a subset of atomic proposition. We refer reader to [7] for a thorough description of WCTL.

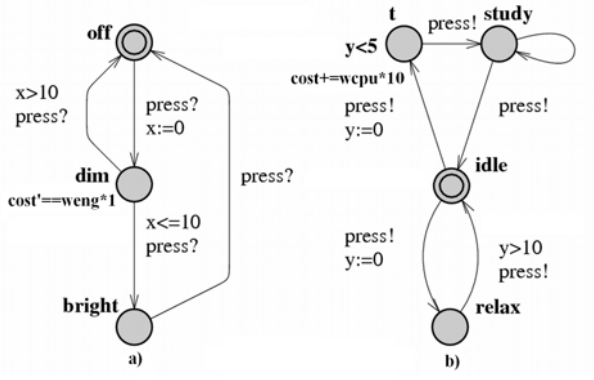

Figure4. An example of simple priced timed automata

Fig.4 extends Fig.3 with a notion of cost. As one can see it is possible to annotate two types of costs that capture resource consumption. In Fig. 4 b) is depicted accumulated resource usage up to some point in time (e.g., cost += wcpu * 10), and in Fig.4 a) is presented a rate of resource consumption over time (e.g., cost' == weng * 1). In Fig.4 wcpu and weng are the assigned weights for resources CPU and energy, respectively.

III. EXTENDING REMES FOR PLC SYSTEM ANALYSIS

In comparison with traditional REMES modes that run to execution, PLC systems can have interrupts in their operating cycles. Therefore, in order to be able to capture interrupts in REMES, we have enriched it with special *history* variables, like in CHARON [13]. When an execution of a mode is interrupted, the control state of that mode is recorded into a history variable *h*, a new local variable that we introduce for every composite mode. Next time when the mode is entered, the control state of that mode is restored according to the value of the history variable. The history variable *h* of mode M contains the names of the submodes of M as values, or a special value ε, which denotes that the mode is not active. A submode SM of composite mode M is called *active* when the history variable of M has the value SM. Additionally, in the extended REMES version for modeling behavior of PLC systems, guards are modeled as a conjunction between regular guards over variables, guards over history variables, and guards over synchronization channels. Synchronization channels are global variables used to establish communication between composite modes.

IV. AN ILLUSTRATIVE EXAMPLE: A CAR WASH CONTROLLER

In order to demonstrate how REMES can be used for resource-wise modeling and analysis of PLC systems, as an illustrative example, we use a simplified version of a car wash system. The car wash system is equipped with double doors. The details of this control system are depicted in Fig.5. There are three photoelectric sensors: s1, s2, and s3. It is assumed that a car has reached the washing position when sensor s3 has been activated. Notice, in one moment only one car may be washed. The washing process lasts for 50 time units and then the car can exist through Door2. Door1 and Door2 are open for four time units.

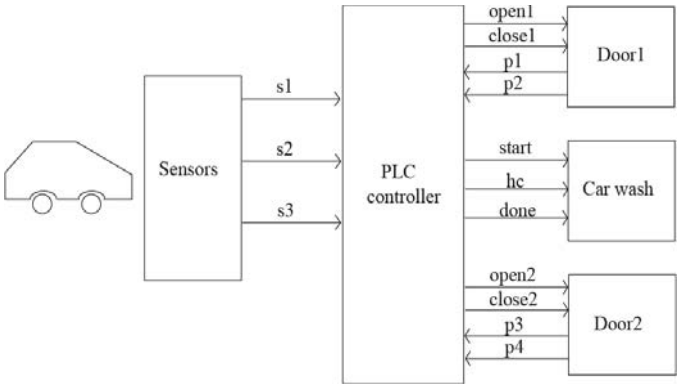

Figure5. The car wash system

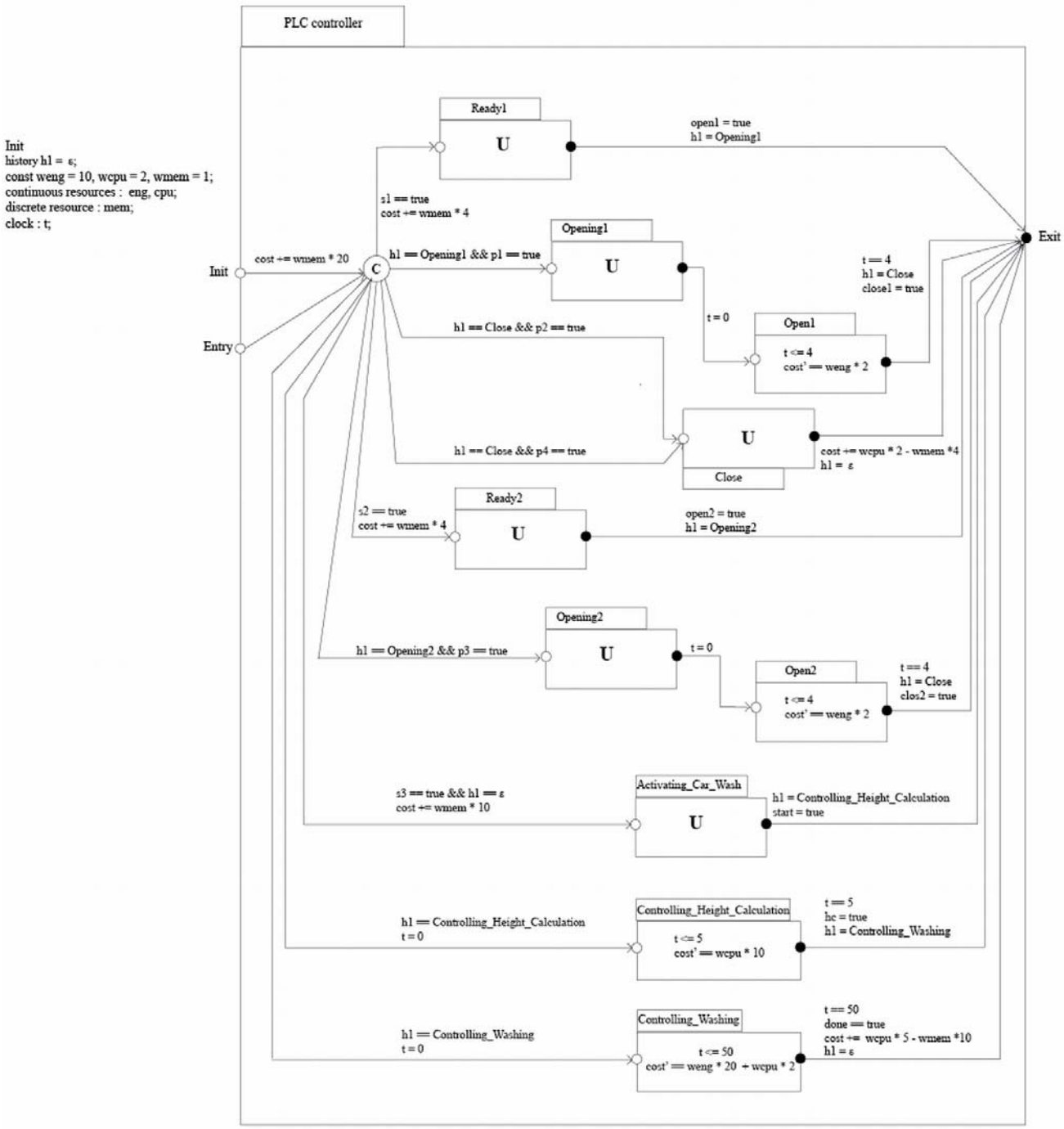

Figure6. REMES composite mode of PLC controller

A. A REMES model of Car Wash Controller

An abstract overview of the car wash system is depicted in Fig.5. The system consists of five parts: ***PLC controller***, ***Door1***, ***Car wash***, ***Door2***, and ***Sensors.*** The sensors are treated as inputs on channels that invoke process flow. Further, an abstracted version of a car is modeled in order to show the states through which the car goes during the washing process. Fig.6 shows the REMES composite mode of the ***PLC controller***. The component ***PLC controller*** first activates the opening of ***Door1*** when sensor ***s1***, modeled as a global variable, indicates that a car is approaching to the washing facility. The clock ***t*** bounds the total time for opening and closing of ***Door1*** to four time units. In the next step, when car has reached the washing position, signal from sensor ***s3,*** also modeled as a global variable, becomes active and car height is being calculated. Calculation is limited to exactly five time units, after which the car washing

facility proceeds with the car washing process that lasts fifty time units. In the last step Door2 is being opened so that the car may leave the car washing facility. This process starts with activation of sensor s2, modeled as a global variable.

Since we are interested in modeling the resource usage of the described system, we make use of three resources: energy, CPU, and memory. These resources belong to two different classes of the taxonomy as described in [1]. In our example we treat only static and simple dynamic memory. Allocation of dynamic memory is done when the ***PLC controller*** mode is entered for each of its cycles (controlling ***Door1***, controlling ***Car wash***, and controlling ***Door2***), and deallocation whenever the mode is exited. We assume that every simple CPU instruction utilizes one CPU tick.

As shown in Fig.6, the REMES mode of ***PLC controller*** supports interrupts. Whenever an interrupt occurs in the mode, the history variable ***h1*** captures the control state of the mode.

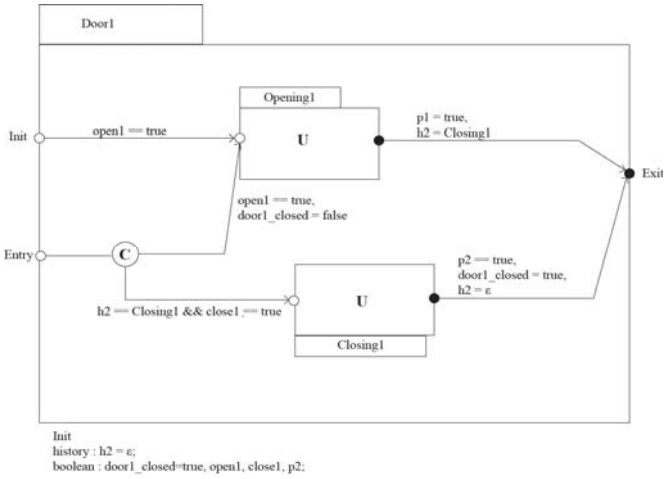

Figure7. REMES mode of Door1

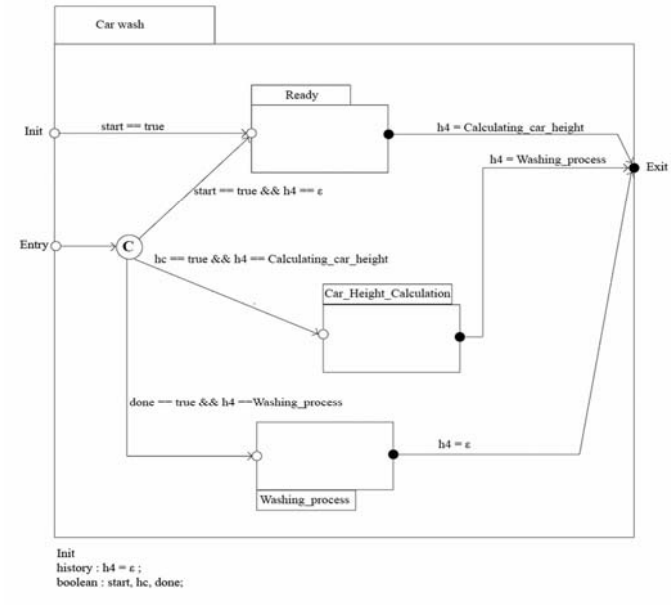

Figure8. REMES mode of Car wash

Fig.7 shows the REMES mode of ***Door1***. In order to start the door opening process, the boolean variable ***open1*** has to be true. When an interrupt occurs, history variable ***h2*** captures the control state (records which is the last visited state in order to know where to start in the next cycle) and variable ***p1*** becomes true. In order to start closing ***Door1,*** opened in the previous step, boolean variable ***close1*** has to be evaluated to true. The door can be closed and exited only when the mode guard ***p2 == true*** holds. After the door is being closed variable ***door1_closed*** evaluates to true. Boolean variable ***open1***, ***close1***, ***p1***, and ***p2*** model synchronization channels established with ***PLC controller***.

The REMES mode of ***Car wash*** is modeled as a composite mode that consists of three submodes: ***Ready***, ***Car_Height_Calculation***, and ***Washing_process,*** as shown in Fig.8. Boolean variables ***start***, ***hc***, and ***done*** model communication with PLC controller and depending on their values, submodes ***Ready***, ***Car_Height_Calculation***, and ***Washing_process*** are being visited, respectively. In order to know where to start in the following cycle, the history variable ***h4*** captures the control state of the mode, whenever an interrupt occurs in the mode.

In similar fashion all other system components can be modeled in terms of the extended REMES modeling language, but due to space limitation we choose to show only the ones shown in Fig.6, Fig.7, and Fig.8. In the next subsection, detailed PTA models of all the components are shown and described.

B. A PTA model of Car Wash Controller

The REMES-based Car Wash Controller has been analyzed in UPPAAL CORA [1] as a network of five PTA models: ***PLC controller***, ***Door1***, ***Door2***, ***Car wash***, and ***Car.***

The ***PLC controller*** model consists of twelve locations as depicted in Fig.9. Initially a static memory of 20 memory units is assigned to the PLC controller. Then it synchronizes on channel ***s1*** with ***Car*** that is approaching to the car washing facility. With the synchronization channels ***open1***, ***open2***, ***p1***, ***p2***, ***p3***, ***p4***, ***close1*** and ***close2***, the opening/closing of ***Door1*** and ***Door2*** is being controlled. Clock ***t*** is used to limit the opening of ***Door1*** and ***Door2*** to four time units. When a car is inside the washing facility ***PLC controller*** gets synchronization signal ***s3*** and sends synchronization signal ***start*** to initiate calculation of the car height. The channel ***hc*** is used to start the washing process that lasts exactly fifty time units. On the channel ***done PLC controller*** synchronizes with ***Car wash*** to finish the process. The model of ***Door1***/***Door2*** consists of four locations: c***losed***, ***opening***, ***open***, and ***closing*** as shown in Fig.10 and Fig.11. The synchronization channels ***open1***, ***open2***, ***p1***, ***p2***, ***p3***, ***p4***, ***close1***, and ***close2*** are used for communication with the model of ***PLC controller***. Boolean variables ***door1_closed*** and ***door2_closed*** are used to indicate the status of the doors. The doors are closed when both ***door1_closed*** and ***door2_closed*** are evaluated to true. The initial position of the doors is closed.

[1] For more information on UPPAAL CORA visit web page uppaal.org/cora.

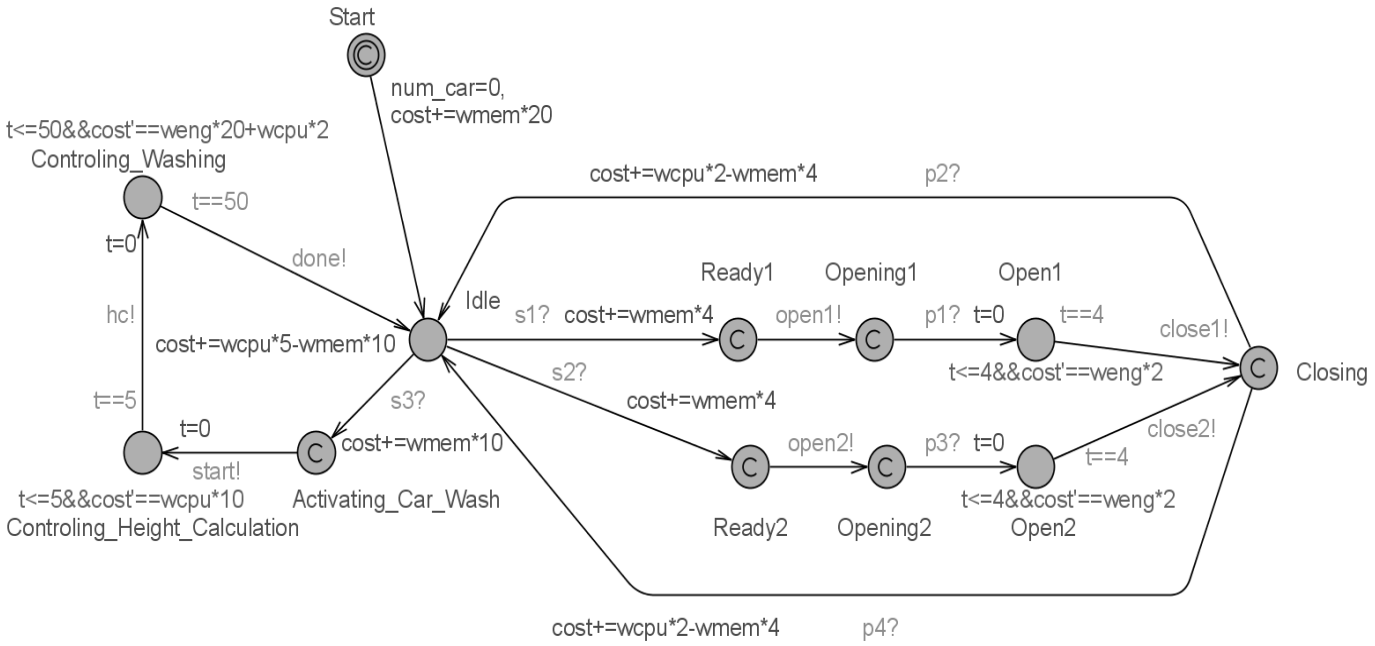

Figure9. The PTA model of PLC controller

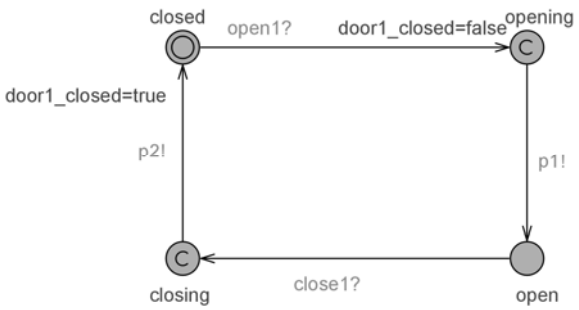

Figure10. The PTA behavior model of Door1

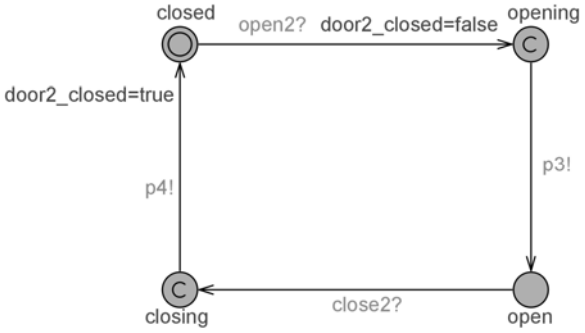

Figure11. The PTA behavior model of Door2

The ***Car wash*** PTA model, as shown in Fig.12, includes three locations: ***Idle***, ***Calculating_Car_Height***, and ***Washing_Process***. The synchronization channels: ***start***, ***hc***, and ***done*** are used to synchronize with ***PLC controller***.

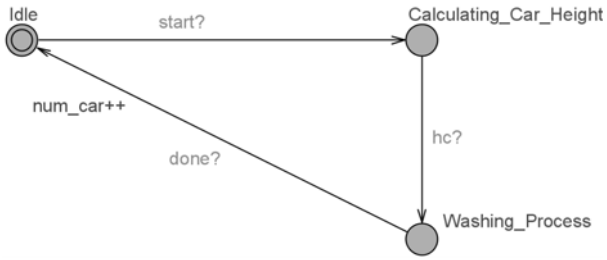

Figure12. The PTA behavior model of Car wash

The PTA model of ***Car*** is given in Fig.13. The behavior of a car is modeled as a simple PTA that consists of three locations: ***Approaching***, ***Inside***, and ***Washing***. In order to synchronize with the model of ***PLC controller*** channels ***s1***, ***s2***, and ***s3*** are used.

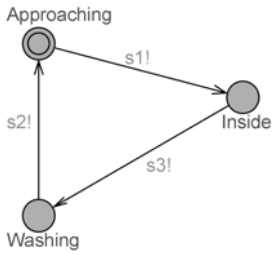

Figure13. The PTA behavior model of a Car

C. Formal Analysis of the PTA model

We encode the resource analysis problem as a weighted sum of resource consumption.

$$c_{tot} = weng * c_{eng} + wcpu * c_{cpu} + wmem * c_{mem} \quad (1)$$

We have identified energy, CPU, and memory as resources of interest. Energy is assumed to be the most critical resource, followed by CPU. In this example memory is not recognized as highly critical resource but in order to make our example as realistic as possible, dynamic memory allocation/deallocation is included at the start/end of every PLC cycle.

The highest weight is given to energy and the lowest to memory. The cost of resource consumption is influenced by the individual weights of each resource, and utilized resources on transitions and locations. Since UPPAAL CORA supports only resources that are monotonically increasing, existing resource weights have to be fine-tuned. Total cost is calculated as given in Eq. (1).

We check the modeled system against the safety properties: *AG not deadlock* and *AG (Car_wash.Washing_Process → (door1_closed==true and door2_closed==true))*. The first property checks whether the modeled system is deadlock free in all existing traces, while the latter ensures that both doors are closed while a car is being washed preventing other car to enter. Deadlock refers to a specific condition when two or more processes are waiting for each other to release a resource. In our example we are interested in analyzing minimal cost reachability problem i.e., finding the optimal trace that results with the lowest possible resource cost for five cars to be washed, as described with the Eq. (2).

$$EF(num\_cars == 5) \quad (2)$$

UPPAAL CORA has found that the minimal cost with respect to the annotated resources for five cars to be washed is 52330.

V. CONCLUSION

In this paper, we have presented how the existing REMES language [1] can be extended for modeling and analysis of PLC systems that are special type of embedded systems with hard constraints on resources such as energy, CPU, memory, etc. So far, they have found their use in industry, and just few results tackled the formal modeling and analysis of such systems.

The model, called REMES, is tailored for embedded systems, which makes it suitable for PLC systems, as well. In order to fully apply REMES for modeling behavior of PLC systems, we have enriched it with notion of interrupts and a special type of local variables called histories. REMES has a notion of resources that are classified by their discrete or continuous nature, how they can be consumed and/or allocated and released, and whether they can be referred to, or not. Moreover, REMES is developed to be used not only by academic staff, but also by control engineers that usually do not have a background in formal methods. Therefore, the use of REMES might reduce the development time of PLC systems and also bring savings in the development process justified by the fact that it provides early prediction of functional and extra functional properties of the system.

To illustrate the principles of REMES, we have shown an example how PLC systems can be modeled in REMES and analyzed in the framework of timed and priced timed automata. The presented example is a car wash PLC-based system that consumes energy, CPU, and memory resources. REMES is used to describe functionality, timing, and resource properties of components that are part of the car wash system. To calculate the optimal resource usage of the system, the resource-wise analysis problem in REMES is given as a cost, expressed by a weighted sum of consumed resources. The analysis of REMES models is carried out by a translation into the framework of timed- and (multi) priced timed automata. UPPAAL CORA tool is used to perform such analysis.

As future work we plan to investigate more PLC systems in order to provide realistic case studies in which REMES can be fully utilized.

REFERENCES

[1] C. Seceleanu, A. Vulgarakis, and P. Pettersson, "REMES: A Resource Model for Embedded Systems," In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009), IEEE Computer Society, June 2009.

[2] R. Alur and D. L Dill, "A Theory of Timed Automata," Theoretical Computer Science, 1994, pp.183-235.

[3] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn and F. Vaandrager, "Minimum-Cost Reachability for Priced Timed Automata," In Proc. of the 4th International Workshop on Hybris Systems: Computation and Control, Springer - Verlag, 2001, pp. 147-161.

[4] R.Alur, "Optimal Paths in Weighted Timed Automata.," In HSCC01 hybrid Syetems: Computation and Control, Springer 2001, pp.49-62.

[5] R. Alur, C. Courcoubetis, and D.L. Dill, "Model-Checking for Real Time Systems," In 5th Symposium on Logic in computer Science (LICS'90), 1990, pp.414-425.

[6] T. Brihaye, V. Bruyµere, and J-F. Raskin. "Model-Checking for Weighted Timed Automata," In Proc. of FORMATS-FTRTFT04, number 3253 in Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 277-292.

[7] A. Mader. "A Classification of PLC Model and Application," Discrete event systems-analysis and control, In Proc. WODES2000, 2000, USA.

[8] G. Canet, S. Couffin, J. J. Lesage, A. Petit, and P. Schnoebelen, "Towards the automatic verification of PLC programs written in Instruction List". In Proc. of the International Conference on Systems, Man and Cybernetics, October 2000, pp. 2449–2454.

[9] M. Heiner, and T. Menzel, "A Petri Net Semantics for the PLC Language Instruction List", IEEE Control, 1998, pp. 161–166.

[10] E. Clarke, O. Grunberg, and D. Peled, "Model Checking," The MIT Press Cambridge, England, 1999.

[11] R. Huuk, "Software Verification for Programmable Logic Controllers," PhD Thesis, Kiel, 2003.

[12] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking in Dense Real-time," Information and Computation 104(1):2-34,1993.

[13] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Skolsky, "Hierarchical Modeling and Analysis of Embedded Systems," In Proc. of the IEEE, 8(3):231-274, 1987.

[14] G. Dunning, Introduction to Programmable Logic Controllers, 3rd ed., Delmar Thomson Learning, 2005.