FORMALISMS FOR VERIFICATION OF DISCRETE CONTROL APPLICATIONS ON EXAMPLE OF IEC 61499 FUNCTION BLOCKS

Vyatkin, V.*, Hanisch, H.-M.*, Starke, P.*, Roch, S.*****

* Institute for Automation Technology (IFAT), Dept. of Electrical Engineering
Otto-von-Guericke University of Magdeburg, PO-Box 4120, D-39016 Magdeburg, Germany
Valery.Vyatkin@E-Technik.Uni-Magdeburg.DE

** Martin Luther University of Halle-Wittenberg
Dept. of Engineering Science
D-06099 Halle, Germany
Hans-Michael.Hanisch@iw.uni-halle.de

***Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6, D-10099 Berlin, Germany
{starke,roch}@informatik.hu-berlin.de

Abstract- *This paper presents the state of the art in the interdisciplinary research work on the development and application of modeling and verification techniques corresponding to the modern level of distributed control applications. We consider verification of function blocks following to the being developed international standard IEC61499, and suggest new modeling formalism of Signal/Net Systems (SNS), which is a place/transition net with usual token-flow arcs from places to transitions and vice versa, as well as with event arcs from transitions to transitions, and condition arcs from places to transitions which correspondingly force or enable transitions without passing tokens. Controller is modeled in a full deterministic and synchronous way (all the transitions fire according to the earliest firing rule) as a non-timed place/transition net, while the model of plant might have spontaneous transitions and discrete-timed arcs to model time consuming processes. Verification of controller/plant closed-loop system includes investigation of reachability problems for standalone controller and object, as well as proving safety properties for the closed-loop system. To express the latter we use an extended version of Timed Computation Tree Logic (TCTL), which operates both in terms of places and transitions.*

1 Introduction

Current practice of automation goes through the change to distributed architectures based on smart field devices instead of centralized programmable logic controllers. Correspondingly, new IEC standard 61499 [2] attempts to set the general rules of the development of distributed event-driven applications, upgrading the current practice ruled mostly by the IEC1131-3 standard [1].

Formal finite-state verification of measurement and control applications, well developed theoretically, currently experiences the growing demand by practical engineers. If properly applied, it promises the following results:

- Validation of that application behaves as specified, i.e. its outputs comply with formal specifications when specified inputs are imposed.
- Avoidance of undesirable states: check whether the application might come to a dead state, or it forces the system under control to move to the state specified as dangerous.
- Check whether differences in hardware or software architectures (to which the distributed applications are especially often subjected) might influence the behavior of the application.
- Non-trivial disclosure of never executable steps. According to the practice or rules accepted in many companies, control applications have to contain 100% tested code. Disclosure of never executable code is an important step to fulfill such requirements.

The general framework of the verification of real-time applications characterizes by three following components:

1. State/Transition formalism for modeling components of the application (controller) and environment (plant),
2. Specification language to define the desired (or undesired) behavior of the system.
3. Model checking facilities, capable to prove the validity or invalidity of the specifications for the given model.

Some previously accomplished works [5, 6, 7, 4] convinced us in the ability of Signal/Net systems to serve as the proper formalism. However, changing nature (or structure) of the modeled systems constantly requires to adjust some of the properties of the formalism to provide more adequate modeling and more exact verification results.

In this paper we discuss the reasons influenced modification of the formalism arised by the modeling of IEC1499 applications and interconnected systems, consisting of such applications and models of the environment. The paper is structured as follows: In Chapter 2 we discuss the new requirements to the modeling formalisms derived from the analisis of IEC61499. Chapter 3 presents the formal definition of the Signal Net Systems modeling formalism and the extended specification language of timed computation tree logic with

cation of SNS for modeling of basic structures of the controllers, and in Chapters 5 and 6 we show some details of modeling the structures of IEC 61499. Short discussion of the plans for future development of the modeling and verification of the IEC61499 applications concludes the paper.

2 Specific requirements to modeling of IEC1499 program structures

According to the draft of IEC61499 [2], the generic structure of an application is a function block, which can be either basic or composite. Functionality of a basic function block in IEC61499 is provided by means of algorithms, which process input and internal data and generate output data. The block consists of 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 the languages defined in IEC 1131. An application which is defined as a collection of interacting function blocks, connected by event and data flows, can be distributed over multiple resources and devices which is the significant difference of this new approach in contrast to IEC 1131. For example, as illustrated in Figure 1, a control application may include a function block FB1, implementing the controller itself, as well as a block FB3 responsible for operator interface (displaying the current state of the system and processing human interactions), and a block FB2 which implements a locking controller or supervisor. All these may be supplied by some standard or user defined blocks (IN1,OUT3) implementing a communication interface of devices where the blocks reside. For instance, the controller may be placed in one device D1 (let us say, a PLC), the operator interface in another device (PC) - D2, and the supervisor in another resource of the first device (PLC) - D1.

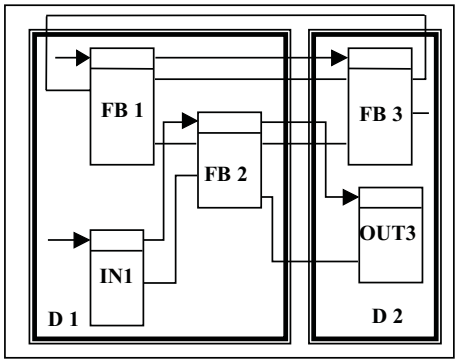

Figure 1: An example of distributed application.

When modeling of such a structure is concerned, especially combined with modeling of plant, one has to take in account the following considerations:

- The model of plant has to reflect asynchronous nature of the real plant. Thus if two units coexist in the plant, one of which generates event 1 and the other event 2, then the model has to provide all possible combinations of the events: none, only event 1, event 1 before event 2, only event 2, event 2 before event 1, event 1 and event 2 simultaneously.

provide an ability to generate finite (processing) sequences of states, initiated by either of the event combinations. These sequences sometimes have to be not interrupted by new events of the plant, while sometimes a certain parts of the controller register new events, while the other execute the processing sequences. Implementation of such a behavior requires either to except the earliest firing rule of the transitions (all the transitions fire as soon as they become enabled), or force them by an external clock transition.

- If the controller contains several modules, which are distributed among different devices (or resources, as they defined in IEC1131 or IEC61499), various combinations of clock transition firing has to be modeled. Consider the sketch example given in Figure 2. The "controller" part of the system consists of three components: Controller 1 which registers the event 1 executing the sequence S11,S12,S13, and generating the output event 3; the similarly functioning Controller 2, which registers event 2; and the View component, which displays the number of occurred events based on the data (events) generated by the controllers 1 and 2. In the reality the parts may be executed by different combinations of devices. Thus, we can imagine the state when even simultaneously occurred events 1 and 2 are followed by the incorrect display "1" due to the different speed of Controllers 1 and 2. Correspondingly the state space of the model must include the state (or states), reflecting such a situation.

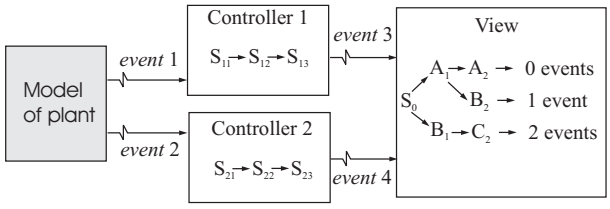

Figure 2: Processing of events by controller built of 3 components.

- When the (discretely) timed models are concerned, it must be noted that controller and plant have different time scales. Implementation of controllers' clock generator often requires to include the "synchronizer" transition of each controller in every state transition.

Certainly this behavior can be somehow modeled within currently existing formalisms, either asynchronous such as pure Petri nets, or synchronous, such as Grafcet. In both cases, however, the model loses its clarity due to various complications.

3 Signal/Net systems

3.1 General remarks

The distinctive feature of the SNS is that they preserve the graphical notation and the non-interleaving semantics of Petri nets and extend it with a clear and concise notion of signal inputs and outputs. Primarily these models were called Net Condition Event Systems (abbr. NCES), and later Signal/Net Systems.

tem problems, including controller synthesis, verification, and performance analysis and evaluation. The SNS support the way of thinking of and modeling a system as a set of modules with a particular dynamic behavior and their interconnection via signals. The behavior of a module can be described by a Petri net in the classical sense or even by a SNS. Each module is equipped with inputs and outputs of two types: condition inputs/outputs carrying state information, and event inputs/outputs carrying state transition information. An illustrative example of the graphical notation of a module is provided in Figure 3.

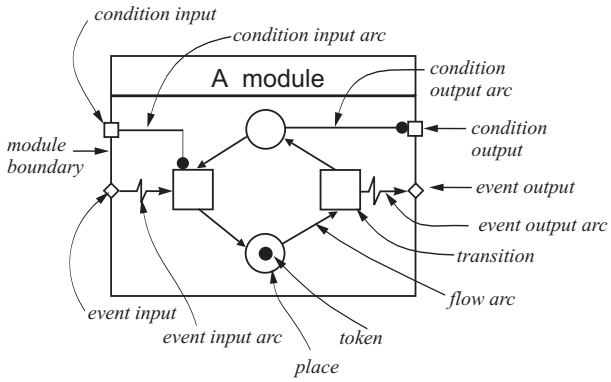

Figure 3: A Signal/Net System module.

Condition input signals as well as event input signals are connected with transitions inside the module. Whether a transition of a module fires does not only depend on the current marking (as it is the case in classical Petri nets) but also on the incoming condition and event signals. Incoming condition signals enable/disable a transition by their values in addition to the current marking. Incoming event signals force transitions to fire if they are enabled by marking and by condition signals. Hence, we get a modeling concept that can represent enabling/disabling of transitions by signals as well as enforcing transitions by signals. More than this, the concept provides a basis for a compositional approach to build larger models from smaller components. "Composition" is performed by connecting inputs of one module with outputs of another module. If the module is autonomous (it has no inputs), such a system is called "Signal/Event net", and it can be analyzed without any additional information about its external environment. Such features of SNS as event arcs and modular design closely correspond to the design techniques as provided by IEC 1499. Thus we have an intuitively clear but formal way to model a system. The modeling technique supports a bottom-up modeling as well as top-down strategy. Even after the composition, the state of the system is distributed, and the original structure of the modules is preserved. Hence, composition is far less complicated as building the cross product of automata or the interleaving language. This allows us to build efficient models of realistic scale.

3.2 Signal/Event nets

Signal/Event net is a place transition net defined as a tuple $N = [P, T, F, V, B, W, S, M, m_0, \tau]$ where:

1. P is a non-empty finite set of places,
2. T is a non-empty finite set of transitions, disjoint with P,
3. F is a subset of $(P \times T) \cup (T \times P)$ (the set of flow arcs),
4. V is a mapping which attaches a positive integer to every arc (the arc weight, $V : F \to \mathbb{N}$),
5. B is a subset of $P \times T$ (the set of condition arcs),
6. W is a mapping which attaches a positive integer to every condition arc (the condition arc weight, $W : B \to \mathbb{N}$),
7. S is a subset of $T \times T$, the irreflexive signal (flow) relation,
8. M is a mapping which attaches a (signal-processing) mode to every transition ($M : T \to \{\boxed{\wedge}, \boxed{\vee}\}$),
9. m_0 is a marking of P called the initial marking, and, finally,
10. $\tau : T \to \{Obliged, Spontaneous\}$ is a mapping which assigns to each transition the type (actually this typing makes sense only for independent transitions, i.e. those having no incoming event arcs);

The sets P, T and F, and the mappings V and m_0 are interpreted in the usual for Petri nets way.

If $[p,t]$ is an element of B then we say that p is a (or serves as) *condition of* t, i.e., in order to fire t it is necessary that p is marked with at least $W(p,t)$ tokens.

We consider condition arcs $[p,t]$ as relaying a piecewise constant signal which informs about the token load of the place p, i.e. the state of p.

If a pair $[t,t']$ of transitions is an element of the signal relation S, then we say that a *signal arc* leads from t to t', which means that firing the transition t sends a *signal-event* to the transition t'.

Signal-events reflect the second type of signals needed to connect the modules of a control device, the impulse type. They are described by time functions which have non-zero values only for isolated time points.

For any transition t the mode $M(t)$ determines the processing of the incoming signal-events. The default mode accepted throughout this paper is $\boxed{\vee}$. If $M(t) = \boxed{\vee}$ then to fire t it is necessary that at least one signal arc which targets t leads a signal-event, i.e. $\exists t_i \in T : [t_i, t] \in S$ and t_i is just firing. If, otherwise, $M(t) = \boxed{\wedge}$ then to fire t it is necessary that all signal arcs leading to t relay a signal-event.

If a transition t has no incoming signal arcs, i.e., the set

$$St := \{t' \mid [t', t] \in S\}$$

is empty, then the transition t is called *independent*, otherwise *forced*. By *Indep* we denote the set of all independent transitions of N, by *Forc* the set of all forced transitions.

The transition type mapping τ affects the rules under which the transition are selected for firing. It will be explained in the following section.

3.3 Rules of transition

Given a marking m, we say that a transition t has *token-concession* at m iff every (pre-)place p such that $[p,t]$ is in F holds at m at least as many tokens as the weight $V(p,t)$.

A transition t is said to be *enabled at the marking* m iff t has token-concession, its conditions are satisfied,

are present.

SE-systems are executed in steps, which are the sets of simultaneously firing transitions. The *firing rule* says, that executable steps are formed by first picking up a nonempty set of enabled independent transitions, which includes at least one of enabled obliged transitions, if any is present, and then adding as many as possible of those transitions that are forced to fire by signal-events produced by transitions in the step. This implies that in every non-dead *SE*-net there exists an independent transition.

If s is an executable step at m, then s may fire, which leads to the new marking $m' := m - s^- + s^+$. This is abbreviated as $m[s\rangle m'$. The marking m completely determines state of the S/E net model, and an executable step determines the state transition.

3.4 Timed Models

In this section we consider SE-nets under time constraints applied to the input arcs of transitions: to every pre-arc $[p,t] \in F$ we attach an interval $[eft, lft]$ of natural numbers with $0 \leq eft \leq lft \leq \omega$. Th interval is also referred to as *permeability* interval. If a pre-arc has no explicitly designated permeability interval it is assumed to be $[0, \omega]$.

The interpretation is as follows. Every place p bears a clock which is running iff the place is marked and switched off otherwise. All running clocks run at the same speed measuring the time the token status of its place has not been changed, i.e. the clock on a marked place p shows the age of the youngest token on p. If a firing transition t removes a token from the place p or adds a token to p then the clock of p is turned back to 0. A transition t is able to remove tokens from its pre-places (i.e. to fire) only if for any pre-place p of t the clock at place p shows a time $u(p)$ such that $eft(p,t) \leq u(p) \leq lft(p,t)$. Hence, the firing of transitions is restricted by the clock positions.

Let $N = [P, T, F, V, B, W, S, M, \tau]$ be a SE-net, eft a mapping from $F \cap (P \times T)$ to $\mathbb{N}_0$ and, lft a mapping from $F \cap (P \times T)$ to $\mathbb{N}_0 \cup \{\omega\}$ such that always $eft(p,t) \leq lft(p,t)$ holds. Then $TN = [N, eft, lft]$ is an *arc-timed signal-event net.*

A *state* of TN is given by a pair $[m, u]$ where m is a marking of P, and u is the P-vector of the clock positions. We assume that a clock which is switched off shows the time 0, and, that the time-scale used is integer. Therefore u is a marking too, and for any (realizable) state it holds: *if* $u(p) > 0$ *then* $m(p) > 0$.

The initial state $[m_0, u_0]$ of TN in general (but not necessarily) consists of the initial marking of N and the zero time vector.

Arc-timed signal-event nets are executed in steps too. The execution of a step does not take time.

A step s of N is said to be *enabled at the state* $[m, u]$ of TN iff it is enabled in the non-timed sense and for every pre-place p of a transition $t \in s$ it holds $eft(p,t) \leq u(p) \leq lft(p,t)$.

In timed models a specific type of obliged transitions required to let time elapse when no transitions is enabled by marking. This type is called *synchronous.* A synchronous transition usually is presumed to have no pre- or post places and one or more outgoing event arcs.

An enabled step is formed from at least one of the enabled obliged transitions, some of enabled spontaneous transitions, and all the transitions forced by the transitions are included in the step conditionally: if the formed step contains no transitions forced by the synchronous transition, then it is removed from the step.

Note, that in the timed models, a step is not necessarily non-empty. In the case if there are some enabled spontaneous transitions and no enabled obliged transitions, or no enabled synchronous transitions were included into the step, one of the possibly executable steps is the empty step $s = \{\}$.

Obviously, a step s may be enabled at the marking m in N, but not enabled at the state $[m, u]$ of TN because some clocks have not reached the *earliest firing time eft* or have passed already the *latest firing time lft*.

If there are no other enabled steps except for the empty step, the state $[m, u]$ of an arc-timed signal-event net may change by elapsing of one time unit to $[m, u']$ where

$$u'(p) \quad := \quad \begin{cases} u(p) + 1, & \text{if } m(p) > 0, \\ 0, & \text{else.} \end{cases}$$

If a state $[m, u]$ of TN is such that no non-empty step is enabled or can become enabled by elapsing of (one or more) time units then this state is called *dead.* Otherwise, the minimal number of time units after which at least one step becomes enabled is called the *delay* $D(m, u)$ of the state $[m, u]$. Hence, the delay is defined only for non-dead states.

For any place p we define the *clock stop position of* p as

$$csp(p) = \begin{cases} 1 + max\{lft(p,t) \mid t \in pF \wedge lft(p,t) \neq \omega\}, \\ \text{if this set is not empty,} \\ \\ max\{eft(p,t) \mid t \in pF\}, \\ \text{else.} \end{cases}$$

Consider two (reachable) states $[m, u], [m, u']$ which differ only in the clock positions u, u' in the following way: If $u(p) \neq u'(p)$ then $u(p)$, $u'(p) \geq csp(p)$. Then both states are indistinguishable in the sense that the same sequences of steps can be fired. Therefore, in our implementation, we stop every clock at their clock stop time, i.e. the clock position will not be increased by elapsing a time unit, although the clock is "running". In this way states of the above described kind will be identified.

3.5 Timed Computation Tree Logic with state/event variables

Timed Computation Tree Logic (abr. TCTL) was introduced in [8] as an extension of *Computation Tree Logic* (abr. CTL) proposed in [3]. It allows to combine the qualitative temporal assertions together with real-time constraints, as intervals $[l, h]$ with $0 \leq l \leq h \leq \omega$ attached to the modalities X, F and U.

If $EX\varphi$ is a formula of CTL then $EX[l,h]\varphi$ is a formula of TCTL which is satisfied by a state z if this state has a successor z' satisfying the formula φ and such that the transition from z to z' takes at least l and at most h time units.

The *interpretation* of formulas from TCTL is done on a structure (or *model*) $\Sigma = [Z, \Rightarrow, D]$ where

1. Z is a non-empty finite set of states z,
2. $\Rightarrow \subseteq Z \times Z$ is the state transition relation,

A CTL formula is built from state-formulas (i.e. Boolean formula over state-related predicates) and temporal operators (modalities). However, our application needs require references to events (which are modeled by the net transitions) along with references to states. Transition-related predicates signal about whether a certain (S/E net) transition is included in the step which makes a state transition. For example, formula $t_1 \wedge t_2$ is true for the state transitions, defined by the steps including both t_1 and t_2.

The state/event precedence relation is defined as follows: First denote state/step precedence. If step s is executable at state z and result of its execution is state z' then we denote this as $z \Rightarrow^s z'$ and say that step s is next to z and precedes z'.

Let $z = [m, u]$ be a state, and $Tr(z)$ is a set of steps, executable at z (correspond to the arcs in the reachability graph going out of the vertex corresponding to the state z). We say that transition t is next to the state z if it is included at least in one step of $Tr(z)$. Similarly, state z is next to the transition t iff t is included to a step s: $\exists z' : z \Rightarrow^s z'$.

The relations of following/precedence can be then defined inductively. For example, state z follows transition t iff either z is next to t or there exist z' next to t such that z follows z'.

In our application here, Σ always will be the reachability graph of an arc-timed signal-event system. For any state $z = [m, u]$ the number $D(z)$ is the number of time units which have to elapse at z before a non-empty step can be executed.

A *fullpath* σ in Σ is an infinite sequence $(z_i, s_i), i \in \mathbb{N}_0$ of states and state transitions, starting in a state, such that $z_i \Rightarrow^{s_i} z_{i+1}$ holds for all $i \in \mathbb{N}_0$. Transition s_i may be empty.

For any fullpath σ and every state $z \in Z$ we put

1. $D(\sigma, z) = 0$, if $z = z_0$ (i.e. if σ starts at z_0),
2. $D(\sigma, z) = D(z_0)+D(z_1)+\ldots+D(z_{k-1})$, if $z = z_k$ and $z_0, \ldots, z_{k-1} \neq z$.

With other words, $D(\sigma, z)$ is the number of time units after which the state z on the fullpath σ is reached the first time, i.e. the minimal time distance from z_0. For a transition $s : z \Rightarrow^s z'$ the number $D(s) = D(z')$.

Consider some examples of TCTL formulas using transition-related expressions:

1. Transition t_1 fires in the state z: $z \models EXt_1$.
2. There exists state where transitions t_1 and t_2 fire simultaneously and this state is always followed by the state where marking in p_1 is 1 within 5 to 7 time units: $z_0 \models EF((t_1 \wedge t_2 \rightarrow EF[5,7](m(p_1) = 1))$
3. Transition t_1 is followed by transition t_2 within time interval $[2,3]$: $z_0 \models AG(t_1 \rightarrow EF[2,3](t_2))$;

4 Modeling of controllers and interconnected systems

4.1 General considerations

We assume that implementation of the functionality of the blocks as described by the standard is achieved by the corresponding program modules, provided either by operating system, or included in the code of tion block are modeled by SNS we keep in mind the correspondence between SNS patterns and the algorithmic patterns, which form the program modules. In this context the formalism of SNS is used for modeling of objects of different nature: plants and control devices. This requires to be careful with usage of some of its features. For example, while model of plant often benefits from the use of non-determinism (in conflict resolution), in the model of controller it would be inadequate. Spontaneous transitions are helpful to model events in the plant, but not appropriate when the controller is concerned. SNS allows simultaneous firing of several transitions, but a computing device cannot perform execution of several commands simultaneously. We must make sure that simultaneous transitions correspond either to the commands, executed in different resources or devices, or to the commands, whose order of execution is really irrelevant. The following examples show the principles of modeling. In general, transitions are used to model commands, while places model states, or values of variables.

4.2 Data storage and assignment

Boolean variable cell can be modeled by the net having two places p_0 and p_1 and two transitions t_s and t_r as shown in the Figure 4. Setting of the variable is modeled by transition t_s and resetting by t_r.

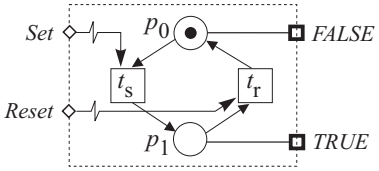

Figure 4: Boolean variable cell.

4.3 Linear sequence of commands

Consider how modeled the linear sequence of commands SET X; RESET Y. In the model, presented in Figure 5, transitions t_1, t_2 which correspond to the commands are forced by transition t_{sg} which makes them to fire as soon as they become enabled. Thus t_{sg} models synchronous nature of the computing device, where commands are forced by the clock generator of impulses. Transition t_{sg} has to fire every time. In the timed models, however time elapses only when no transition is able to fire immediately. Therefore, to obtain the required behavior, t_{sg} has to fire only when any of the transitions, forced by it, is enabled. That is why t_{sg} is made a *synchronous* transition. Model of every resource or device (in terms of IEC1499) has only one synchronous transition, which forces thereby all the commands within this resource/device.

4.4 Boolean operators

Since every Boolean variable is modeled by two places (as was shown in Figure 4), we do not need a specific model for getting negation of a Boolean variable. As for AND, and OR operations, they can be modeled as shown in Figures 6,7 correspondingly. Both models have two incoming event signals: compute and reset. Computation of the result takes one state transition.

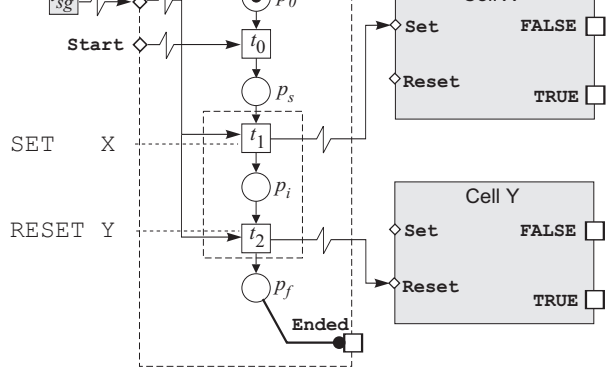

Figure 5: Model of linear sequence of two commands.

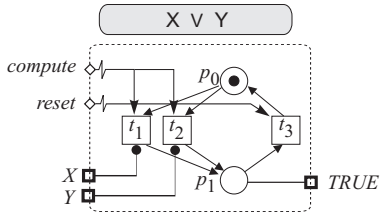

Figure 6: SNS model of logic OR.

4.5 Interconnected systems

In the example in Figure 8,a we have a non-timed model, where the left module corresponds to the model of plant, its transitions 1 and 2 are spontaneous. Places 1 and 2 model two states of the plant. The right module corresponds to the controller, which registers the event of transition from state 1 (p_1) to the state 2 (p_2) - transition t_3, then performs some computations - transition t_4, and returns to the initial state. Transition t_5 is obliged and serves to model the clock generator of the controller. Thus, the first step in the given marking is formed from the spontaneous t_1, forced t_3, and obliged t_5: $S_1 = \{t_1, t_3, t_5\}$. Second steps can be $S_2 = \{t_4, t_5\}$ and $S2 = \{t_2, t_4, t_5\}$. In the timed model in Figure 8,b the arc (p_1, t_1) is marked with the time interval [1,3], where 1 is the low time bound, and 3 is the high bound, which means that t_1 would fire only within [1,3] interval. However, at normal firing rules t_1 would never fire because t_5 is always enabled and is included in every step. For this purpose t_5 is made synchronous, it is included into the step only if t_4 also would be included (in this case, iff t_4 is enabled). Thus we can get the following sequence of steps: $S1 = \{\}$ - time elapses by 1 unit, $S2 = \{t_1, t_3\}$, $S3 = \{t_4, t_5\}$, and so on.

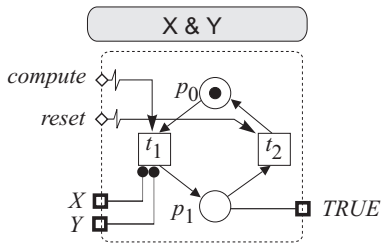

Figure 7: SNS model of logic AND.

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

5 Modeling of a basic function block

The standard provides certain hierarchy of program structures, which implies the corresponding strategy of modeling - we begin with the simplest basic functional blocks, gradually extending the modeling framework to the whole applications and systems. We concentrate mostly 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 component 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 state or time delays when it is essential.

As for the data types defined in the standard, we divide them to 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 relay the value of variable without affecting the token 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 modelled 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:

- Event input state machine (EI-SM) implementation module (one for each event input);
- Event input variables storage (EIVS) models (one for each event input);
- Model of EC operation state machine (ECO-SM);
- Module implementing ECC (ECC model), including the sub-modules implementing actions and algorithms (optional);
- Model of the scheduling function, which serves in the standard to implement the discipline of call of algorithms. Since the draft of the standard provides little information on the issue, so far we accepted only the "immediate execution" mode;
- Event output variables (EOV) models (one for each event output);

Details of each model can be found in [9, 11, 10]. Consider in particular the execution control of the block. Operation of the Execution Control is described in the IEC1499 by means of the state machine (EC SM) shown in Figure 8,c with transitions and actions defined as in the Table 1.

Multiple actions, associated with arcs of the automaton, are interpreted as executed sequentially. Thus,

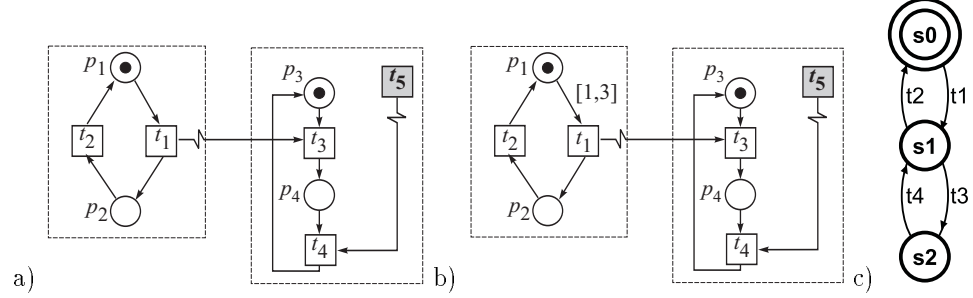

Figure 8: a) Non-timed model of the interconnected system; b) Timed model of the interconnected system; c) Execution control state machine.

actions associated with t_1 are interpreted as follows: 1.Confirm input(s) mapped; 2. Evaluate transitions. First operation sends an event signal to the model of corresponding state machine implementing the storage element of the event input, and the second sends an activation signal to the NCES model of the Execution Control Chart. It is essential to ensure that the latter signal is issued after the first operation has been completed, because value of the event input variable might be required for evaluation of transitions in the model of ECC.

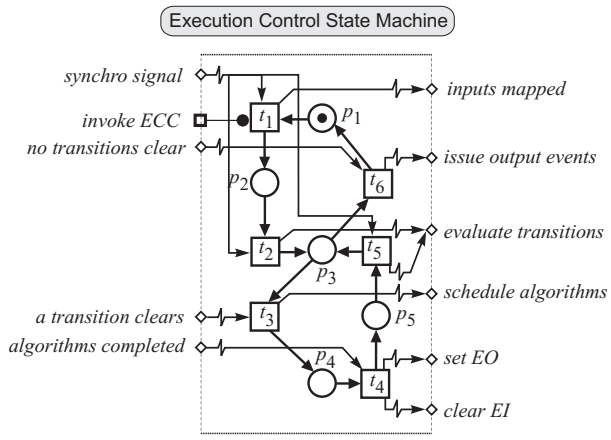

Figure 9: SNS model of the execution control state machine.

Basic structure of ECO-SM model (Figure 9) is similar to the original ECO SM (places p_1,p_3,p_4 correspond to the states S0,S1,S2). Marking in the place p_1 (state S0) means that the state machine is ready to process event, which we call "ECC idle". Transitions t_1, t_2 and t_5 are forced by the synchro-signal, which is issued by the resource in which the block is placed. Operation of the ECO SM is started when the resource sets the flag "Invoke ECC". We introduce also additional "transitional" places p_2 - which ensures that variable setting is completed before the transition evaluation, and p_5 - it ensures that new transition evaluation is done after the event variables used in the previously cleared transition have been reset.

Literally, the standard provides the following definition of the last operation associated with transition t_4: "This operation consists of resetting to FALSE (0) the values of all the EI variables used in evaluating the transition conditions of the previously cleared transition."

In our opinion, this phrase is rather ambiguous, since no definition is given what means "used in evaluating the transition conditions of the previously cleared transition". Logically, these are all the transitions from

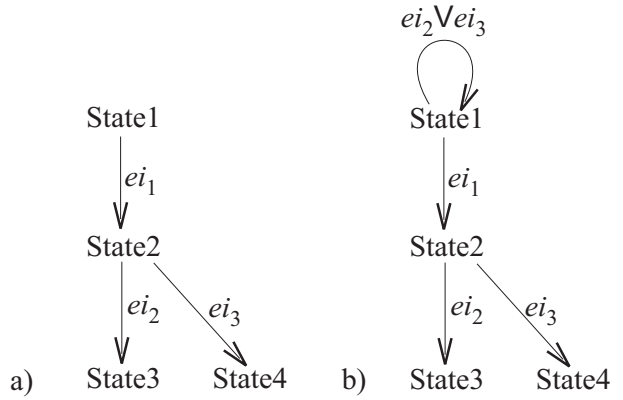

Figure 10: a)ECC which accepts sequence ei3,ei1 as ei1,ei3; b)Corrected ECC.

the previously active state of the ECC. However, the standard does not require the set of transitions to be complete. In this case, if an event occurs, variable of which is not included in any of the transition conditions in a state S, it would not be cleared. Consider example. Let function block has 3 event input variables: ei1, ei2, ei3 and the execution control chart presented in Figure 10-a. Desired behavior of the block is to pass to the state 3 at input sequence ei1,ei2 and to the state 4 at the sequence ei1,ei3. Implicitly the evaluation rules imply that if ei3 occurs at the state 1, the ECC remains to be in this state. However, the sequence ei3, ei1, ei2 might take the ECC in either of the states 3 or 4, instead of state 3, as desired. This is due to the fact, that ei3 will not be cleared if occurs at state 1, and will remain TRUE in state 2, where it might fire the transition to the state 4, after the event ei2 occurs, depending on the sequence of condition evaluation. That is defined in the standard as follows: "This operation (t1-evaluate 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. Software tools may provide means for determining the order in which the EC transitions following an active EC state are to be evaluated."

To provide the required behavior, the correct ECC must look like one in Figure 10-b. In this case ei3 will be cleared if occurred at the state 1.

6 Modeling of resource

Resource is a functional unit contained within a device which has independent control of its operation, and which provides various services to applications,

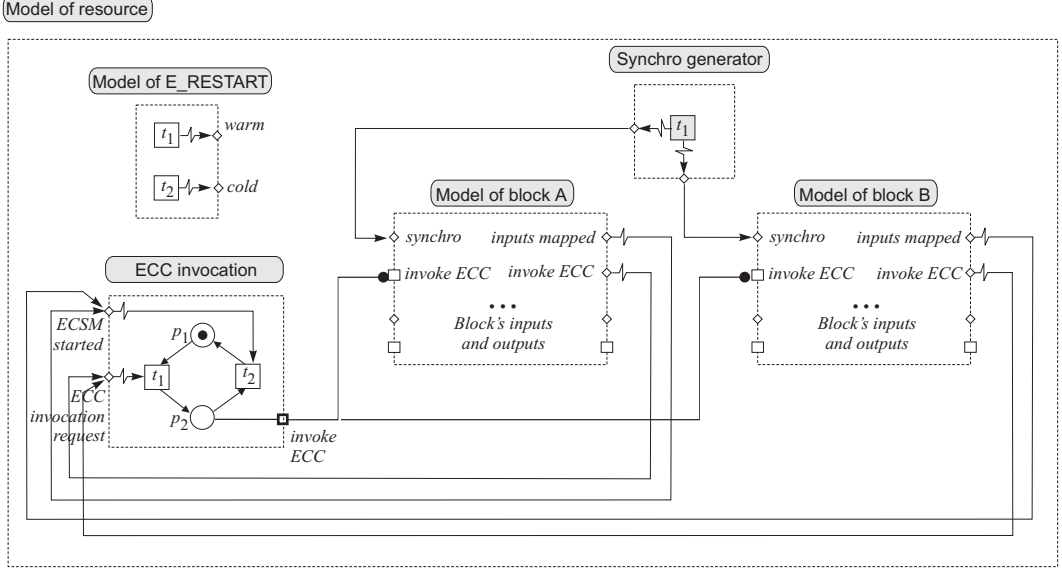

Figure 11: SNS model of resource.

including the scheduling and execution of algorithms. The RESOURCE defined in IEC 1131-3 corresponds to the resource defined above. A device contains one or more resources. We model the following functions of resource: processing of invocation requests, scheduling and execution of algorithms, synchronizing the execution of blocks, and *E_RESTART* block, which generates corresponding signals for the cold and warm restart events. We understand that in such a way that no requests should be lost in case if the execution control is unable to process the request immediately.

Sketch model of the resource containing two function blocks is given in Figure 11. Processing of invocation requests is modeled by the SNS module which sets a Boolean flag (token in the place p_2) after at least one ECC invocation request was generated by models of event input variables of the blocks contained in the resource. When the execution control becomes idle, it uses this condition to start work again. Once processing of the event is started by the execution control, it issues the signal "Input mapped" to the models of event inputs, as well as to the model of invocation processing which fires transition t_2 and returns token from p_2 to p_1.

7 Conclusion and future work

Several new features of the SNS modeling formalism have been discussed in the paper. Those have been inspired by the needs of modeling and verification of IEC61499 applications. Some of them have been already implemented in the model-checking tools, such as SESA net analyzer, or model generator and analyzer VEDA. The others yet have to be implemented. Preliminary analysis shows that the proposed formalism provides more correct way of modeling compared to the previously used ones, although extends the number of reachable states, especially in the timed models. An effective way to cope with this problem, is to use both non-timed and timed models for the same system. While the former reveals most of the faults, related to the sequencing of signals, and the dead states, the latter can be used only for some performance estimations.

8 Acknowledgement

The work is supported by the Deutsche Forschungsgemeinschaft under the references Ha 1886/10-1 and Sta 450/4-1.

References

[1] *International Standard IEC 1131-3, Programmable Controllers - Part 3.* International Electrotechnical Commission, Geneva, Suisse, 1993.

[2] *Function Blocks for Industrial Process Measurement and Control Systems.* International Electrotechnical Commission, Tech.Comm. 65, Working group 6, Geneva, 1998.

[3] Emerson, Mok, Sistla, and Srinivasan. *Quantitative Temporal Reasoning*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. 1991.

[4] H.-M.Hanisch, T.Pannier, D.Peter, S.Roch, and P.Starke. Modeling and verification of a modular lever crossing controller design. *Automatisierungstechnik*, 48, 2000.

[5] H.-M. Hanisch and A. Lüder. On the mutual dependency of safety controllers and procedural controllers. In *Proc. of International Symposium on Industrial Electronics (ISIE'98)*, pages 622–627, Pretoria, South Africa, July 1998.

[6] H.-M. Hanisch, J. Thieme, A. Lüder, and O. Wienhold. Modeling of plc behavior by means of timed net condition/event systems. In *International conference on Emerging Technologies and Factory Automation (ETFA'97)*, Los Angeles, USA, September 1997.

[7] L.E.Pizon, H.-M. Hanisch, and M.A.Jafari. Sequential specifications modeling with temporal logic and net/condition event systems. In *Proc. of International workshop on Discrete Event Systems (WODES'98)*, Cagliari,Italy, August 1998.

[8] P.Starke, S.Roch, K.Schmidt, H.-M. Hanisch, and A.Lüder. Analysing signal-event systems, July 1999. Internal report,.

[9] Valeriy Vyatkin and Hans-Michael Hanisch. A modeling approach for verification of IEC1499 function blocks using net condition/event systems. In *Proceedings of the ETFA'99 Workshop*, pages 261–270, Barcelona, Spain, 1999.

[10] Valeriy Vyatkin and Hans-Michael Hanisch. Modeling of IEC 61499 function blocks as a clue to their verification. In *Proc. of XI Workshop on Supervising and Diagnostics of Machining Systems*, Karpacz, Poland, March 2000.

[11] Valeriy Vyatkin, Hans-Michael Hanisch, Peter H. Starke, and Stephan Roch. Systematic modeling of discrete event systems with net condition/event systems and its application to verification of IEC1499 function blocks. *IEEE Transaction on Man, Machine, and Cybernetics, Special Issue on Discrete Event Systems*, submitted.