Formal modeling and verification of IEC 61499 function blocks on the basis of transition systems

Victor Dubinin
Penza State University,
Penza, Russian Federation
victor_n_dubinin@yahoo.com

Valeriy Vyatkin
Luleå University of Technology,
Luleå, Sweden, Aalto University,
Helsinki, Finland
vyatkin@ieee.org

Anatoly Shalyto
ITMO University,
St. Petersburg, Russian Federation
anatoly.shalyto@gmail.com

Abstract— **The IEC 61499 standard has become one of the key approaches to building distributed component-based control systems in industrial automation. The problem of adoption of this standard in industrial practice is often associated with incompletely defined semantics of functional blocks (FB), which are main design artifacts of the standard. In this paper we propose formal (operational) semantics of IEC 61499 FB using transition systems (by example of FB systems operating in accordance with the cyclic execution model). The proposed FB operational semantics is most convenient for the formal verification of FB systems on the basis of model checking as it describes the direct transitions between states. A limitation of this approach is the need to map hierarchical FB systems to flat models. Thetechnique of code generation used for transforming FB transition systems to SMV models is briefly discussed followed by a simple case study.**

Keywords—function block; IEC 61499; formal semantics; transition system; model checking; SMV

I. INTRODUCTION

The IEC 61499 standard has become one of the key technologies for building distributed component-based control systems in industrial automation [1,2]. The main design artefacts in the IEC 61499 are function blocks (FBs).

The state of the art in the field of IEC 61499 FB semantics is characterized by the following: 1) ambiguities in semantic definitions ; 2) the presence of a number of FB execution models; 3) the lack of a recognized formal FB semantics. As a result, these factors lead to an increase of skepticism with respect to the IEC 61499 and to a slowdown of its adoption in industry.

The semantical complexity of FBs is determined by the following: 1) the presence of nontrivial FB execution models; 2) the presence of data processing algorithms in basic FBs; 3) event driven execution of FBs; 4) hierarchy and modularity of FB systems; 5) the presence of typing and instantiation for the most design artifacts (FBs, sub-applications, resources, devices); 6) the presence of both software and hardware modules properties in case of FBs; 7) FB language is a visual language programming language. Such visual component-based, event-oriented languages require special approaches to the description of their semantics.

Let us briefly consider works associated with FB formal semantics. Well-known models of FBs based on the NCES [2], finite state machines, timed automata, and Petri nets cannot be referred to FB formal semantics because of their abstraction. For example, they do not quite adequately cover all the elements of FB language. In [3] FB semantics is presented in the form of an abstract machine, but not fully (without taking into account the specific FB execution models) and sometimes informally. In [4] FB structural operational semantics is developed, but only for synchronous execution model. In [5,6] it is proposed a formal framework for FB modeling in various execution models and three FB execution models is considered. In principle, this model can be considered as a FB formal semantic model, but with some limitations: 1) Operational State Machine (OSM) machine which plays a key role in managing a basic FB execution and defined in IEC 61499 [1] is not considered; 2) the proposed model is rather abstract in a number of other points, for example, an algorithm is considered as an abstract function, without detailing the steps of its execution, that is important, for example, when FB system implementing. In [7,8] FB formal semantics based on abstract state machines (MAC) is proposed. The disadvantage of this approach is the large time and space complexity of the verification process of FB systems, designed in accordance with this model.

An attractive method for FB formal semantics definition are transition systems [9]. Historically, transition systems were the first tool used to give the formal semantics of a programming language [11]. Their big advantage is ease of mapping to Kripke structures that plays an important role in the formal verification based on model checking [10], which is supported by some industrial verifiers (e.g., nuSMV [11]).

Transition systems are most suitable to define semantics when the concept of a state is explicitly or implicitly presented in the languages. Typically, these are languages for specification of hardware, reactive, control and hybrid systems. From this point of view, it can be assumed that transition systems will be suitable for modeling FBs, since their base is an automata model [13].

The paper is structured as follows. Section II provides FB formal semantics based on a transition system. In Section III, we propose a model checking-based approach to verification of FBs represented in the form of the transition system. Section

IV shows a simple case study of FB model checking in SMV. Finally, Section V gives the conclusion and future work.

II. Formal Model of Function Blocks in the Form of a Transition System

Below a transition system is represented, whose main elements are (global) states of a FB system and (global) transitions between the states, which clearly define transition conditions and values of all variables after firing the transitions. In what follows, the syntactic FB model proposed in [8,9] will be used. It should be noted that for simplicity we consider only the FB systems functioning within the cyclic execution model that does not break generality of the approach.

As noted in [3], an unfolded FB system can be represented as a tree of FB instances. The model of the FB system is defined as follows: *MS*= (*SM_B*, *SM_C*, *VM*), where $SM_B = \{M_B^1, M_B^2, ..., M_B^n\}$ is a set of basic function block modules (BFBM); $SM_C = \{M_C^1, M_C^2, ..., M_C^m\}$ is a set of composite function block modules (CFBM); *VM* is a set of maps of module variables representing, in fact, relations of formal and actual arguments of parent/child modules.

Since FB system model includes modules of different kinds, then we define their state separately. BFBM state is defined as

$$S_B = (Z_{EI}, Z_{EO,} Z_{VI}, Z_{VO}, Z_{VOB}, Z_{VV}, Z_Q, Z_S, Z_{NA}, Z_{NI}, Z_\alpha, Z_\beta),$$

where symbols *Z* with indexes are functions of FB runtime variables values, and the indexes themselves refer to sets of corresponding variables. For example, *EI* and *EO* are sets of FB event input and output variables; *VI, VV* and *VO* are sets of FB input, internal and output variables; *VOB* is a set of data buffers; *Q* and *S* are state variables of Execution Control Chart (ECC) and Operation State Machine (OSM) [1], *NA* and *NI* are counters of *EC*-actions (in a list of EC-actions) and instructions (in an algorithm); α and β –are flags of FB execution start and finish.

BFBM state is defined as the following tuple:

$$S_C = (Z_{EI}, Z_{EO,} Z_{VI}, Z_{VOB}, Z_\alpha, Z_\beta).$$

It should be noted that values of FB output variables are not included in state S_C, since these variables are replaced by their representatives which are output buffers of component FBs.

A state of an unfolded FB system is defined as follows:

$$S = (S_B^1, S_B^2, ..., S_B^n, S_C^1, S_C^2, ..., S_C^m),$$

where S_B^i $(i = \overline{1,n})$ and S_C^j $(i = \overline{1,m})$ are states of *i*-th BFBM and *j*-th CFBM, included in the FB system, accordingly.

Transitions of a FB system are defined as follows:

$$R = T_{FB}^B \cup T_{SD}^B \cup T_{EX}^B \cup T_{CA}^B \cup T_{CS}^B \cup T_{CF}^B \cup T_{ES}^B \cup T_{TC}^C \cup T_{TO}^C \cup T_{SF}^D \cup T_{PF}^D \cup T_{PS}^D \cup T_\varnothing$$

where $T_{FB}{}^B$ is a set of transitions of type "Firing of EC-transition"; $T_{SD}{}^B$ is a set of transitions of type "Simultaneous data sampling in basic FB"; $T_{EX}{}^B$ is a set of transitions of type "Execution of algorithms step"; $T_{CA}{}^B$ is a set of transitions of type "Completion of EC-action execution"; $T_{CS}{}^B$ is a transition "Completion of EC-state execution"; $T_{CF}{}^B$ is a transition "Completion of basic FB execution"; $T_{ES}{}^B$ is a transition "Dummy launching of basic FB"; $T_{TC}{}^C$ is a set of transitions of type "Simultaneous data sampling in composite FB"; $T_{TO}{}^C$ is a transition "Transfer and issue of signals by composite FB"; $T_{SF}{}^D$ is a transition "Launching of component FB"; $T_{PF}{}^D$ is a transition "Completion of composite FB execution"; $T_{PS}{}^D$ is a transition "Reset of composite FB execution start flag"; $T_\varnothing$ is the empty transition introduced for the case when a state $s \in S$ has no followers. This ensures that the transition relation in Kripke structure will always be total.

There is the following correspondence between transitions' superscripts and modeling objects: "*B*" to a basic FB, "*C*" to a composite FB; "*D*" to a dispatcher. In what follows, to unify the presentation we will represent the transitions in the form of production rules as they have been used for FB formal model definition on the basis of Abstract State Machines (ASM) [9,10]. But in contrast to ASM rules the transition system rules can include a lot of operations executed in parallel. Symbol ";" is used for separation of operations in the right hand side of a rule. To represent a group of similar parallel operations sign Ξ is used. The general view of the rule is the following: $p_{id}^m : c \Rightarrow a_1; a_2; ...a_n$, where *c* is a rule application condition; $a_1, a_2, \dots, a_n$ are operations to change variables; *id* is an transition type identifier; *m* is a modifier describing the denotation of FB kind and the number of the rule.

It should be noted that the use of the concept of global states and transitions brings some difficulties to describe a hierarchical FB system, even if it is unfolded and brought into the one-level representation. The designation of the used variables must correspond to a global identification. This can be done, for example, by appropriate indexes. In order not to overload the formal model by over-indexing in a subsequent we will use the local designation of variables adopted from ASM-based FB model [8,9]. This is possible because changes of variables are mostly performed only in local areas and especially within FB instance. The local area of variables changes will be pre-specified.

As an example, let us consider a few transitions. There are some modifications of transitions of type "Firing *EC*-transition" depending on the following conditions: a) whether the *EC*-transition is event-triggered or not; b) whether the target *EC*-state contains *EC*-actions or not. In the case of event-triggered *EC*-transitions whose target *EC*-states have *EC*-actions, the corresponding subset of transitions from T_{FT}^B is represented by the following set of rules:

$$\{T_{FT}^{B,1}[i,k,j] : Z_S(S) = s_1 \wedge$$
$$\wedge \bigvee_{(q_i, ei_k, q_j) \in ECTran} (Z_Q(Q) = q_i \wedge EnabledECTran(q_i, ei_k, q_j, Z_{EI}, Z_V) \wedge$$
$$\wedge \bigwedge_{\substack{(q_i, y, q_m) \in ECTran, \\ (q_i, y, q_m) \prec (q_i, ei_k, q_j)}} \overline{EnabledECTran(q_i, y, q_m, Z_{EI}, Z_V)}) \Rightarrow Z_Q(Q) \leftarrow q_j;$$
$$Z_S(S) \leftarrow s_2; \mathop{\Xi}_{ei_j \in EI} Z_{EI}(ei_j) \leftarrow false; Z_{NA}(NA) \leftarrow 1; Z_{NI}(NI) \leftarrow 1 |$$
$$| (q_j, ei_k, q_j) \in ECTran, q_j \in Q^A, ei_k \in EI\},$$

where $Q^A \subseteq DomQ$ is a set of *EC*-states having attached *EC*-actions; *EnabledECTran* is a predicate defining EC-transition condition. The above transition includes the following simultaneous operations: 1) changing *EC*-state q_i to q_j; 2) changing *OSM*-state s_1 to s_2; 3) setting the *EC*-action counter to "1"; 4) setting the instruction counter *NI* to "1"; 5) resetting all event input variables.

Transitions of type "Transfer and issue of signals by composite FB" can be represented by the following set of generalized rules:

$$\{T_{TO}^{C,1}[x,k]: Z_{EO^x}(eo_k^x) \Rightarrow Z_{EO^x}(eo_k^x) \leftarrow false;$$

$$\underset{(eo_k^x, ei_m^y) \in EvConn}{\Xi} Z_{EI^y}(ei_m^y) \leftarrow true; \quad \underset{(eo_k^x, eo_j) \in EvConn}{\Xi} (Z_{EO}(eo_j) \leftarrow true;$$

$$\underset{(eo_j, vob_m) \in OW}{\Xi} Z_{VOB}(vob_m) \leftarrow Z_{VO}(repr_{VO}(vob_m)))\,|$$

$$|\, x \in \overline{1, N_{FB}}, eo_k^x \in EO^x\}.$$

Transition of this type processes one of the active sources of signals, which is here the event output of component FB. When activating this rule, the following simultaneous operations are fulfilled: 1) transferring the signals to the target event inputs of component FBs and the target event outputs of FB shell; 2) issuing data via information connections which are *WITH*-associated with the active event outputs; 3) resetting the source of the signal. From the above rule one can derive more specific rules depending on the topology of connections. In particular, the rule can have no relation to data sampling.

III. Verification of FB Systems on the Basis of Model Checking

It is well-known that SMV-based verifiers are some of the best for verification on the basis of Model checking [11]. Transforming a transition system-based FB model to SMV does not require overcoming any semantic difficulties. Each transition of the formal FB model corresponds to one transition defined in SMV by TRANS statement. However, due to using the concept of global states and lack of modularity there are some problems of a "constructive" nature. As noted above, this approach requires unfolding the system, and as a consequence, the use of global names. At that, the structured name space consisting of local name spaces for modules is transformed to a general (linear) name space of the system as whole. Various methods of forming global names can be used. One of the quite acceptable options is to use hierarchical names, reflecting the path from the root to the desired FB instance in FB hierarchy tree. An example of a variables global name is *ei*1_*fa*2_*fb*7_*fc*3. As it can be seen, a global FB instances name is used as a suffix of the of *ei*1 variable.

The section of transitions description consists of a single statement TRANS which contains a description of all transitions of a system. This description actually is in the form of a formula containing conjuncts, separated by the disjunction sign "|". Each conjunct describes a transition. All terms of a conjunct is linked by the conjunction sign "&". Functionally, each conjunct is divided into two parts: a conditional part ("head"), which defines a transition's enabling condition, and an executive part ("body"), containing operations on variables, which will be performed simultaneously at firing the transition. Logically, for the convenience we will divide the transition description on into three parts: the above mentioned and preservation part, additionally.

In the section of predefined conditions it is advisable to define reusable conditions of a FB model using DEFINE-statements. In general, they can be divided into two classes: 1) the conditions that determine the logic of the FB model functioning; 2) conditions that preserve values of groups of variables. The first class includes, for example, the following:

GuardCond are guard conditions of *EC*-transitions;

EnabledECTran are enabling conditions of *EC*-transitions;

ExistsEnabledECTran is a condition of the existence of enabled *EC*-transitions;

AbsentsEnabledECTran is a condition of the lack of enabled *EC*-transitions.

The convenience of conditions of the second class follows the fact that rules for constructing a global transition in SMV require the determination of values of all variables in the global state. Since only a very limited part of the variables are changed at transitions firing and the rest of the variables remain unchanged, it makes sense to move the corresponding immutable part in a separate section. In the section of the specification it can be determined FB system properties (invariants) to be verified using temporal logic LTL and CTL.

IV. Case Study of FB System Model Checking

As an example, let us consider a system that consists of two arithmetic-logic unit (ALU) FBs (Fig. 1)[7].

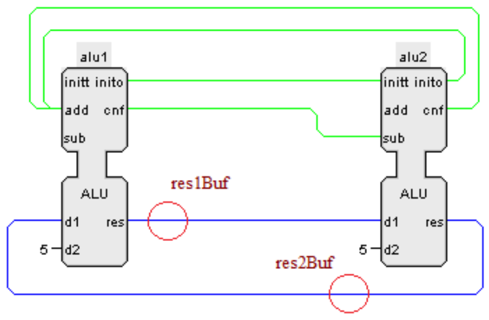

Fig. 1. "Two ALU" FB system

This example is very simple and used for illustrative purposes only. Following the firing of the *initt* input of *alu*1, the system enters an infinite sequence of computations consisting of alternating arithmetic operations addition and subtraction. In Fig. 1 *res1Buf* and *res2Buf* are data buffers (denotes as circles). They are not included in the syntactical part of FB system description, but they play an important role at functioning. The basic FB ALU in Fig. 2 is designed to perform arithmetic operations of addition and subtraction, depending on its input events.

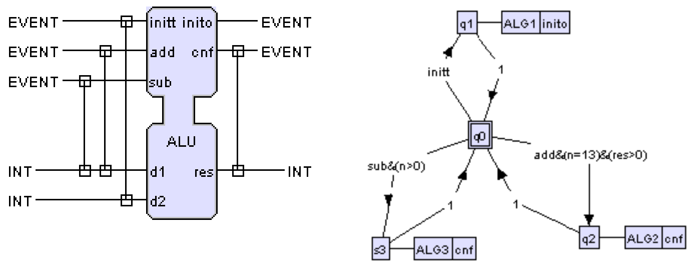

Fig. 2. The basic FB ALU: interface (left), ECC diagram (right)

Let us consider in more detail the SMV coding of the FB system shown in Fig. 1 and 2 and operating according to the cyclic execution model, using the metamodel of FBs based on the transition system. Since the FB system of two ALU is quite simple, it is not difficult to "unfold" the system and build a proper system of FB instances manually. As an example, below definitions of transitions of various types are given in SMV.

Transition "Firing EC-transition *q0→q1* in FB instance *alu1*" of $T_{FT}^{B,1}$ type of the formal model is represented by the following fragment of SMV:

S_alu1=s1 & Q_alu1=q0 & EnabledECTran_q0q1_alu1 & -- Enabling transition condition

next(Q_alu1)=q1 & next(S_alu1)=s2 & next(initt_alu1)=0 & next(add_alu1)=0 & next(sub_alu1)=0 & next(NA_alu1)=1 & next(NI_alu1)=1 & -- Transition operations

u_var_alu1 & u_EO_var_alu1 & u_disp_var_alu1 & u_obj_alu2 -- Variables values preservation

The preservation condition for event output variables of FB instance *alu1* is as follows:

DEFINE u_EO_var_alu1:= next(inito_alu1)= inito_alu1 & next(cnf_alu1)=cnf_alu1;

The preservation condition for event variables of FB instance *alu1* is:

DEFINE u_event_var_alu1:= u_EI_var_alu1 & u_EO_var_alu1;

The guard condition of *EC*-transition *q0→q2* of FB instance *alu1* is as follows:

DEFINE GuardCond_q0q2_alu1:= (n_alu1=13) & (res_alu1>0);

The enabling condition of *EC*-transition *q0→q3* of FB instance *alu2* is:

DEFINE EnabledECTran_q0q3_alu2:= select_sub_alu2 & GuardCond_q0q3_alu2;

For the analysis of the FB model, CTL temporal logic has been used in SMV. A sample request on liveliness (template *Possibility* [12]) is: *SPEC AG (EF (alu1.Q = q2))* - "Whether *EC*-state *q2* of *alu1* is repeated infinitely often, in other words, whether *EC*-state *q2* of *alu1* is alive?" (Answer: *true*). An example of another type of requests (template *Eventual Response* [12]) is: *SPEC AG (alpha2 -> AF alpha1)* - "Whether the launch of the first FB involves inevitably the launch of the second FB” (Answer: *true*).

V. CONCLUSION

A transition systems-based approach to the formal definition of operational semantics of FB systems, in particular, functioning in the cyclic execution model, was proposed. A method of coding FB systems in SMV, built on the basis of the transition system, was developed. The applicability of this method on a simple example of FB system was shown. Model checking of this FB model with proving some invariants was performed in SMV verifier. The direction of future research is software implementation of the proposed method.

ACKNOWLEDGMENT

This work was partially supported by the Russian Science Foundation (project number 15-11-10010), by the SAUNA project of SAFIR program (Finland), and by Vetenskapetsråd (Sweden) project DIACPA D.nr. 2015-04675. The work was conducted at the Penza State University (the transformation of FB transition systems to SMV models), the Aalto University and the Luleå University of Technology (the rest of results).

REFERENCES

[1] Function blocks — Part 1: Architecture, IEC Standard 61499-1, Second ed., 2012.

[2] Vyatkin, V., Hanisch, H.-M. A modeling approach for verification of IEC1499 function blocks using NetCondition/Event Systems, IEEE conference on Emerging Technologies in Factory Automation (ETFA'99), Barcelona, 1999, pp. 261–270.

[3] Dubinin, V., Vyatkin, V. On Definition of a Formal Model for IEC 61499 Function Blocks, EURASIP Journal on Embedded Systems, 2008, 10 p.

[4] Yoong, L. H., Roop, P., Vyatkin, V., Salcic, Z. A Synchronous Approach for IEC 61499 Function Block Implementation, IEEE Transactions on Computers. – 2009. – Vol. 58 (12). – P. 1599–1614.

[5] Čengic, G., Åkesson, K. On Formal Analysis of IEC 61499 Applications, Part A: Modeling, IEEE Transactions on Industrial Informatics, 2010, Vol. 6, № 2, pp. 136–144.

[6] Čengic, G., Åkesson, K. On Formal Analysis of IEC 61499 Applications, Part B: Execution Semantics, IEEE Transactions on Industrial Informatics, 2010, Vol. 6, № 2, pp. 145–154.

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

[8] Patil, S., Dubinin, V., Vyatkin, V. Formal Verification of IEC61499 Function Blocks with Abstract State Machines and SMV – Execution Semantics, 1st Int. Symp. on Dependable Software Engineering: Theories, Tools and Applications (SETTA 2015), Nanjing, China, 2015. LNCS, Vol. 9409. Springer, 2015, pp. 300-315.

[9] Arnold, A. Finite transition systems: semantics of communicating systems, Prentice Hall Int., 1994, 177 p.

[10] Clarke, E.M., Glumberg, O., Peled, D. Model Checking, The MIT press, 1999, 330 p.

[11] NuSMV – New Symbolic Model Checker. – URL: http:// nusmv.irst.itc.it

[12] Campos, J. C., Machado, J. Pattern-based Analysis of Automated Production Systems, 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, 2009, pp. 976–981.

[13] Velder, S.E., Lukin, M.A., Shalyto, A.A., Yaminov, B.R. Verification of automata-based programs, ITMO University Publishing House, St. Petersburg, 242 p. (in Russian)