Applying REMES Behavioral Modeling to PLC Systems
Aneta Vulgarakis, Aida Čaušević

Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
(e-mail: aneta.vulgarakis, 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.

1. 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.

REMES (REsource Model for Embedded Systems) [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 behaviour 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 behaviour of PLC systems. Section 4 models and analysis a car wash PLC-based system in REMES, and Section 5 concludes the paper.
2. PRELIMINARIES

2.1 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 environments. 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.

2.2 REMES

RESource Model for Embedded Systems (REMES) is introduced in [1] in order to describe resource-wise behaviour of embedded systems. REMES supports modeling of both continuous (e.g., energy) and discrete resources (e.g., memory, access to external devices, etc.). The behaviour 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 behaviour 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 behaviour 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 \( \text{Inv} \) 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.

2.3 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 \( \phi \) is defined as follows:

\[
\phi := \text{true} \mid \text{false} \mid a \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \rightarrow \phi \mid \text{EX} \phi \\
\text{AX} \phi \mid \text{EF} \phi \mid \text{AF} \phi \mid \text{EG} \phi \mid \text{AG} \phi \mid \text{E} \phi U \phi \mid \text{A} \phi U \phi
\]

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 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 $\phi$ holds.

### 2.4 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 $\Sigma$ 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 $x, y \in C$, $\sim \in \{\leq, <, = >\}$ 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 $(N, I, e, I)$ where:

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

We will write $\bar{i} \rightarrow \bar{i}'$ when $(\bar{i}, g, a, r, \bar{i}') \in E$.

**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 $\bar{u}$ is clock assignment mapping $C$ to the non-negative reals $R_+$, $u \in g$ with meaning that the clock values denoted by $\bar{u}$ 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 $[r \mapsto 0]$ denote the clock assignment that maps all clocks in $r$ to 0 and agree with $\bar{u}$ 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 2** We shall write $\langle l, u \rangle \rightarrow \langle l, u' \rangle$ if $\langle l, u \rangle \xrightarrow{\sigma, R} \langle l, u' \rangle$ 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 $\langle l_0, u_0 \rangle \rightarrow \langle l, u \rangle$. 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 $\phi$.

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.

**Fig. 3.** 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 behaviour 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., $x = 0$).

### 2.5 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 $c \triangleright n$ where $c \in C$, $n \in N$, and $\triangleright \in \{\leq, \geq\}$. The elements of $\chi(C)$ are called clock constraints over $C$.

**Definition 3 (Priced Timed Automata)** Let $C$ be a set of clocks and $\text{Act}$ a set of actions. A priced timed automata over $C$ and $\text{Act}$ is a tuple $(L, E, l_0, I, P)$, where $L$ is a set of locations, $E \subseteq L \times \chi(C) \times \text{Act} \times \mathbb{Z} \times L$ is a set of edges, $l_0 \in L$ is the initial location, $I : L \rightarrow \chi(C)$ assign invariants to locations, and $P : L \cup E \rightarrow \mathbb{R}^+$ assigns cost rates and cost to locations and edges respectively.

**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 $s \xrightarrow{a,s} s'$ 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 propositions. We refer reader to [7] for a thorough description of WCTL.

3. 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 $\epsilon$, 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 behaviour 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.

4. 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.

Fig. 4. 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' == weng * 1). In Fig.4 wcpu and weng are the assigned weights for resources CPU and energy, respectively.

Fig. 5. The car wash system
4.1 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 PLC controller. The component PLC controller first activates the opening of Door1 when sensor $s_1$, 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 the car has reached the washing position, a signal
Fig. 7. REMES mode of Door1

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.

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.

4.2 A PTA model of Car Wash Controller

The REMES-based Car Wash Controller has been analyzed in UPPAAL CORA 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 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.

![Fig. 7. REMES mode of Door1](image1)

![Fig. 8. REMES mode of Car wash](image2)

1 For more information on UPPAAL CORA visit web page uppaal.org/cora.
Fig. 9. The PTA model of PLC controller

Fig. 10. The PTA behaviour model of Door1

Fig. 11. The PTA behaviour 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.

Fig. 12. The PTA behavior model of Car wash

The PTA model of Car is given in Fig. 13. The behaviour 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.

Fig. 13. The PTA behaviour model of a Car
4.3 Formal Analysis of PTA model

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

\[ c_{\text{tot}} = w_{\text{eng}} c_{\text{eng}} + w_{\text{cpu}} c_{\text{cpu}} + w_{\text{mem}} c_{\text{mem}} \]  
(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: \( AG \text{ not deadlock} \) and \( AG (\text{Car}_1\text{.washer.\text{Washing}_1} \rightarrow (\text{door}_1\text{.closed}==true \land \text{door}_2\text{.closed}==true)) \). 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).

\[ EF (\text{num}_1\text{.cars} == 5) \]  
(2)

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

5. CONCLUSIONS

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 behaviour 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

ACKNOWLEDGMENT

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).

Aneta Vulgarakis is a PhD student at Mälardalen University. She defended her licentiate thesis entitled "A Resource-Aware Component Model for Embedded Systems" in September 2009 at Mälardalen University, Västerås, Sweden. She received a M.Sc. in July 2006 at the Faculty of Electrical Engineering, Skopje, Macedonia. Her main research interests are component-based software engineering, formal models and verification techniques for constructing correct and predictable real-time embedded systems.

Aida Čaušević is a PhD student at Mälardalen University. She received a M.Sc. in computer science from Faculty of Electrical Engineering, University Sarajevo in September 2007. Her research interests include behavioral modeling and analysis of services, service composition and service refinement in service oriented systems using formal methods, model checking.