A MODELING APPROACH FOR VERIFICATION OF IEC1499 FUNCTION BLOCKS USING NET CONDITION/EVENT SYSTEMS

Vyatkin, V., Hanisch, H.-M.

Institute for Automation Technology (IFAT), Dept. of Electrical Engineering
Otto-von-Guericke University of Magdeburg, PO-Box 4120, D-39016 Magdeburg, Germany
phone:+49-391-6712747; fax: +49-391-6711191, E-mail:{vyatkin,hami}@hamlet.et.uni-magdeburg.de

Abstract- *This paper presents a preliminary report on verification of discrete control applications defined by a new international standard draft IEC 1499. As a first step to verification, the structures presented in the IEC1499 are modeled with Net Condition/Event Systems, for which there exist formal methods and tools of proofing various qualitative and quantitative properties. The paper illustrates the methodology of modeling and outlines further steps towards the full-scale verification of execution control of IEC1499 applications.*

1 Introduction

The soon-to-be-revealed international standard IEC-1499 provides a new event-driven framework of software development for measurement and control systems. It follows the way paved by the previous standard on programming languages for programmable logic controllers (PLCs) IEC1131-3 [2] extending it according to the new requirements of distributed control systems.

According to the draft of IEC 1499 [1], the generic structure of an application is a function block, which can be either basic or composite. Application, which is defined as a collection of function blocks, connected by event and data flows, can be distributed over multiple devices. For example, a control application may include a function block, implementing the controller itself, as well as a block responsible for operator interface (displaying current state of the system and processing human interactions), and a block which would implement a locking controller or supervisor. All these may be supplied by some standard or user defined blocks implementing communication interface of devices where the blocks reside. For instance, the controller may be placed in one device (let us say a PLC), the operator interface in another device (PC), and the supervisor in a third device (another PLC).

The standard offers an extensive framework to describe system architecture on the logic level, which includes the notions of system, device, resource, sub-application, composite and basic function blocks. This logic architecture can be mapped onto various physical hardware structures to provide means of portability and better software reusability.

The IEC1499 defines a framework for processing events, which includes as the notion of execution control for a single function block, as well as collection of predefined blocks performing basic logic operations over events. Those can be used in definition of composite function blocks along with other user defined blocks, implementing thereby a quite complicated logic of the blocks activation.

Since the standard is especially dedicated to the software reusability issue, stressing on such means as modularity, encapsulation, and inheritance, there is an obvious need for a theoretical ground, which would enable development and investigation of applications by formal methods.

An inherent feature of present real-time systems development is their formal verification. Theoretically it is based on the works of E. Clarke [8], J. Ostroff [12, 13], Z. Manna and A. Pnueli [11], R. Alur et al. [5, 6]. In particular, verification of PLC programs (within IEC1131) was studied in [7, 17, 10]. Emerging of the new programming concept of IEC1499 requires to adjust and extend the verification techniques according to the new realities.

A typical framework for verification includes some kind of formal model for the system that is studied. Usually, finite state machines, Petri nets, Timed transition systems, etc. are used for this purpose. The properties which ensure validity or invalidity of the system are formally expressed as a predicate (static property) or an expression in Temporal Logic (dynamic property). Once the properties have been expressed in the formal language, they are checked by means of some "verification engine", which usually attempts to build the whole space of reachable states of the system and check the validity of required properties for all of them.

Although the theory is quite well developed, some additional aspects have to be taken into consideration if one deals with practical control applications. The most significant are:

1. Industrial control systems are usually designed and implemented by engineers. Their main task is to come up with a control system which actually has to perform some specific tasks such as controlling a manufacturing system, a process system, etc. Formal methods for modeling and verification should support the design and implementation process, but dealing with

the theory of formal modeling and verification cannot be the central part of daily business of an engineer. Hence, the formalisms of models and methods have to be adapted smoothly to the practical needs, and, more than this, they should be encapsulated and hidden as far as possible. That means that these methods should establish an optional support in improving the quality of a design but not a must in the design which requires detailed background knowledge of formal methods and eventually delays the design process significantly. Otherwise, an engineer would not accept the methods in his daily business.

2. A model is not the system itself but always an abstraction reflecting some properties of the system which are of interest for a particular question. It must be ensured that the model really copes with the aspects which are of interest. This means that a proper modeling formalism should be chosen to meet the requirements of the application, and that a formal or semi-formal methodology of model building should be suggested.

3. Formal methods give answers to formal questions in terms of the formal model. Questions and answers have to be mapped to properties of the real system. According to our experience it is extremely useful if the modeling formalism is as close as possible to the real system for this purpose. Each transformation causes a need for establishing a re-transformation and therefore more formal overhead and possibly loss of relevant information.

4. The modeling formalism as well as the verification engine should successfully cope with the state explosion problem, which is unavoidable for the realistic-size applications. At least efficient construction and simulation of rather large models should be possible.

Keeping these requirements in mind when trying to come up with models for function blocks according to IEC1499, a suitable model must be chosen or even developed.

The remainder of our paper deals with some basic considerations on how to establish a formal model at least for a part of function block specification, namely the execution control. For this purpose, we first consider the general structure and behavior of function blocks following IEC 1499. We then illustrate a way of modeling which seems to significantly meet the requirements stated above from our point of view. An outlook of further work that needs to be done concludes the paper.
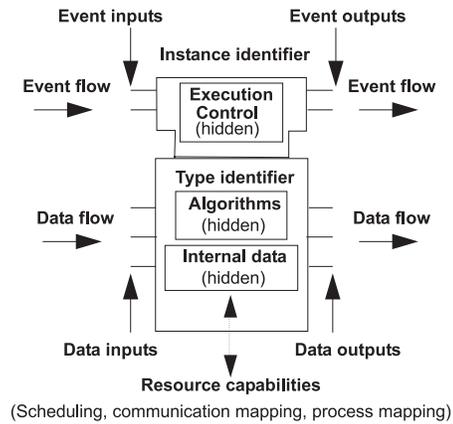

Figure 1: Characteristics of function block.

2 Structure of IEC1499 function blocks

A general diagram of a function block is shown in Figure 1. Functionality of a basic function block in IEC1499 is provided by means of algorithms, which process input and internal data and generate output data. The block consists of head and body, where 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 IEC1131.

Causal behavior of the block (i.e. sequencing of algorithms' calls) is organized in IEC1499 by means of Execution Control (EC), which is a state machine, connecting event inputs with algorithms and event outputs. Execution Control is defined by Execution Control Charts (ECCs), whose notation is simplified from the Sequential Function Charts of IEC1131-3.

The significant difference of this new approach in contrast to IEC 1131 is that a collection of interacting function blocks may be distributed over several control devices, and there is no more a sequential control function for interacting function blocks as it would be the case in IEC 1131. The execution control of function blocks is distributed as well and is established by event interconnections among several function blocks.

The draft of IEC1499 states that ECC consists of EC states, EC transitions, and EC actions, which shall be represented and interpreted as follows:

1. The ECC shall be included in an execution control block section of the function block type declaration, encapsulated by the control block construct.

2. The ECC shall contain exactly one EC initial state, represented graphically as a round or rectangular, double-outlined shape with an associated identifier. The EC initial state shall

have no associated EC actions. The ECC shall contain one or more EC states, represented graphically as round or rectangular, single-outlined shapes, each with an associated identifier.

3. The ECC can utilize (but not modify) event input (EI) variables, and utilize and/or modify event output (EO) variables. Also, the ECC can utilize but not modify Boolean variables declared in the function block type specification.

4. An EC state can have zero or more associated EC actions. The association of the EC actions with the EC state shall be expressed in graphical or textual form. The algorithm associated with an EC action, and the event to be issued on completion of the algorithm, shall be expressed in graphical or textual form.

5. An EC transition shall be represented graphically or textually as a directed link from one EC state to another (or to the same state). Each EC transition shall have an associated Boolean condition, equivalent to a Boolean expression utilizing one or more event input variables, input variables, output variables, or internal variables of the function block.

Figure 4-a illustrates the type declaration of a function block on the example of INTEGRAL-REAL function block borrowed from the draft of IEC 1499 (Table 2.2.1 and Annex H), and Figure 4-b shows its Execution Control Chart and internal algorithms, illustrating the rules stated above. Functionality of the block is quite trivial - it integrates the function given by the value of the input XIN (of type REAL) driven by the event input EX, providing the result as an output XOUT. Initial value 0 is set to XOUT by the algorithm INIT, which is called driven by the signal INIT, and then every occurrence of EX event adds to XOUT value of XIN integrated over the time interval DT.

Thanks to the simple functionality, we use this example to illustrate the rules of transformation of function blocks to the model. The same rules can obviously be applied to blocks with much more sophisticated behavior.

As far as the IEC1499 function blocks concerned, we are interested in verification of temporal and qualitative liveness and safety properties expressed in terms of timing and sequencing of input and output signals, resource sharing, etc. Especially in composite function blocks, where total execution control is defined as a net of component ECCs, verification of the behavior is far from being trivial.
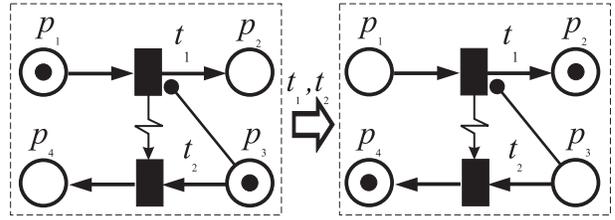

Figure 2: Example of NCES in two states: transition t_1 forces t_2 to fire by means of event arc.

3 Net Condition-Event Systems

Net Condition/Event Systems (abbr. NCES) have been developed in our prior work [14, 15]. It is a formalism for modeling of discrete event systems, based on Petri nets augmented by information and signal exchange capabilities (condition and event arcs), which provide a certain data exchange and synchronization between modules without exchange of tokens.

We illustrate briefly behavior of NCES without giving a rigorous definition of the evaluation rules on a trivial example given in Fig. 2. The net has as usual components of Petri nets: places $p_1 - p_4$, two transitions t_1, t_2 and arcs (p_1, t_1), (t_1, p_2), (p_3, t_2), (t_2, p_4) as well as specific NCES parts: event link from the transition t_1 to transition t_2, and condition link from the place p_3 to transition t_1. The event arc (t_1, t_2) forces transition t_2 to fire simultaneously with t_1 if it is enabled. In general, a transition having several incoming event arcs fires (when it is enabled) when at least one of the event inputs is true (one of the driving transitions fires). Firing of such dependent transitions occurs simultaneously, which is called as *transition step*. In the example the transition step is $TR = [t_1, t_2]$.

The condition arc (p_3, t_1) adds an additional condition on the enableness of t_1 - along with marking in p_1, marking of p_3 is also required to make t_1 enabled. So, in the state, described by the marking $P = \{p_1, p_3\}$ (state is defined by the set of places, having non-zero marking) the net exhibits the following behavior: spontaneous transition t_1 is enabled, since both p_1 and p_3 are marked. Transition t_2 is enabled but cannot fire spontaneously because of incoming event arc. However, when t_1 fires, it drives t_2 to fire too. So the next state of the net would be $P = \{p_2, p_4\}$ which is terminal, since no more enabled transitions exist.

A timing mechanism for NCES is described in [3]. It is based on Petri nets with timed arcs as presented in [16]. In the Timed NCES (TNCES) each arc leading from a place to a transition may have an attached "permeability" time interval $[DL, DR]$, where DL stands for the lower, and DR - for the upper limit, which are also called delay and perme-

ability of the arc. In this case the arc is called timed. A timed arc forces transition to fire if it becomes enabled within the permeability intervals of all of its incoming timed-arcs. If even a single arc is timed, the whole net becomes timed. In this case we assume that all non-timed arcs have the permeability interval $[0, \infty]$ and the net would have no really spontaneous transitions, which means that even a non-timed and not event-driven transition fires as soon as it becomes enabled. This provides a good framework for modeling synchronous objects, in particular programs such as Execution Control Charts.

A possible modeling framework might include not only the TNCES model of a function block (say a controller), but also model of uncontrolled plant behavior. In this case use of timing in the net brings a certain duality: on one hand it allows to make more precise model, which would provide better check of some quantitative properties, but on the other hand would hamper the check of some qualitative properties because of elimination of spontaneous transitions. However, it preserves non-determinism of the net which is especially important for many qualitative estimations.

As for the formal verification engine, the TNCES are supported by the tool SESA developed in Humboldt University of Berlin [9], and by the simulator, developed in Magdeburg University. SESA allows to perform quite sophisticated analysis of TNCES, which in particular includes:

1. Analysis of the integrity, liveness and boundness of the net (in terms of Petri nets);

2. Building the reachability graph of the net and its processing, such as finding paths which satisfy a predicate over states of the net, finding the shortest path, and the path with minimal time duration;

3. Checking the temporal logic properties of the net's evolution, expressed in the Computation Tree Logic (CTL) [8];

Using SESA it is possible, for example, to find fragments of dead code (which is never activated), answer whether the modeled system passes through a certain state or a sequence of states, measure shortest or longest time between specified events (say response of the controller). However, at the current stage of the work major progress in the application of all these abilities of SESA to verification of IEC1499 function blocks is yet to be achieved.

4   Modeling of IEC 1499 by TNCES

The hierarchy of structures provided in the standard implies the corresponding way of modeling - we going to start from the simplest basic functional blocks,

gradually extending the modeling framework to the whole applications and systems. We mostly concentrate on the execution control issues described by the Execution Control Charts and by the structure of event interconnections. Thus we pay little attention to the modeling of the internal algorithms as long as they do not strongly concern the execution logic of the block. Calls of the algorithms can be modeled by the corresponding time delays when such a timing is essential.

This agrees with the practical view on modeling - to get more or less valuable results we need to concentrate only on the important issues sacrificing the neglectable ones.

Concerning the variety of data types admitted by the standard, we divide them on the four categories: event signals, which are modeled mainly by event arcs of NCES; Boolean variables which can be modeled by marking of a place in the NCES model and by condition signals, which convey the value of variable without affecting tokens flow of the net; time parameters, which are mapped onto corresponding permeability intervals of some NCES arcs, and other numerical data which are not precisely modeled as far as it does not concern the logic of execution.

Based on the analysis of the IEC1499 draft we conclude that model of a basic function block should include the following components:

1. Event input state machine (EI-SM) implementation module (one for each event input).

2. Event input variables storage (EIVS) models (one for each event input).

3. Model of EC operation state machine (ECO-SM).

4. Module implementing ECC (ECC model), including the sub-modules implementing actions and algorithms (optional).

5. Event output variables (EOV) models (one for each event output).

According to the standard, information about events is transmitted between function blocks by means of event variables. For each event input (EI) shall be maintained an EI variable plus a storage, which exhibits the behavior defined by the state machine, given in the draft in Figure and Table 2.2.2.2-1. The state machine along with its NCES model is shown in Fig.3-a. Its transition arcs are marked both with conditions of transitions and with operations, executed upon the transitions as listed in Table 1. The operations consist of issuance of event signals "Invoke ECC" at $t1$ and "set Event Input variable" at $t3$.

In our model, we implement the behavior of event input and event input variables by two NCES presented in Figure 3-a,b.
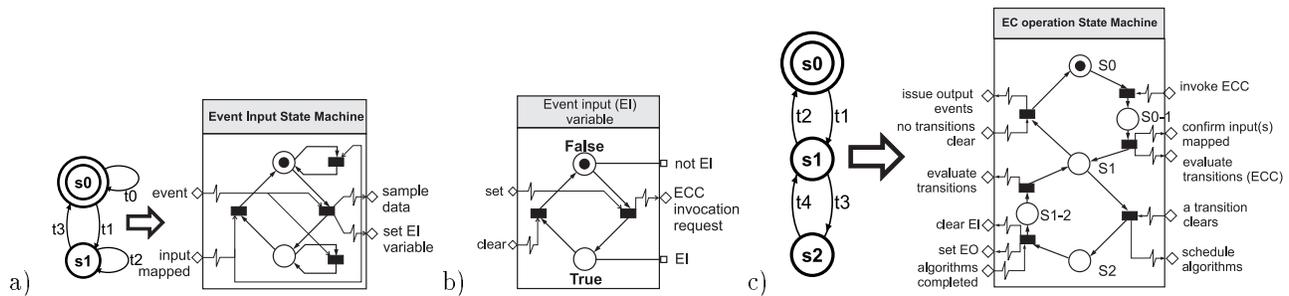
Figure 3: a)Event input state machine and its NCES model; b) Event input variable model; c)Operation of execution control state machine modeled by NCES.

Operation of the Execution Control Chart is described in the IEC1499 by means of the state machine shown in Figure and Table 2.2.2.2-2 of the draft. It is represented by the NCES model in Figure 3-c. Similar to the event input state machine, the state machine of execution control also defines a Mealey automaton with input and output symbols associated with arcs. The table of conditions and operations is presented in Table 2. It requires some comments since there are multiple operations associated with some transition arcs. Sequence of the operations and their timing seem to be important. Thus transition $t1$ is driven by the event "Invoke ECC" and has the following associated operations:

1. Set EI variables and sample associated variables;

2. Confirm input(s) mapped;

3. Evaluate transitions;

First operation sends an event signal to the model of corresponding event input variable, second - to the state machine implementing the corresponding event input, and the third sends activation signal to the NCES model of Execution Control Chart. It is essential to ensure that the latter signal is issued after the first operation has been completed, because value of event input variable might be used for evaluation of transitions in the model of ECC. To ensure this functionality we initiate the first operation in the model of event input (EI-SM). By means of event arc "Set EI" it forces transition in the Event Input Variable Storage, which, in turn invokes the ECO-SM model. Finally, the latter issues the signal "Evaluate transitions". Though basic structure of ECO-SM model is similar to the original ECO SM (places $S0,S1,S2$ correspond to the states $S0,S1,S2$), we introduce an additional "transitional" place in the ECO-SM model (marked $S0-1$), which ensures that variable setting is completed before the transition evaluation. Since the transition from $S0-1$ to $S1$ is not forced by any event, and the net is assumed to be timed, it fires as soon as $S0-1$ becomes marked. The same role

plays the place $S1-2$ - it ensures that new transition evaluation is done after the event variables used in the previously cleared transition have been reset.

For structural clearness we connect the corresponding EI-SM and EIVS modules by the event arc "Set EI" though originally this operation was initiated by the state machine ECO-SM (the following behavior is defined in the standard: first event X occurs and forces transition of EI-SM from S0 to S1, then it issues "Invoke ECC" which is input of ECO-SM. The latter must issue the "Set EI" for the event input variable X). Implementing exactly this requirement would need one ECO-SM model for each event input. Instead, we change the sequence of the transitions as follows: first event X occurs, then EI-SM issues "Set EI" event signal which forces transition in the EIVS of X, and the latter invokes ECO-SM.

Building the NCES model for execution control chart is described in the next section.

5   Model of Execution Control

The draft of IEC1499 states that "the operation of the ECC shall be functionally equivalent to the rules of evolution for Sequential Function Charts (SFCs) given in subclause 2.6.5 of IEC 1131-3".

The ECC is a simplified sequential function chart (SFC), where the only initial state is present, and each transition has exactly one source and target state. As a consequence, in ECC only one active state can be present at every time instance.

The execution control chart is transformed into an NCES model, which consists of the corresponding "modules" for each state, action, and transition, and has an additional module, named Transition Evaluation Monitor (further abbr. TEM), which insures the issuance of either of the output signals "A transition clears" or "No transitions clear", required by the EC state machine.

Transition	Condition	Operation
t0	map input	none
t1	event arrives	ECC invocation request
t2	event arrives	implementation dependent
t3	map input	set EI variable

Table 1: Conditions and actions associated with transitions of the Event Input state machine.

Transition	Condition	Operation
t1	invoke ECC	set EI variables
		confirm input mapping
		evaluate transitions
t2	no transition clears	issue events
t3	a transition clears	schedule algorithms
t4	algorithms complete	clear EI variables
		set EO variables
		evaluate transitions

Table 2: Conditions and actions associated with transitions of the ECC operation state machine.

5.1 Transitions

The standard sets the following rules concerning transitions between states of EC:

1. Each EC transition shall have an associated Boolean condition, equivalent to a Boolean expression utilizing one or more event input variables, input variables, output variables, or internal variables of the function block.

2. Evaluation of an EC transition condition is disabled until ALL the algorithms associated with its predecessor EC state have completed their execution.

3. "Evaluation of transitions" consists of evaluating the conditions at all the EC transitions following the active EC state and clearing the first EC transition (if any) for which a TRUE condition is found. "Clearing the EC transition" consists of deactivating its predecessor EC state and activating its successor EC state. The order in which the EC transitions following an active EC state are to be evaluated may be provided by software tools.

The model has event input "Evaluate transitions" and is also connected to the event input variable modules. Input signal "Evaluate transitions" is connected to every transition of ECC. The latter are transformed into NCES transitions with condition input from the Boolean expression. (Implementation of Boolean expressions in NCES was in detail illustrated in [3]). Besides, a transition has an event output, which is connected to the "Transition Clears" input of TEM. The main purpose of TEM is to detect a situation when no ECC transition clears in
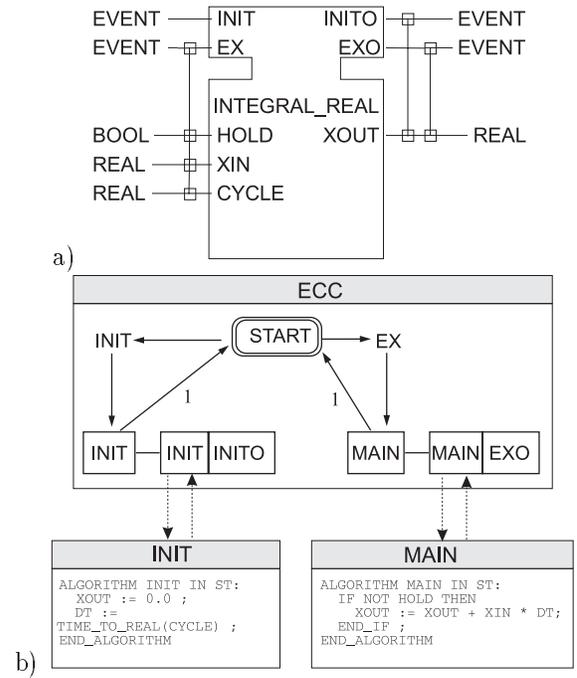

Figure 4: Type declaration of INTEGER-REAL function block (a), its execution control chart (b).

response to the forcing signal "Evaluate transitions" and to issue the corresponding signal.

Figure 5-a shows an example of TEMs binding to the NCES model of ECC. Structure of the TEM is shown in detail in Figure 5-b. Places in the TEM have the following meaning:

p_1 - ECC is ready to "Evaluate transitions"
p_2 - Transitions are being evaluated;
p_3 - No transition cleared so far;
p_4 - A transition cleared;

Transitions in the TEM have the following notation:

t_1 - TEM turns into evaluation mode;
t_2 - Transition clears, return to the initial state;
t_3 - Clearance is established;
t_4 - No clearance of transition is proved;

The input signal "Evaluate transitions" is directly connected to every transition which models an ECC transition. Every such a transition has a single output event link, all of which are merged at the transition t_3 of TEM. In our example all the transitions have the conditions expressed only in terms of condition variables.

In case if any of such transitions clears, it immediately forces t_3 to fire and moves the token from p_4 to p_3. Simultaneously, the token of p_1 moves to p_2 driven by the original "Evaluate transition" signal. Thus, in the next state places p_2 and p_3 will be marked and transition t_2 becomes enabled and is forced to fire by the arc from p_3. Therefore, it will fire, issuing event $TransitionClears$ and returning a token to the initial state.
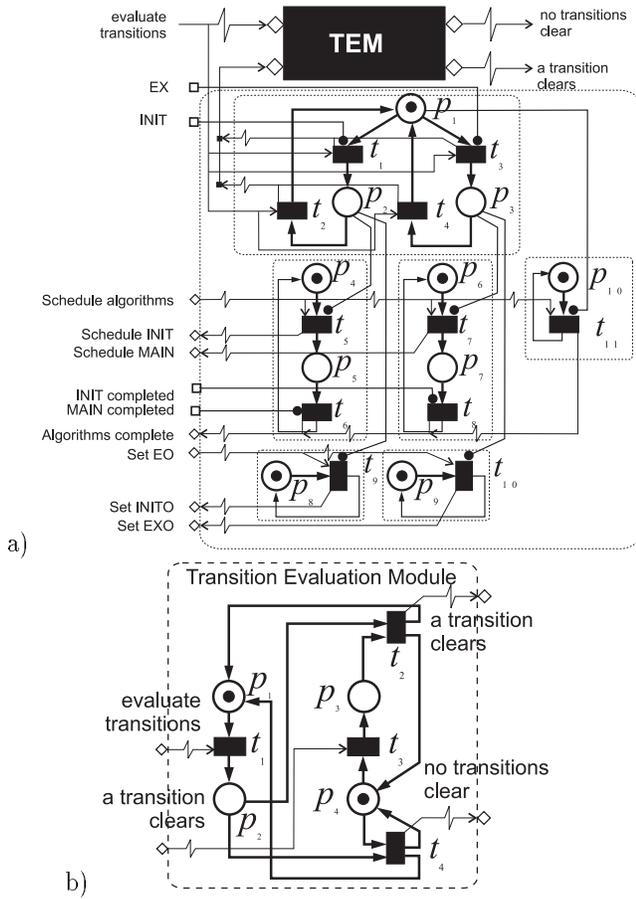
a)

b)

Figure 5: NCES model of the Execution Control Chart of the INTEGRAL-REAL function block (a) and NCES implementation of the Transition Evaluation Monitor (b)

Otherwise, if the initial signal "Evaluate Transitions" causes no subsequent firing of ECC transitions, a token moves from p_1 to p_2, but another token remains in p_4. It will cause t_4 to fire and issue the signal "No transitions clear".

5.2 States

An EC state can have zero or more associated EC actions. Each action can have an associated algorithm and an associated output event, which is issued upon completion of the algorithm.

Figure 5-a shows a NCES model for the ECC of INTEGER-REAL function block (Fig. 4-b). The module consisting of places $p_1 - p_3$ and transitions $t_1 - t_4$ corresponds to the ECC itself. Fragment $p_4 - p_5, t_5 - t_6$ models state INIT with call of INIT algorithm. Module p_8, t_9 is responsible for the issue of INITO output event. Correspondingly, module $p_6 - p_7, t_7 - t_8$ models state MAIN, and module p_9, t_{10} issues output event EXO.

In the case if no actions associated with a state (like in case of the START state), the corresponding "Algorithm completed" condition is always true.
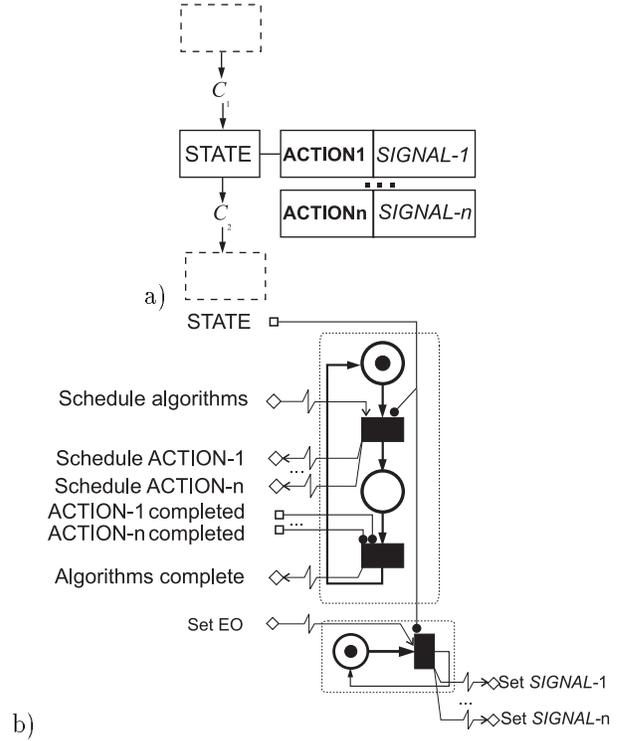

a)

b)

Figure 6: Example of ECC state with n actions (a), and its NCES implementation (b)

This is modeled by the module p_{10}, t_{11} which issues the signal "Algorithms complete" provided the signal "Schedule algorithms" is issued.

If more than one action is associated with a state, then their execution follows the evaluation rules for SFC given by IEC1131-3. Since the notion of action qualifier is not mentioned in IEC1499, we assume that each action has the null qualifier (N), which means that all the actions have to be started simultaneously.

A model of such a state is shown in Figure 6-a,b. Models of each action can be further detailed using definition of its particular algorithm. In the NCES in Figure 6-b it is assumed, that conditions "ACTION-i completed" are issued by the corresponding NCES models of algorithms. If detailed models of algorithms are not available, they can be substituted by simple "bi-stable" models having two states: "Algorithm active" and "Algorithm idle", and two transitions: one (Idle to Active) forced by the "Schedule ACTION-i", and the other (Active - Idle) being a (pseudo-) spontaneous one.

6 Example of the model

To illustrate the above-stated ideas of modeling we consider a simple example of function block INTEGRAL-REAL, the type declaration of which was given in Figure 4-a. The resulting NCES model is presented

in Figure 6.

Interconnection of the modules within the model is implemented by means of event and condition arcs. Connection of several function blocks in a composite function block can be realized in a similar way.

Let us consider a simple verification example, checking how the model responds to the event input EX. Assume the model to be in the initial state $s_0 =$

$$\{p_1, p_3, p_5, p_7, p_9, p_{14}, p_{17}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{29}, p_{31}\}$$

as shown in Figure 6. First, the event signal EX forces to fire the following set of transitions: $TR_0 = [t_6, t_{12}, t_{13}]$. The following sequence of states and transition steps unfolds then:

$s_0 \rightarrow TR_0 \rightarrow s_1$:
$\{p_1, p_4, p_5, p_8, p_{10}, p_{14}, p_{17}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{29}, p_{31}\}$
$s_1 \rightarrow TR_1 = [t_{14}, t_4, t_{20}, t_{23}] \rightarrow s_2$:
$\{p_1, p_3, p_5, p_8, p_{11}, p_{15}, p_{16}, p_{20}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{29}, p_{31}\}$
$s_2 \rightarrow TR_2 = [t_{36}, t_{15}, t_{27}] \rightarrow s_3$:
$\{p_1, p_3, p_5, p_8, p_{13}, p_{14}, p_{17}, p_{20}, p_{21}, p_{23}, p_{25}, p_{26}, p_{27}, p_{29}, p_{31}\}$
Algorithm MAIN completed:
$s_3 \rightarrow TR_3 = [t_{29}, t_{16}, t_{31}, t_{34}, t_{11}] \rightarrow s_4$:
$\{p_1, p_3, p_5, p_7, p_{12}, p_{14}, p_{17}, p_{20}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{30}, p_{31}\}$
$s_4 \rightarrow TR_4 = t_{17}, t_{20}, t_{25}, t_{19}] \rightarrow s_5$:
$\{p_1, p_3, p_5, p_8, p_{11}, p_{15}, p_{16}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{30}, p_{31}\}$
$s_5 \rightarrow TR_5 = [t_{36}, t_{15}, t_{37}, t_{16}] \rightarrow s_6$:
$\{p_1, p_3, p_5, p_8, p_{12}, p_{14}, p_{17}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{30}, p_{31}\}$
$s_6 \rightarrow TR_5 = [t_{17}, t_{20}] \rightarrow s_7$:
$\{p_1, p_3, p_5, p_8, p_{11}, p_{15}, p_{17}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{30}, p_{31}\}$
$s_7 \rightarrow TR_6 = [t_{21}, t_{18}, t_{35}] \rightarrow s_8$:
$\{p_1, p_3, p_5, p_8, p_9, p_{14}, p_{17}, p_{18}, p_{21}, p_{23}, p_{24}, p_{26}, p_{27}, p_{29}, p_{31}\}$
Output event EXO issued;

Thus $s_8 = s_0$, so the initial state is reached again, and the output signal EXO has been generated in the sequence of 9 states. This proves that the block correctly responds to the event input EX. Such a result could be easily obtained using the SESA, for example as result of checking a temporal logic property, stating that in every path of the reachability graph the initial state is eventually reached.

7 Conclusion and outlook

Thus, our preliminary study of NCES suitability for IEC1499 function blocks modeling shows, that there are no obstacles in expressing in this formalism the rules of function blocks execution control. It gives us a hope on the successful realization of the following goals:

- Express the event-function blocks of IEC1499, which implement basic event operations, by means of NCES.

- Develop formal rules of transformation of IEC1499 function blocks into NCES.

- Implement real examples of distributed control algorithms by means of IEC1499 function blocks.

- Verify their properties using transformation of function blocks into NCES and the tool SESA.

Among other goals, which we regard as the most important are formulation of validity conditions for the function blocks and their translation into Computation Tree Logic form, which is the input language for SESA.

8 Acknowledgement

References

[1] Function Blocks for Industrial Process Measurement and Control Systems *International Electrotechnical Commission, Tech.Comm. 65, Working group 6*, Committee draft

[2] International Standard IEC 1131-3, Programmable Controllers - Part 3, *Bureau Central de la Commission Electrotechnique Internationale*, 1993, Geneva, Suisse

[3] H.-M. Hanisch, J. Thieme, A. Lüder, O. Wienhold. Modeling of PLC Behavior by Means of Timed Net Condition/Event Systems *International conference on Emerging Technologies and Factory Automation (ETFA '97)*, September 1997, Los Angeles, USA

[4] H.-M. Hanisch, A. Lüder. On the mutual Dependency of Safety Controllers and Procedural Controllers *International Symposium on Industrial Electronics (ISIE '98)*, Pretoria, South Africa, July 1998, Proceedings, pp.622–627

[5] Alur, R., C. Courcoubeitis and D.L.Dill. Model checking for real-times. In Proc 5th Annual IEEE *Symposium on Logics in Computer Science*, Philadephia, 1990.

[6] Alur, R., T.A. Henzinger and P.H.. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans Software Engineering*, 1996, 22,181-201

[7] Aygalinc, P. and J.P.Denat. Validation of functional Grafcet models and performance evaluation of the associated systems using Petri Nets. *Automatic Control Production Systems A.P.I.I.*,1993, 27,81-93

[8] Clarke, E., E.A. Emerson and A.P. Sista. Automatic verification of finite state concurrent systems using temporal logic. *ACM Trans. on Programming Languages and Systems.*, 8,244-263.

[9] S.Roch, P.H.Starke. *INA - Integriert Netz Analysator*, Version 2.1, Hanbuch *Humboldt-Universitat zu Berlin*,1998, (in German)

[10] De Loor, P. J.Zaytoon and G.Villerman-Lecolier. Abstraction and heuristics for the validation Grafcet controlled systems *European Journal of Automation*, 1997, 31, 561-580

[11] Manna, Z. and A. Pnueli. The temporal Logic of reactive and concurrent systems. *Springer, Berlin*, 1992

[12] Ostroff, J.S. Temporal Logic for real-time systems. *Wiley, London*, 1989

[13] Ostroff, J.S. Verification of safety-critical systems using TTM/RTTL *In L.N.C.S.*, 1991, Vol. 600, pp 573-602, Springer, Berlin

[14] M. Rausch and H.-M. Hanisch. Net condition/event systems with multiple condition outputs. *Symposium on Emerging Technologies and Factory Automation*, Paris, France, October 1995, Proceedings volume 1, pp 592–600, INRIA/IEEE, 1995.

[15] M. Rausch. Modulare Modellbildung, Synthese und Codegenerierung ereignisdiskreter Steuerungssysteme. *PhD Thesis, Otto-von-Guericke-University of Magdeburg, Dept. of Electrical Engineering*, 1996.

[16] H.-M. Hanisch. Analysis of Place/Transition Nets with Timed Arcs and its Application to Batch Process Control. *Lecture Notes in Computer Science*, Vol. 691, pp. 282–299, Springer-Verlag, 1993.

[17] J.E. Reich, B.H. Krogh, I.D. Baxter. Symbolic Simulation-Based Techniques for Debugging Discrete Control Programs. *13th IFAC World Congress*, San Francisco, USA, 1996, Preprints, Vol. J, pp. 413–418.
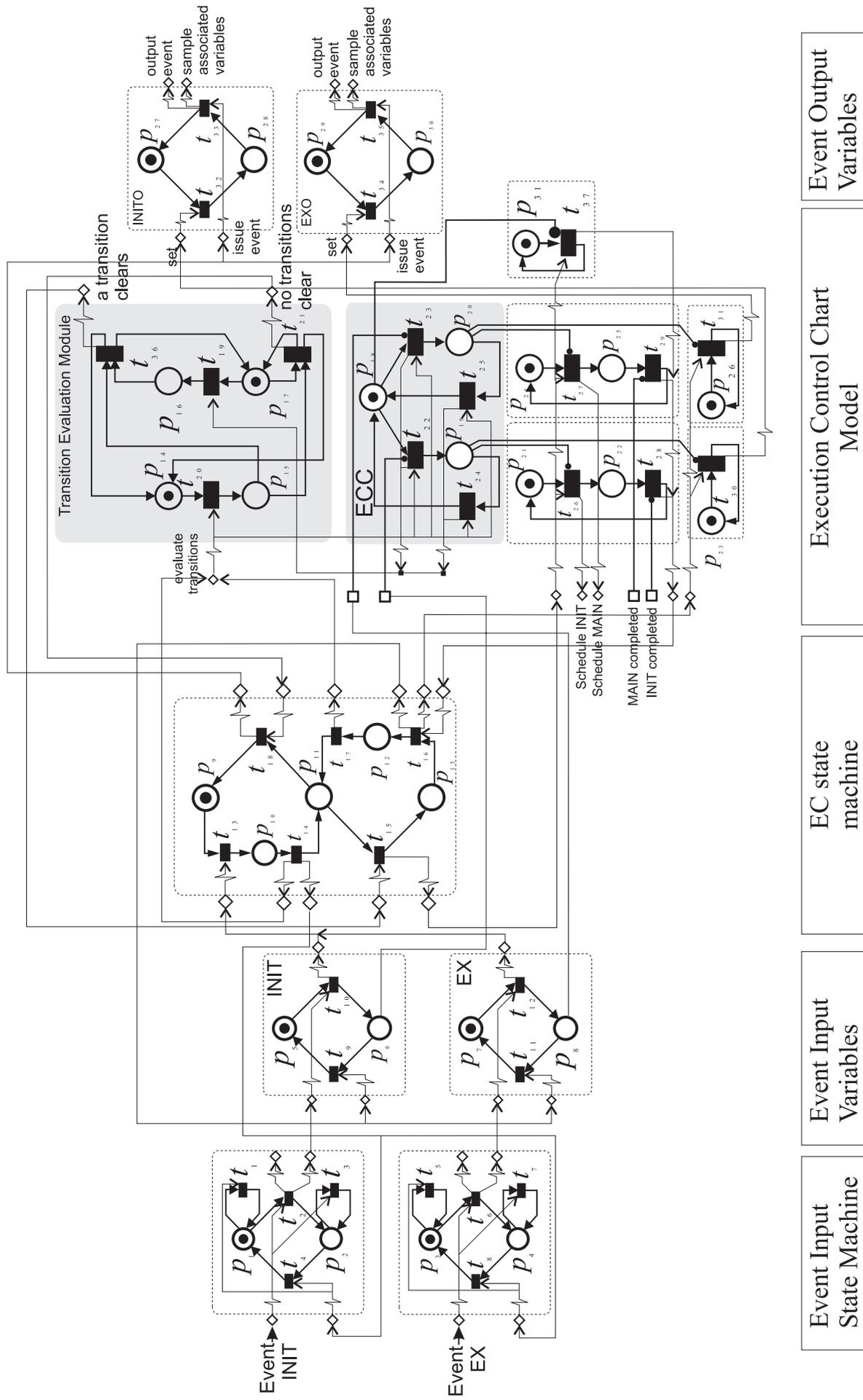
Figure 7: NCES model of the INTEGRAL-REAL function block execution control.