Formal Verification of IEC 61499 Enhanced with Timed Events

Viktor Shatrov, Valeriy Vyatkin

▶ **To cite this version:**

Viktor Shatrov, Valeriy Vyatkin. Formal Verification of IEC 61499 Enhanced with Timed Events. 11th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS), Jul 2020, Costa de Caparica, Portugal. pp.168-178, 10.1007/978-3-030-45124-0_16 . hal-03741571

HAL Id: hal-03741571
https://inria.hal.science/hal-03741571

Submitted on 1 Aug 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

This document is the original author manuscript of a paper submitted to an IFIP conference proceedings or other IFIP publication by Springer Nature. As such, there may be some differences in the official published version of the paper. Such differences, if any, are usually due to reformatting during preparation for publication or minor corrections made by the author(s) during final proofreading of the publication manuscript.

Formal Verification of IEC 61499 Enhanced with Timed Events

Viktor Shatrov[1] and Valeriy Vyatkin[1,2,3]

[1]ITMO University, Saint Petersburg, Russia {vvshatrov@yandex.ru}
[2]Aalto University, Helsinki, Finland
[3]Lulea University of Technology, Lulea, Sweden {vyatkin@ieee.org}

Abstract. Many applications of Cyber-Physical Systems (CPS) play a crucial role in shaping the quality of life. The malfunction of such systems can lead to dangerous consequences, hence safety takes an important role. One of the common issues of CPS is the variability of event delays. This paper addresses formal verification of system ability to withstand event delays. To achieve such property, we use the concept of event timestamps. The paper proposes changes in IEC 61499 syntax to introduce timestamps into the programming language. The application of new syntax introduces better opportunities for formal verification. We enhanced FB2SMV tool to generate SMV models automatically from the initial system. The paper presents a case study of an elevator system that illustrates the proposed approach. Results of verification show that the system can perform time-aware computations. The proposed approach has shown better verification performance compared to timestamps manually added through the data connections between function blocks.

Keywords: Cyber-physical systems, formal verification, IEC 61499, time-aware computations

1 Introduction

The application of the distributed Internet of Things (IoT) architecture in industrial control systems raises questions about the robustness of the system and guarantees of invariance of physical system behavior when the same software is executed on different IoT configurations. CPS software based on the IoT architecture is executed on distributed devices that communicate via network and changes in the physical environment such as wireless communication distortions, battery discharge, physical location change, and many other environmental parameters change may affect the behavior of the system. The cyber-physical agnosticism (CPA) is a property of automation software that was proposed in [1] and it is defined as the system's ability to maintain the programmed behavior of the physical process despite interference from the physical environment. Event timestamps for IEC 61499 were proposed in [1] to achieve the CPA property. We elaborate the proposal of [1] by introducing the extension of the language of function blocks (FBs), which is described by the standard IEC 61499, with event timestamps. A formal verification approach can be applied to

verify that CPA property holds. Formal verification of programs in the language of function blocks usually is carried out by translating the source program into a model in the input language of model-checkers. The review of early works on formal modeling of IEC 61499 was conducted in [11]. Work [13] presented a method of FB modelling with timed-automata. However, the actual execution of the algorithms has not been modelled, only the time of their execution, which allowed to verify the properties based only on the structure of the system, but not dependent on data. Works [7][12] proposed rules for formal modelling of IEC 61499 FB's for popular model checking environment of SMV using Abstract State Machines as an intermediate model. This allowed an automatic generation of models for the verifier. To support verification of function block systems there exists an open source tool called FB2SMV [4], which allows generating models of basic and composite function blocks in the language of the NuSMV[1] model-checker. However, this system does not support event timestamps so it cannot be used as is. A timestamp simulation approach was used in works [2][3] to check the CPA property. The following points can be made as a drawbacks of the simulation approach:

- The verifiable model does not correspond to the semantics of programs with timestamped events;
- The method requires numerous changes in the original system;
- It is not possible to utilize all the advantages of timestamps. For example, it is not possible to order events based on timestamps;
- The method uses *service interface function blocks* (SIFB), which depend on particular hardware, which makes automatic generation of models for verification impossible;
- Verification time increases dramatically because of the addition of numerous function blocks to simulate timestamps.

Taking this into consideration another approach to verify CPA property is required.

The rest of the paper is organized as follows. In Section 2 it is shown how verifying the CPA property relates to life improvement. In Section 3 we describe event timestamps semantics and propose special syntax for them. Section 4 describes the necessary formal models. Section 5 presents a case study of the elevator system. The paper is concluded with a summary, acknowledgments and references.

2 Relationship to Life Improvement

The technological advances of cyber-physical systems have enabled us to develop systems and applications that have changed our daily life. The impact of CPS on people's well-being has increased, but at the same time the possible harmful consequences of the malfunctioning of these systems have also increased. Application areas of CPS that contribute to life-improvement is very diverse. These areas include *personalized healthcare* with examples such as health monitoring systems and cardiac pacemakers; *transportation systems* with examples of adaptive cruise control and self-

[1] NuSMV: a new symbolic model checker. http://nusmv.fbk.eu/

braking cars; *building automation* systems with examples of lift control, escape systems and many others.

All the provided examples have a great impact on shaping the quality of life and at the same time malfunction of these systems can lead to dangerous consequences. Taking into account the points made, the safety and reliability of such systems become very important.

One of the possible causes of systems malfunctioning could be event delays. Event delays can occur because of the distributed nature of CPS. Wireless transmission interference, hardware reconfiguration, changes in network topology and many other reasons can cause event delays. Event delays can cause a violation of deadlines in systems that need to provide an immediate reaction if some condition occurs. Design of such systems must be robust against event delays, which is a feature of the cyber-physical agnosticism property. In this regard, methods for verifying the CPA property contribute to the safety of cyber-physical systems which is an essential part to improve human well-being.

3 Event Timestamps

3.1 Timestamps Semantic

The work [1] proposes to use event timestamps in IEC 61499 to achieve cyber-physical agnosticism. Events are considered as data structures containing timestamps.

Adding timestamps to events can help to solve several problems:

- Non-determinism of the system. One of the features of IEC 61499 is an event-triggered execution. If several events occur simultaneously at the event inputs of the function block, there is an uncertainty in which of the events should be processed. Adding timestamps directly to the events enables the events to be ordered and thus make the system execution deterministic;
- Robustness of the system to the event delays. Timestamps can be used directly by the control program to determine the occurrence of delays and to make the necessary adjustments related to changes in the physical process that have occurred during the delay of the event.

There are different approaches to defining the semantics of timestamps. The work [5] proposes to determine the deadline for each service FB by which the event initiated by the block should be processed. The time limit is determined according to the hardware specification. For example, it can depend on the polling rate at which the sensor updates the readings. By the time new data becomes available, the data from the previous poll is not relevant and should, therefore, be ignored by the system. However, it is worth noting that this approach makes the system dependent on the hardware configuration. If a sensor needs to be replaced with a sensor with a different polling rate, the control program is required to be changed, as the timestamps are directly related to the polling frequency of the sensor.

In [1] it was proposed to supplement the events with two timestamps not related to the hardware configuration: a stamp reflecting the time of event creation (*TB*) and a stamp reflecting the time of the last processing of the event in the system (*TL*). The paper [8] describes the formal semantics of composite FBs with timestamps on the basis

of the theory of abstract state machines. At the same time, a formal model for basic FBs and timers has not been defined.

During the propagation of an event signal through the system, the TB stamp keeps the time of the creation of the first event in the event chain. The first event in the chain is created by the service interface blocks and the value of the TB stamp is set. The TL stamp is set each time the event is processed in the chain. The process diagram is shown in Fig. 1 (EI/EO – names of event input/output; DI/DO – names of data input/output).

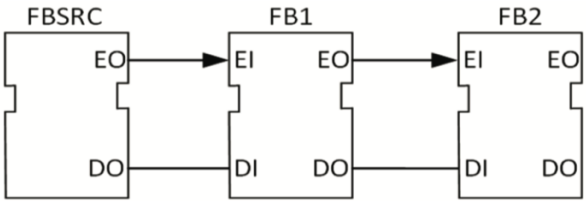

Fig. 1. Distribution of events [1].

At source functional block (FBSRC), timestamps are initialized with the value of the current system time:

```
FBSRC.EO.TL := System time in FBSRC;
FBSRC.EO.TB := System time in FBSRC;
```

When FB1 is invoked by the event produced by FBSRC, timestamp is copied from FBSRC.EO:

```
FB1.EI1 := FBSRC.EO;
FB1.INVOKEDBY:= FB1.EI1;
```

When FB1 emits event EO, birth timestamp is copied from event that invoked execution and last execution timestamp is assigned to current system time:

```
FB1.EO.TB := FB1.INVOKEDBY.TB;
FB1.EO.TL := System time in FB1;
```

3.2 Timestamps Syntax

To determine the necessary syntax changes to the IEC 61499 function blocks language, it is necessary to identify contexts in which access to information about event timestamps values can be required. These contexts are:

- Event dispatcher;
- Function block algorithms;
- Guard conditions of EC transitions.

In the first case, timestamps can be used by event dispatcher to order events. The event dispatcher is a system's internal component, and there is no explicit use of timestamps by the programmer.

In the second case, to enable usage of event timestamps values in algorithms, the syntax of the Structured Text (ST) language which is used in algorithms must be extended.

Considering the third case, IEC 61499 allows ST expressions as well as events to be used in guard conditions. Hence, the syntax from the second case may be applied.

To access event timestamps values, two new identifiers are needed. '*ts_born*' to access event creation time and '*ts_last*' to access event last processing time. In addition

to these identifiers, two more are needed. To compute event delay in guard conditions and ST algorithms, access to system time is needed. Therefore, we suggest adding an identifier 'Systemclock' to enable access to the current system time. Regarding the application of timestamps in algorithms, it is necessary to take into account that the execution of the same algorithm can be caused by different events. Therefore, it is necessary to provide an opportunity to determine the event that caused the execution of the algorithm. For this purpose, it is proposed to add the 'INVOKEDBY' identifier.

ST syntax is defined in the standard IEC 61131-3 [6]. Section B.3.1 defines production rules for expressions. To introduce the proposed identifiers, these production rules should be changed as follows:

B.3.1 Expressions PRODUCTION RULES

```
…
primary_expression ::= constant
      | enumerated_value
      | variable
      | '(' expression ')'
      | function_name
        '(' param_assignment {',' param_assignment} ')'
```

Proposed changes:

```
primary_expression ::= constant
    | enumerated_value
    | variable
    | '(' expression ')'
    | function_name
      '(' param_assignment { ',' param_assignment } ')'
    | structured_event
    | time_reference

time_reference ::= 'Systemclock'
structured_event ::= event_reference [ '.'
                                        time_attribute ]
event_reference ::= 'INVOKEDBY' | identifier
time_attribute ::= 'ts_last' | 'ts_born'
```

4 Formal Modelling and Verification

4.1 Function Blocks

Formal model of the IEC 61499 in abstract state machines was described in [7]. The most important parts and changes related to the introduction of event timestamps are given below. The notation used for the formal model is the same as in [7].

First of all, events defined as a tuple:

```
E = (V, TB, TL)
V – event variable, Dom(V) = {true, false}
```

```
TB – event creation time, Dom(TB) = Time
TL – event last processing time, Dom(TL) = Time
Time = {undef, 0, 1, ...} – set of discrete time values
                                in the system.
```

The event inputs are reset before the Execution Control Chart (ECC) Operation State Machine (OSM) is started. Therefore, the basic FB needs a buffer to save the timestamps of the event that activated the execution of the block. When event inputs are reset, timestamps are stored in the IB (InvokedBy) buffer. Buffer values are used to access timestamps from algorithms, guard conditions of EC transitions, as well as to create new events.

Transition rules for the function of resetting the event inputs (Rule Set 1 [7]) is redefined by the following rules:

$$p_{EI}^{B,1}[k] : S = s_1 \wedge selectEI_k \rightarrow IB = (Z_{TB}(ei_k), Z_{TL}(ei_k))$$
$$\wedge Z_V(ei_k) = false | ei_k \in EI$$
$$p_{EI}^{B,2}[m] : \alpha \wedge S = s_0 \wedge \overline{selectEI_m} \rightarrow Z_V(ei_m) = false | ei_m \in EI,$$

where $selectEI_k$ is a predicate which indicates k-th event input activation. The first sub-rule states that once the current OSM state is s_1 and the input signal is selected, then we need to save timestamp values to buffer *IB* and reset $ei_k \in EI$. Second sub-rule resets all non-selected input signals.

Transition rules for the function of triggering of event outputs (Rule Set 8 [7]) is redefined by the following rules:

$$p_{EO}^{B,1}[k] : putoutEO_k \rightarrow Z_V(eo_k) = true \wedge Z_{TB}(eo_k) = Z_{TB}(IB)$$
$$\wedge Z_{TL}(eo_k) = \tau | eo_k \in EO,$$

where $putoutEO_k$ is a predicate which indicates k-th event output activation and τ represents the value of current systems time. The rule states that once k-th event output is activated, its value is set to *true* and birth timestamp value is copied from buffer *IB* and last execution timestamp value is taken from τ.

For composite function blocks with event timestamps, the formal model described in [8] is used.

4.2 Timers Models

The concept of time, timers' formal model and the system of time schedulers were described in [3][9]. These models need to be adjusted to operate properly in the systems with timestamped events. IEC 61499 defines two types of timers: E_DELAY and E_CYCLE.

At first, we will consider E_DELAY function block. When creating an EO event, E_DELAY must record current system time in the TL timestamp. The EO event is a continuation of the event chain after the START event, therefore the event chain creation time must be copied from the START event TB timestamp. Therefore, the E_DELAY block needs additional buffer TB to store the timestamp. Semantical rules from [9] are changed as follows (the rules are listed in descending order of priority):

$$p_s^{D,1} : (\alpha \wedge D = 0) \rightarrow (Z_V(EO) = true \wedge Z_{TL}(EO) = \tau \wedge Z_{TB}(EO) = TB \wedge D = -1)$$
$$p_s^{D,2} : (\alpha \wedge Z_V(STOP)) \rightarrow D = -1$$
$$p_s^{D,3} : (\alpha \wedge Z_V(START) \wedge D = -1) \rightarrow (D = DT \wedge TB = Z_{TB}(START))$$

Considering E_CYCLE function block, each EO event creates a new event chain and at this point, both timestamps must record the current system time. Hence, E_CYCLE does not need any additional buffers for timestamps. The only difference is that it needs to record current system time in timestamps when EO event is triggered.

4.3 Modelling Event Delays

The standard assumes that there are no delays when events are transmitted between function blocks within a single resource. However, when an application is deployed, function blocks can be mapped to different resources. At the same time, the IEC 61499 standard specifies that data and event connections between function blocks distributed to different resources are redirected through the communication SIFBs and messages are transmitted through the communication network [10]. Message delays may occur while messages are being sent over the communication network. To verify the robustness of the system against event delays, it is necessary to simulate the delays of events. To this end, when generating SMV models, the selected event inputs and outputs are supplemented by the communication SIFBs models and the network communication model represented by the FIFO event queue. Thus, no changes to the original system are required to simulate delays and resulting model conforms to formal specification. The system's robustness against delays is checked by adding non-determinism on selection of the event delay time in the queue.

4.4 Automatic Models Generation

The presence of the formal specification makes it possible to generate models for verification automatically. For generation of models of programs with timestamps, FB2SMV tool was modified. We added support for generation of SMV models based on the formal model defined previously. These changes have been implemented in the FB2SMV project.[2]

5 Case Study

As a case study we continue an elevator example from [2]. The system model in IEC 61499 consists of three components: a controller, a plant simulation model and sensors (Fig. 2 (b)). The experimental model differs from the previous one in that the sensors are moved into a separate component. Previously, they were incorporated into the plant model. However, the plant simulation model is usually modelled separately, which

[2] https://github.com/dmitrydrozdov/fb2smv/tree/timestamps

requires separation of components. In addition, the controller configuration has been modified to support an arbitrary number of floors. Sensors and the controller are located on different devices and transmit signals wirelessly. Random delays may occur during wireless transmission. Due to these delays, the controller can stop the motors when the elevator cabin has already passed the floor, which can lead to the opening of the doors when the elevator is between the floors (Fig. 2 (a)).

The controller must take into account the delay time of the event in order to be able to withstand delays. When receiving an event from the *AtFloor* variable event sensor, the controller does not immediately enter the floor EC-state (Fig. 3). The first step is to

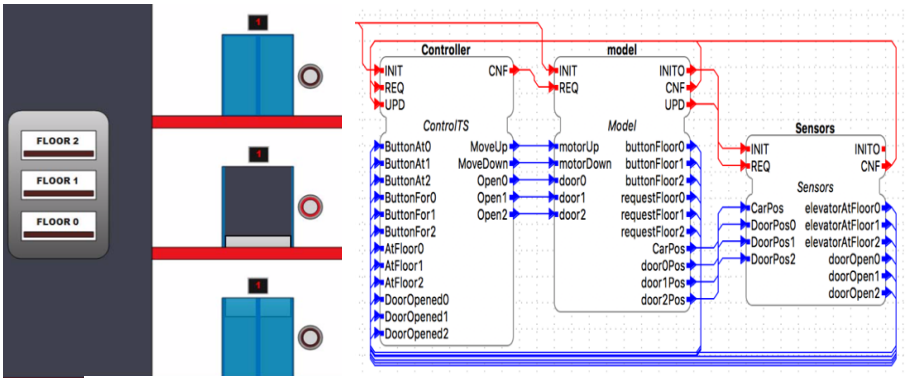

Fig. 2. (a) Elevator error; (b) Closed-loop elevator model

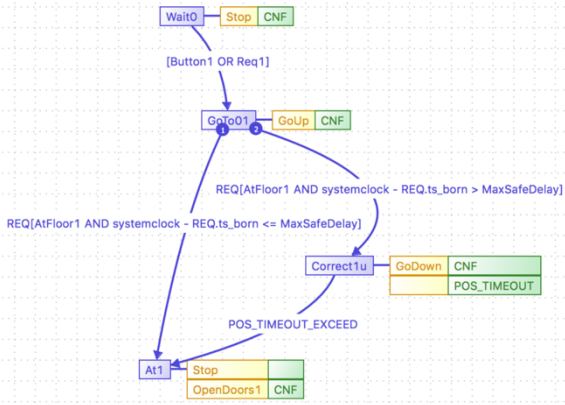

Fig. 3. ECC for correction of the elevator cabin position

check that the event delay does not exceed the *MaxSafeDelay* threshold. If the delay is greater than the allowable one, the controller switches to the state of correction of the elevator cabin position and issues a command to move in the opposite direction for the duration equal to the delay of the event.

The SMV models of the controller and sensors are generated automatically by enhanced FB2SMV, and then combined with the SMV model of the plant, which is made by means of a temporary automaton in UPPAAL [2].

In case study [2] elevator followed only one predefined route. In our case, the floor on which the elevator is located initially, as well as the floor to which the elevator will

be called, are chosen in a non-deterministic way. This allows the system to be checked more thoroughly, however, the verification time increases.

Cyber-physical agnosticism in this case expressed as the safety property — the fact that the doors can only be opened when the elevator is on the appropriate floor. Also, we check that elevator arrives at the corresponding floor on call.

As a result, if the *MaxSafeDelay* delay threshold is set too high, the verifier detects a safety issue and provides a counter example.

For a proper comparison of proposed approach with the timestamp simulation method used in [2], we made performance measurements of both methods on the same elevator system on the same machine. Measurements were taken on a personal computer with an Intel Core i7, 2.2 GHz processor and 16 GB RAM. Before measurements were made, a variable ordering file was created with the option of NuSMV verifier '*dynamic_var_ordering -e sift_converge'*.

Table 1 shows the results of comparison of verification time and memory consumption for non-deterministically selected delay times. As can be seen from the table, the proposed approach shows better performance, consuming less memory and requiring less time for verification. The improvement is achieved due to the fact that the final SMV models are simpler than the models resulting from the simulation method.

Table 1. Verification performance comparison

	Proposed approach		Simulation	
Property	Time,*min*	Memory,*Mb*	Time,*min*	Memory,*Mb*
$G(DoorOpen0 \rightarrow AtFloor0)$	31.5	191	871	704
$G(DoorOpen1 \rightarrow AtFloor1)$	31.2	201	866	703
$G(DoorOpen2 \rightarrow AtFloor2)$	31.3	192	879	704
$G(Button0 \rightarrow FAtFloor0)$	41.3	222	1280	739
$G(Button1 \rightarrow FAtFloor1)$	35.1	208	1246	734
$G(Button2 \rightarrow FAtFloor2)$	41.7	213	1293	744

5 Conclusion

As the result of this work, special syntax and formal models for event timestamps were defined. The FB2SMV tool was modified and based on this automatic generation of models was implemented. The elevator case study has shown the usage of timestamps to achieve systems safety. The results of the comparison with the method of timestamps simulation have shown an improvement in verification performance.

Acknowledgments. This work was partially supported by the JetBrains Research initiative.

References

1. Vyatkin V, Pang C, Tripakis S.: Towards cyber-physical agnosticism by enhancing IEC 61499 with PTIDES model of computations. In: IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society 2015 Nov 9 (pp. 001970-001975). IEEE.
2. Drozdov D, Patil S, Dubinin V, Vyatkin V.: Towards formal verification for cyber-physically agnostic software: A case study. In: IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society 2017 Oct 29 (pp. 5509-5514). IEEE.
3. Drozdov D, Patil S, Vyatkin V.: Formal modelling of distributed automation CPS with CP-Agnostic software. In: International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing 2016 Oct 6 (pp. 35-46). Springer, Cham.
4. Drozdov D., FB2SMV Tool, https://github.com/dmitrydrozdov/fb2smv.
5. Dai W, Pang C, Vyatkin V, Christensen JH, Guan X. Discrete-Event-Based Deterministic Execution Semantics With Timestamps for Industrial Cyber-Physical Systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2017
6. International Standard IEC 61131-3: Programmable controllers – Part 3: Programming languages / International Electrotechnical Commission. — 2013.
7. Patil S, Dubinin V, Vyatkin V.: Formal Verification of IEC61499 Function Blocks with Abstract State Machines and SMV--Modelling. In: 2015 IEEE Trustcom/BigDataSE/ISPA 2015 Aug 20 (Vol. 3, pp. 313-320). IEEE.
8. Drozdov D., Dubinin V., Vyatkin V.: Formal Semantics of IEC 61499 Functional Blocks with Temporary Tags. In: University Proceedings. Volga Region. Engineering Sciences, 2019, no. 1 (49) (in Russian).
9. Drozdov D, Patil S, Dubinin V, Vyatkin V.: Formal verification of cyber-physical automation systems modelled with timed block diagrams. In: 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE) 2016 Jun 8 (pp. 316-321).
10. International Standard IEC 61499. Function blocks for industrial-process measurement and control systems. Part 1: Architecture / International Electrotechnical Commission. — 2005.
11. Hanisch, H.M., Hirsch, M., Missal, D., Preuße, S. and Gerber, C.: One decade of IEC 61499 modeling and verification-results and open issues. In: Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing, 2009, January.
12. Patil, S., Dubinin, V., Vyatkin, V.: Formal modelling and verification of IEC 61499 function blocks with abstract state machines and smv–execution semantics. In: International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, pp. 300–315. Springer (2015)
13. Stanica, M., Guéguen, H.: Using timed automata for the verification of IEC 61499 applications. In: Discrete Event Systems 2004 (WODES'04): A Proceedings Volume from the 7th IFAC Workshop, Reims, France, 22–24 Sept 2004, p. 375. Elsevier (2005)