Promela Formal Modelling and Verification of IEC 61499 Systems with comparison to SMV

1st Viktor Shatrov
Computer Technologies Department
ITMO University
Saint Petersburg, Russia
vvshatrov@yandex.ru

2nd Valeriy Vyatkin
Department of Electrical Engineering and Automation
Aalto University
Helsinki, Finland
vyatkin@ieee.org

***Abstract*—This paper presents a method of formal modelling of IEC 61499 systems of Function Blocks with Promela[1]. The existing method of formal verification of IEC 61499 using SMV (Symbolic Model Verifier) is compared with a new approach of verification using SPIN[2] which is an explicit-state model-checker. The performance of both approaches is studied using a set of deterministic systems of multiple computational units as an example and a more complex non-deterministic elevator model.**

***Index Terms*—IEC 61499, formal verification, model-checking, function blocks, SPIN, SMV, Promela**

I. Introduction

To design and develop industrial cyber-physical systems (CPS), there exists the IEC 61499 standard [1], which defines the operation of distributed systems made of *Function Blocks (FB)*. The standard allows the development of decentralized control systems having the architecture of the Internet of Things.

To ensure the reliability of such systems, formal verification methods are used to detect errors in the system. One of the effective methods of formal verification is the method of model-checking [2]. Model-checking performs verification by checking the fulfillment of the specification property in all possible states of the system. If a property is violated, the verifier provides a counter-example which is a sequence of system states that leads to a state, where the checked property is violated. To apply the method of model-checking it is necessary to make a formal model of the system being checked.

A. Related work

Various approaches have been used for formal modelling of the IEC 61499 function blocks, an overview of early works could be found in [3]. The early works on modelling of IEC 61499 adopted an approach of verification of structural properties. An example of such an approach is modelling with *timed automata* [4].

The timed automata approach allowed to verify structural properties of the system such as the presence of never-ending event loops. However it has not modelled function blocks entirely, e.g. modelling of data variables was omitted and execution of algorithms were replaced with time delay. Such an approach can not verify system properties apart from structural ones.

In contrast, *Petri nets* and *Net-Condition Event Systems (NCES)* [5], [6] approach modelled FBs in their entirety. The advantages of NCES are: possibility to simulate synchronous and asynchronous systems; availability of means for hierarchical structuring and modularity; simplicity and large computational power. NCES networks are equal to Turing machines, which makes them a universal tool for modelling FB systems of any degree of complexity. However, it requires the development of its own model-checker which is a quite complex task and requires a lot of resources.

The development of a formal meta-model of the IEC 61499 systems based on *Abstract State Machines (ASM) [7], [8]* allowed an automatic generation of formal models in the input language of modern model-checker NuSMV [9].

B. SMV Modelling Issues

However, verification of FBs by SMV imposes several issues:

- The IEC 61499 standard allows deployment of system components on different devices which can not be fully modelled due to asynchronous work of devices. SMV is intended for verification of synchronous systems and support of asynchronous processes is deprecated in modern SMV model-checkers [10].
- In many studies, verification performance is observed to deteriorate exponentially as the number of components in the system increases.
- SMV is admittedly more suitable for verification of open-loop systems than closed-loop systems [11].

In view of the issues presented, a goal was set to model IEC 61499 using Promela and compare it with SMV modelling.

The rest of the paper is structured as follows. Section II presents a method of modelling function blocks in Promela based on the ASM-based formal meta-model. Section III describes IEC 61499 systems which were used for studying the performance of SPIN and SMV verification. Section IV presents the results of the measurements. Finally, this paper is concluded in Section V.

[1] Input language for SPIN http://spinroot.com/spin/Man/promela.html

[2] http://spinroot.com/spin

II. Modelling of IEC 61499 with Promela

The formal meta-model of FBs opens possibilities for the construction of formal models for different model-checkers. We will be using the same meta-model of function blocks based on ASM which was used for SMV models [7]. In this way, Promela models will have the same level of detail and same execution semantics of the system as SMV models and their comparison will be accurate.

The functionality of a basic function block (BFB) in IEC 61499 is provided by means of algorithms, which process input and internal data and generate output data. The block interface consists of the head and body, where the head is connected to the event flow, and the body - to the data flow. The algorithms included in the block are programmed in terms of IEC 61131 [12]. To control the execution of its algorithms BFB utilizes *Execution Control Chart (ECC)* which is a graphical or textual representation of the causal relationships among events at the *event inputs* and *event outputs* of a function block and the execution of the function block's algorithms, using *execution control states*, *execution control transitions*, and *execution control actions* [1]. The operations of ECC are specified in the ECC Operation State Machine (OSM), shown in Figure 1. Figures 2 and 3 show examples of basic function block interface and execution control chart. This example describes the Arithmetic Logic Unit (ALU) which can add or subtract two numbers. We will use this example to show how Promela models are made.

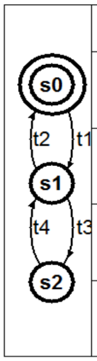
	State	Operations	Transition	Conditions
	s0	Sample inputs	t1	Input event occurs
	s1	Evaluate transitions	t2	No transitions clears
	s2	Perform actions	t3	A transition clears
			t4	Actions completed

Fig. 1. ECC Operation State Machine [1].

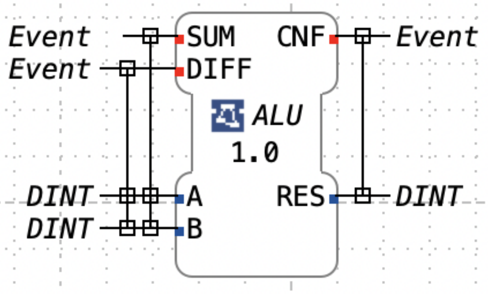


Fig. 2. Example of Function Block interface.

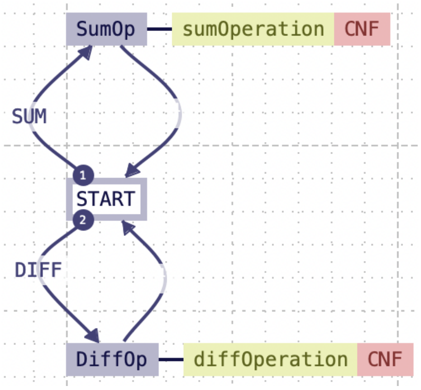


Fig. 3. ECC of ALU Function Block.

Promela (Process or Protocol Meta Language) is a verification modelling language with C-like syntax that allows for the dynamic creation of concurrent *processes*. Processes are global objects, communication between them is performed by the means of *message channels*. Message channels can be defined to be synchronous (i.e., *rendezvous*), or asynchronous (i.e., *buffered*). Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, channels and global variables define the environment in which the processes run [13].

In Promela, we model function block types as separate *proctype's* and function block instances are modelled as processes. The transfer of events and data between blocks is performed by means of message channels. Every event input and output, data input and output is modelled as a separate channel.

```
chan ALU1_EI_SUM = [1] of {bit};
...
chan ALU1_VO_RES = [1] of {int};


proctype ALU(chan EI_SUM, EI_DIFF, EO_CNF,
                   VI_A, VI_B, VO_RES,
                   alpha, beta)  {
    ...
}
```

Although SPIN is designed to model asynchronous processes, it can also model the synchronous operation of individual components. For this purpose, we will utilize *rendezvous ports*. Thus we can make a model in which the work of various devices will be asynchronous, while the work of the blocks within one device will be synchronous. According to the ASM model, α and β signals are used to indicate the start and finish of function block execution. α signal is emitted by the dispatcher in composite function block while β signal is emitted by basic function block when its execution is finished.

```
chan ALU1_alpha = [0] of {bit};
chan ALU1_beta = [0] of {bit};
```

A. Basic Function Block

The semantic part of formal ASM description consists of a set of variables at execution time VRT_B and set of functions of elementary transitions T_B. The full formal ASM description is described in [7], here we provide a description of ASM model parts and the way of their modelling in Promela.

$VRT_B = (VIB, VOB, Q, S, NA, NI, \alpha, \beta)$
$VIB = \{vib_1, vib_2, \ldots, vib_{N_{VI}}\}$ – is a set of external buffers linked to the input variables.
$VOB = \{vob_1, vob_2, \ldots, vob_{N_{VO}}\}$ – is a set of external buffers linked to the output variables.
In Promela VIB and VOB are modelled by buffered channels initialized with a unit capacity.
Q – is a variable representing the current ECC state.
In Promela to keep track of the ECC state, we define special *mtype* for each block type.

```
mtype:ALU_ECC={START_ecc, SumOp_ecc,...};
...
mtype:ALU_ECC Q = START_ecc;
```

S – is a variable representing the current state of the Operation State Machine (OSM), $Dom(S) = \{s0, s1, s2\}$
NA – is the pointer to current ECC action.
NI – is the pointer to the current step of running algorithm

Promela allows natural execution of sequential algorithms and has its own implicit command pointer, therefore there is no need for special variables for NA and NI tracking the current step of the execution. State of the OSM S is modelled with labels and transition between OSM states performed by *goto* statements.

As mentioned before α and β are modelled by *rendezvous ports*. Set of functions of elementary transitions T_B is defined by 10 Rule Sets (RS) [7]:

RS1: Once the input signal is selected, we need to reset it.
RS2: Sample all the associated input data when an event fires.
RS3: OSM transition rules.
RS4: ECC transition rules.
RS5: EC-action NA counter rules.
RS6: EC-algorithm NI counter rules.
RS7: Change of output variables values.
RS8: Triggering of output events.
RS9: Changing output data buffers.
RS10: Triggering of β termination signal.
The ASM meta-model also defines input signal selection condition (ISSC) $selectEI_k$.

Here we present the structure of the Basic Function Block model in Promela with comments indicating implementation of each Rule Set (RS) in the model.

Listing 1. Basic Function Block model structure

```
s0_osm:
  end: // s0 is valid end state
  alpha?true; // blocking communication
  ExistsInputEvent
    = nempty(EI_SUM) || nempty(EI_DIFF);
  // ISSC:
  bit selectEI_SUM, selectEI_DIFF;
  selectEI_SUM=nempty(EI_SUM)&&Q==START_ecc;
  selectEI_DIFF= !selectEI_SUM
    && nempty(EI_DIFF) && Q==START_ecc;

  if
  :: atomic {selectEI_SUM ->
        EI_SUM?true; //RS1: reset event input
        VI_A?A; //RS2: sample input data
        VI_B?B;
  }
  :: atomic { selectEI_DIFF ->
     ...
  }
  :: (!ExistsInputEvent) -> goto done;
  :: else -> skip;
  fi
  // RS1: reset all other inputs
  do
  ::atomic{nempty(EI_SUM) -> EI_SUM?true}
  ::atomic{nempty(EI_DIFF) -> EI_DIFF?true}
  ::else -> break;
  od
  goto s1_osm; // RS3

s1_osm:
  bit trans_START_SumOp, ...;
  trans_START_SumOp = nempty(EI_SUM);
  trans_START_DiffOp = nempty(EI_DIFF);
  trans_DiffOp_START = 1;
  trans_SumOp_START = 1;

  ExistsEnabledECTran = (...);
  atomic {
    if // RS4: ECC transition.
    ::(Q == START_ecc && trans_START_SumOp
       && selectEI_SUM) -> Q = SumOp_ecc;
    ::(Q == SumOp_ecc && trans_DiffOp_START)
      -> Q = START_ecc;
    ...
    ::!ExistsEnabledECTran -> goto done;//RS3
    :: else -> skip;
    fi;
    selectEI_SUM = 0;
    selectEI_DIFF = 0;
  }
  goto s2_osm; //RS3

s2_osm: //RS6
  atomic {
    if
    :: (Q == START_ecc) -> skip;
    :: (Q == SumOp_ecc) ->
      // action 1
      RES = A + B; //RS7
      // action 2 RS5
      //emit event
      VO_RES!RES; //RS9
      EO_CNF!true; //RS8
    :: (Q == DiffOp_ecc) ->
      ...
    fi
  }
  goto s1_osm; // RS3

done: // RS10
  beta!true;
  goto s0_osm;
```

B. Composite Function Block

An ASM model for Composite Function Block is described in [14]. The model proposes a concept of *scheduler* for invoking component function blocks inside a composite function block.

SMV models use shared variables to transfer messages between modules. One output message can be received by several function blocks. Channels in Promela cannot be used in the same way as shared variables because when one process receives a message, the message is withdrawn from the channel and will no longer be received by other processes. Therefore, the composite function block first puts the outgoing messages of the basic function blocks in the buffer and then copies them for all recipients.

Listing 2. Composite Function block structure

```
dispatch:
  if
  :: atomic { dispatch_state == ALU1_turn ->
     ALU1_alpha!true;
     ALU1_beta?true;
     dispatch_state = ALU2_turn;
  }
  :: atomic { dispatch_state == ALU2_turn ->
         ...
  }
  fi
  goto read_component_event_outputs;

read_component_event_outputs:
  omega = empty(ALU1_EO_CNF) && empty(ALU2_...

if
  :: nempty(ALU1_EO_CNF) ->
     ALU1_EO_CNF?true;
     ALU1_VO_RES?buf_ALU1_RES;
     reset(ALU2_EI_SUM);
     ...
     ALU2_EI_SUM!true;
     ALU2_VI_A!buf_ALU1_RES;
     ALU2_VI_B!1;
  :: (omega && dispatch_state == DONE_turn) ->
     goto done;
  :: (omega && dispatch_state != DONE_turn) ->
       goto dispatch;
fi
goto read_component_event_outputs;
```

III. Verification performance study

To compare SMV and SPIN formal verification performance, we model the same systems both in Promela and SMV.

A. System of Arithmetic Logic Units

To study the dependence of verification performance on the number of function blocks in the system, we will use a set of ALU systems.

Each system consists of a set of ALUs connected in series so that the result of a block is passed to the input of the next one. Figure 4 shows an example of a system with 5 ALUs. The systems differ only in the number of ALUs and are deterministic. Thus, excluding any other factors and leaving only the number of blocks as a variable, we can clearly show the dependence of the verification performance on the number of function blocks

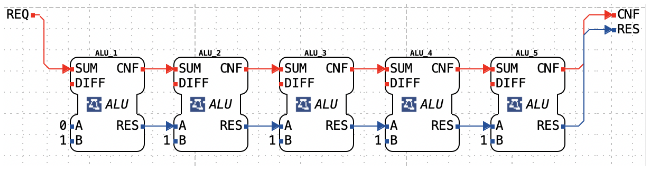


Fig. 4. System of ALUs.

B. Elevator

The system of ALUs is simple and deterministic. To study the performance of verification on complex and non-deterministic systems we perform another comparison using an elevator system (Figure 5). It is the same system that was used in [15]. The system consists of three composite function blocks: controller, cabin model, and sensors. The system has timers, which requires modeling of time and delays. Counting the basic blocks there are a total of 12 blocks in the system. The initial floor of the elevator cabin and the floor for the elevator request are selected non-deterministically.

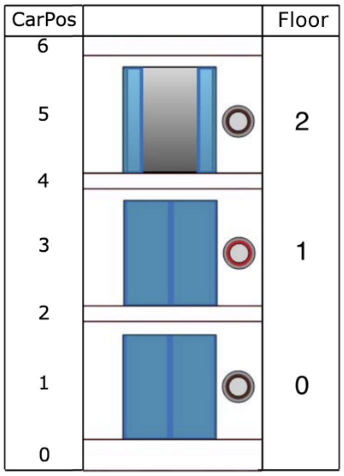


Fig. 5. Elevator system.

As in [15], event delays may occur between the sensors and the controller. If the event delay is too long, the elevator may pass a floor and needs to return and correct the position of the cabin. This considerably increases the complexity of the system behavior. We perform verification in two variants: with a constant fixed event delay, and with a non-deterministic choice of the delay for each event.

System liveness and safety properties are checked, which are written as LTL-properties.

Liveness properties are expressed as the requirement that the elevator arrives on request at the right floor.

$G(Button\_press = N \rightarrow FAtFloorN)$

Safety properties check that cabin doors are opened only when the elevator stays on the floor.

$G(DoorOpenN \to AtFloorN)$

Full SMV and Promela models and the source files of IEC 61499 systems are available in the public repository[3].

IV. Results and Discussion

Measurements were performed on *MacBook Pro (16-inch, 2019), 2,3 GHz 8-Core Intel Core i9, 32GB RAM*.

Verification time and memory consumption for systems of ALUs are presented in Figures 6 and 7.

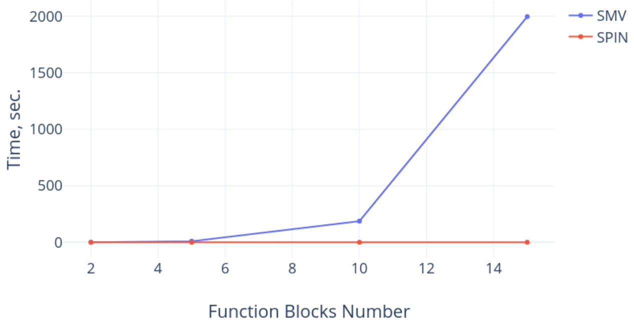


Fig. 6. Verification Time

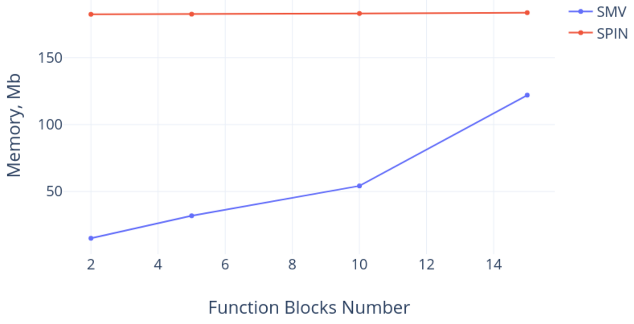


Fig. 7. Required Memory

The graph in Figure 6 shows the exponential degradation of the SMV models performance as the number of function blocks in the system increases. The time varies from one second for verification of two blocks, to half an hour for verification of a system of fifteen blocks. At the same time, the verification of the same systems in SPIN varies from 0.35 to 0.85 seconds and show a linear increase in time.

Regarding memory consumption in Figure 7, it can be seen that SPIN initially requires an order of magnitude more memory. However, as the number of function blocks in the system increases, the gap between SPIN and SMV narrows and changes places.

The results of verification for elevator system with fixed and non-deterministic event delays are presented in tables I and II respectively.

TABLE I
Elevator results

Property	Time, sec.		Memory, Mb.	
	SMV	***SPIN***	***SMV***	***SPIN***
G (DoorOpen0 → AtFloor0)	1054	2,4	108	881
G (DoorOpen1 → AtFloor1)	1182	2,6	112	881
G (DoorOpen2 → AtFloor2)	1160	2,5	105	881
G (Button_press = 0 → F AtFloor0)	1388	9,5	120	923
G (Button_press = 1 → F AtFloor1)	1700	8,2	129	914
G (Button_press = 2 → F AtFloor2)	1663	9,6	125	923

TABLE II
Elevator with non-deterministic event delays

Property	Time, min.		Memory, Mb.	
	SMV	***SPIN***	***SMV***	***SPIN***
G (DoorOpen0 → AtFloor0)	146	0,3	336	2593
G (DoorOpen1 → AtFloor1)	221	0,3	338	2593
G (DoorOpen2 → AtFloor2)	141	0,3	289	2593
G (Button_press = 0 → F AtFloor0)	510	1	432	2855
G (Button_press = 1 → F AtFloor1)	314	0,9	369	2790
G (Button_press = 2 → F AtFloor2)	156	1	302	2855

The significant difference in performance between SMV and SPIN, as well as the slowdown of SMV as the number of function blocks increases, can be explained by the following features of the verifiers: SMV is a symbolic model-checker, while SPIN is an explicit-state model-checker. Symbolic verification is less prone to state-space explosion problems, as it is with explicit-state model-checkers. Symbolic verification uses binary decision diagrams (BDD) for this purpose. The verification performance, in this case, depends to a greater extent not on the state space size, but on the size of resulting BDD formulas, which in turn depend on the total number of variables in the initial model. With full-scale modelling of function blocks, the resulting model is quite detailed and has a large number of variables. Adding even the simplest function block to the system adds dozens of variables to the model, while the system's state space may increase insignificantly. At the same time, a closed-loop modelling approach is applied, in which a model of the control object is supplemented to the controller. Modelling the controller in conjunction with the control object model in a closed-loop enables the simulation of the real behavior of the system and reduces the state space of the system but at the same time the number of variables increases due to the introduction of the control object model. The peculiarities of SPIN operation allow it to work better in all the above circumstances than SMV.

V. Conclusion

With regard to the IEC 61499 formal verification task, we can distinguish the following advantages and disadvantages of verification with SPIN and SMV.

SMV advantages:

- low memory consumption for small systems;
- tooling for automatic model generation.

SMV disadvantages:

- performance;

[3] https://github.com/vi34/conf/tree/main/INDIN2021

- only synchronous modelling.

SPIN advantages:

- short verification time;
- multiple device modelling;
- dispatcher models beyond the cyclic model;
- verification of systems in closed-loop.

SPIN disadvantages:

- absence of tools for an automatic model generation;
- limitation of the maximum system size of 255 function blocks.

In conclusion, this paper presented a way of modeling IEC 6499 with Promela, compared verification performance with SMV model checker, and showed that SPIN is better suited for the task of formal verification of IEC 61499 function blocks in a closed-loop. Further direction of work is the development of a toolkit for automatic generation of Promela models from XML descriptions of function blocks.

Acknowledgment

This work was sponsored, in part, by the JetBrains Research initiative and by the H2020 project 1-SWARM co-funded by the European Commission (grant agreement: 871743).

References

[1] IEC, *International Standard IEC 61499. Function blocks for industrial-process measurement and control systems. Part 1: Architecture*, International Electrotechnical Commission, 2005.

[2] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[3] H. M. Hanisch, M. Hirsch, D. Missal, S. Preuße, and C. Gerber, "One decade of IEC 61499 modeling and verification - Results and open issues," vol. 13. IFAC Secretariat, 1 2009, pp. 211–216.

[4] M. Stanica and H. Guéguen, "Using timed automata for the verification of IEC 61499 applications," *IFAC Proceedings Volumes*, vol. 37, no. 18, pp. 375–380, 2004.

[5] V. Vyatkin and H.-M. Hanisch, "Verification of distributed control systems in intelligent manufacturing," *Journal of Intelligent Manufacturing*, vol. 14, no. 1, pp. 123–136, 2003.

[6] V. Vyatkin, "Modelling and verification of discrete control systems," 2007.

[7] S. Patil, V. Dubinin, and V. Vyatkin, "Formal Verification of IEC61499 Function Blocks with Abstract State Machines and SMV–Modelling," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 3. IEEE, 2015, pp. 313–320.

[8] ——, "Formal modelling and verification of IEC61499 function blocks with abstract state machines and SMV-execution semantics," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer, 2015, pp. 300–315.

[9] D. Drozdov. FB2SMV: IEC 61499 Function blocks XML code to SMV converter. [Online]. Available: https://github.com/dmitrydrozdov/fb2smv

[10] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, "Nusmv 2.4 user manual," *CMU and ITC-irst*, 2005.

[11] I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Explicit-state and symbolic model checking of nuclear I&C systems: A comparison," in *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2017, pp. 5439–5446.

[12] IEC, *61131-3: Programmable controllers–part 3: Programming languages*, International Standard, Second Edition, International Electrotechnical Commission, Geneva, 2003.

[13] R. Gerth. (1997) Concise promela reference. [Online]. Available: http://spinroot.com/spin/Man/Quick.html

[14] S. Patil, V. Dubinin, C. Pang, and V. Vyatkin, "Neutralizing Semantic Ambiguities of Function Block Architecture by Modeling with ASM," in *Perspectives of System Informatics*. Springer Berlin Heidelberg, 2015, pp. 76–91.

[15] V. Shatrov and V. Vyatkin, "Formal verification of IEC 61499 enhanced with timed events," in *11th Advanced Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS 2020), 1-3 July, 2020, Costa de Caparica, Portugal*. Springer, 2020, pp. 168–178.