IEC61499, distributed applications, discrete state modelling, verification

Valeriy VYATKIN
Hans-Michael HANISCH

MODELING OF IEC 61499 FUNCTION BLOCKS A CLUE TO THEIR VERIFICATION

Abstract. This paper suggests a way of modelling of discrete measurement and control applications following to the new international standard IEC 61499. The applications which are distributed and event driven, require an adequate formalism, clear enough to be adopted by engineers. The presented formalism of signal/net systems proved to be suitable for modelling of distributed applications alone, as well as for the interconnected systems combined with the model of the plant.

Based on the modelling, numerous problems of verification can be effectively solved. The model checking is performed by a bunch of developed analysing tools, including the VEDA (Verification Environment for Distributed Applications) and SESA (Signal/Event systems analyser). It includes: test for reachability of dangerous states, search of the never executable code, and validation or invalidation of arbitrary input/output specifications.

INTRODUCTION

Modelling of controllers is widely recognised in academic circles as an important step towards their verification and validation of the control systems. Gradually this understanding finds its way to the practical applications. Meantime practice of the controller design undergoes dramatic changes, which are associated with the use of new distributed architectures based on smart field devices and distributed controllers. The latest trends in the programming for such architectures are presented in the IEC61499 – international standard under development [3]. Purpose of the research, whose some results this paper presents, is to provide an adequate modelling and verification framework for the applications developed following to the IEC61499. The standard especially emphasises on the software reusability issue, stressing on such means as modularity, encapsulation, and polymorphism, which explains the need for a theoretical ground, which would enable development and investigation of IEC61499 applications by formal methods.

Based on the analysis of the standard, we have chosen the Signal/net systems (abbr. SNS) as the basic formalism. The SNS were developed for discrete state modelling of manufacturing systems and their control. They were partly inspired by the idea of Condition/Event systems as defined by Sreenivas and Krogh in 1991 [5]. 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. They have been applied to a few discrete event system

Martin-Luther University of Halle-Wittenberg
Department of Engineering Science

problems, including controller synthesis, verification, and performance analysis and evaluation.

The SNS support the way of thinking of and modelling a system as a set of modules with a particular dynamic behaviour and their interconnection via signals. The behaviour 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 1.

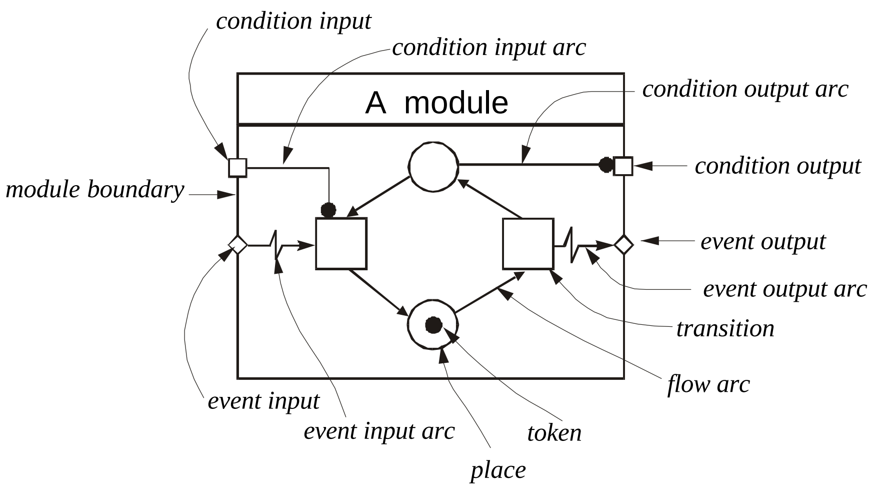


Figure 1. Graphical notation of the module.

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 modelling 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 analysed 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 modelling technique supports a bottom-up modelling 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.

Due to the space limitations, we illustrate in this paper only basic principles of modelling with SNS, omitting rigorous definitions and proofs. They can be found in the journal versions, one of which is already submitted [7].

FIRING RULES FOR MODELLING OF INTERCONNECTED SYSTEMS

Modelling of a controller interconnected in the closed loop system with a plant gives best results but also brings a lot of complications due to the deeply different nature of processes in plants and controllers. Some experience in modelling of interconnected systems and some features of distributed applications have required a certain modification of the firing rules of Signal/Event nets. Lack of space would not allow us to give the rigorous definitions of Signal/Event nets behaviour, but interested reader can find it in [4]. We present here only remarks on the firing rules which have to be mentioned in addition to the intuitively clear semantics of Petri nets. Attempts to use such already existing formalisms, as purely asynchronous Petri nets, or the Grafcet, which is a purely synchronous Petri net variation, brings considerable complication of the model.

Transitions in SNS can be either independent, or forced. Independent transitions are those having no incoming event arcs, while the forced have one or more of such arcs. Independent transition can be either spontaneous, or obliged to fire. A spontaneous transition may or (may not) fire when it is enabled by tokens. In non-timed models we assume that at least one of enabled independent transitions (including at least one of currently enabled obliged transitions, if any) must fire.

Transitions in SNS are executed in steps. A step is formed from some (at least one, or no, depending on timed/non-timed the model is) of currently enabled spontaneous transitions, at least one of currently enabled obliged transitions, and all the transitions forced by event signals coming from the transitions included in the step.

In timed models a specific type of obliged transitions is required to let time elapse when no transition is enabled by marking. This type is called *synchronous*. A synchronous transition has no pre- or post-places, and has one or more outgoing event arcs. It fires only if at least one of the transitions forced by it also is included into step.

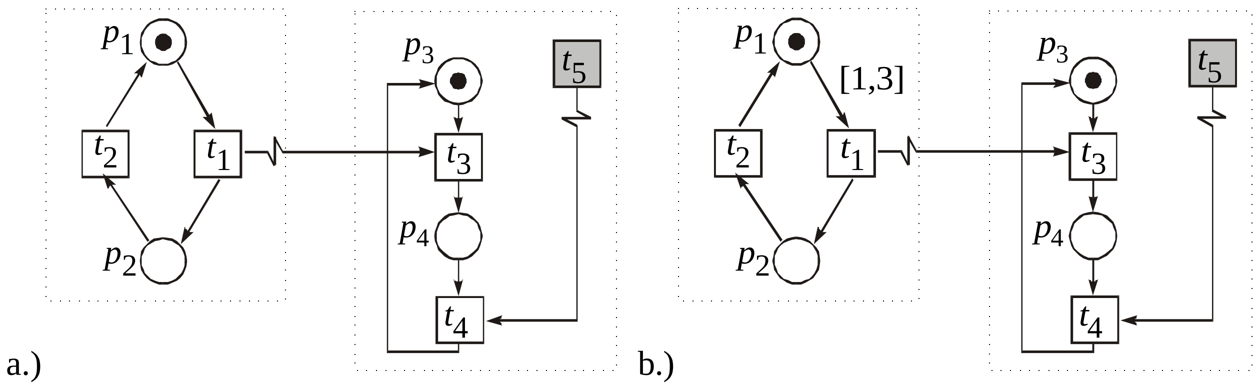


Figure 2. Non-timed (a) and timed (b) sketch plant/controller models.

In the example in Figure 2,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 (p1) to the state 2 (p2) – transition t3, then performs some computations – transition t4, and returns to the initial state. Transition t5 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 t1, forced t3, and obliged t5: S1={t1,t3,t5}. Second steps can be S2={t4,t5} and S2={t2,t4,t5}.

In the timed model in Figure 2,b the arc (p1,t1) is marked with the time interval [1,3], where 1 is the low time bound, and 3 is the high bound, which means that t1 would fire only within [1,3] interval. However, at normal firing rules t1 would never fire because t5 is always enabled and is included in every step. For this purpose t5 is made synchronous, it is included into the step only if t4 also would be included (in this case, iff t4 is enabled) . Thus we can get the following sequence of steps: S1={} – time elapses by 1 unit, S2={t1,t3}, S3={t4,t5}, and so on.

PROGRAM STRUCTURES DEFINED BY THE IEC1499 STANDARD

According to the draft of IEC 1499 [IEC1499], the generic structure of an application is a function block, which can be either basic or composite. 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 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 3, 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 would implement 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.

Causal behaviour of the block (i.e. sequencing of algorithms' calls) is organised 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. Therefore, 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.

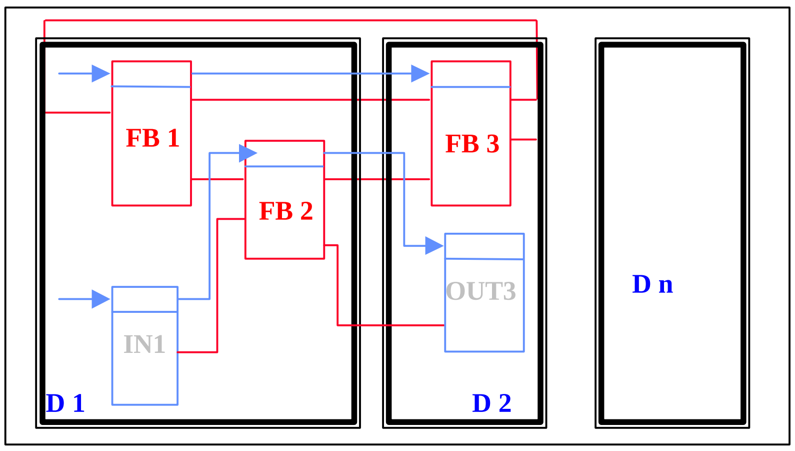
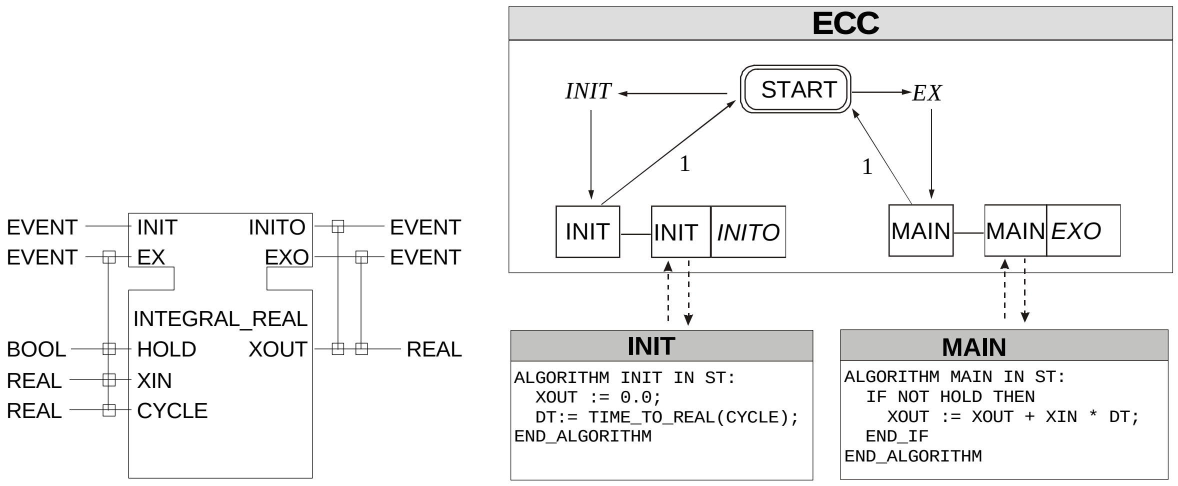


Figure 3.



```
ALGORITHM INIT IN ST:
  XOUT := 0.0;
  DT:= TIME_TO_REAL(CYCLE);
END_ALGORITHM
```

```
ALGORITHM MAIN IN ST:
  IF NOT HOLD THEN
    XOUT := XOUT + XIN * DT;
  END_IF
END_ALGORITHM
```

Figure 3. An example of interface of function block, its execution control and definition of algorithms.

The ECC consists of EC states exactly one of which is initial, EC transitions, and EC actions. The EC initial state shall have no associated EC actions. Other EC states may have zero or more associated EC actions. An EC action consists of the associated algorithm, and the event to be issued on completion of the algorithm. The ECC can utilise (but not modify) event input (EI) variables, and utilise and/or modify event output (EO) variables. Also, the ECC can utilise but not modify Boolean variables declared in the function block type specification. 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 utilising one or more event input variables, input variables, output variables, or internal variables of the function block.

Figure 3 shows an example of both the external interface of the block INTEGER-REAL, and its execution control chart.

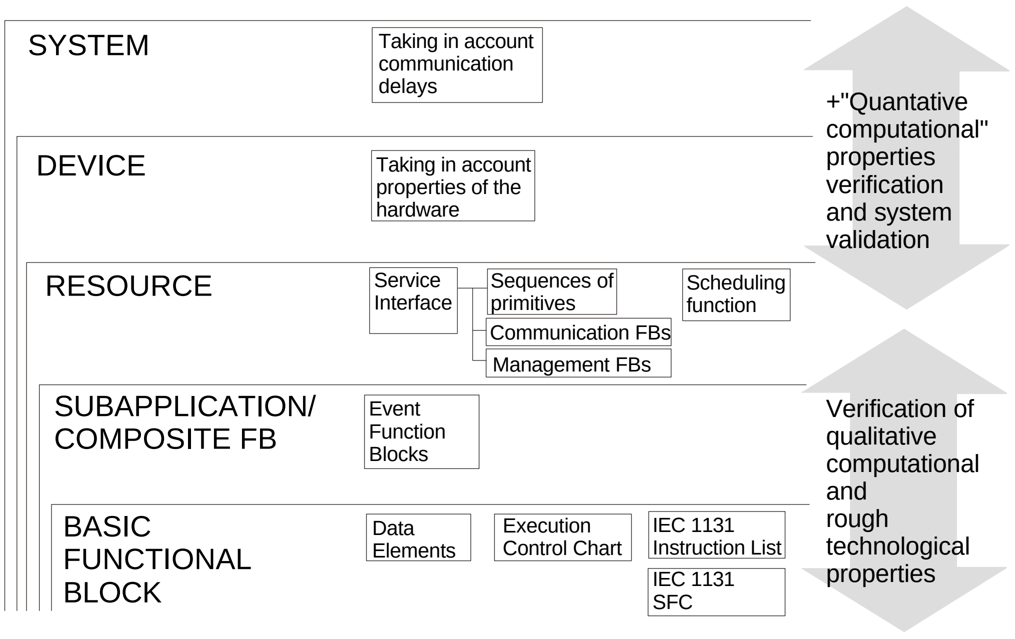

SYSTEM
Taking in account communication delays
DEVICE
Taking in account properties of the hardware
RESOURCE
Service Interface
Sequences of primitives
Communication FBs
Management FBs
Scheduling function
SUBAPPLICATION/ COMPOSITE FB
Event Function Blocks
BASIC FUNCTIONAL BLOCK
Data Elements
Execution Control Chart
IEC 1131 Instruction List
IEC 1131 SFC
+"Quantative computational" properties verification and system validation
Verification of qualitative computational and rough technological properties

Figure 4. An extent of modelling and verification results depending on the scale of modelled structures.

ESSENTIALS OF MODELING BY SIGNAL/NET SYSTEMS

The work on modelling of the structures of IEC1499 is based on some previously conducted research on the modelling of interconnected controller/plant systems. Interested reader may refer for example to the work [2]. Here we consider the function blocks of IEC1499 as parts of a controller.

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 the block by compiler. When the components of function block are modelled 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 modelling 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 modelling. In general, transitions are used to model commands, while places model states, or values of variables.

DATA STORAGE AND ASSIGNMENT

Boolean variable cell can be modelled by the net having two places p_0 and p_1 and two transitions t_s and t_r as shown in the Figure 5. Setting of the variable is modelled by transition t_s and resetting by t_r.

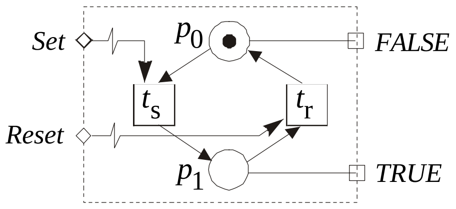


Figure 5. Model of a Boolean variable cell.

LINEAR SEQUENCE OF COMMANDS

Consider how modelled linear sequence of commands SET X; RESET Y. 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 generator of impulses.
Transition t_{sg} has to fire every time. In timed models, however time elapses only when no transition is able to fire immediately. Therefore, to obtain the required behaviour, t_{sg} has to fire only when any of the transitions, forced by it, is enabled. To implement this specific feature, we introduce particular type of transitions, called "synchronous". Model of every device (in terms of IEC1499) has only one synchronous transition, which forces thereby all the commands within this device.

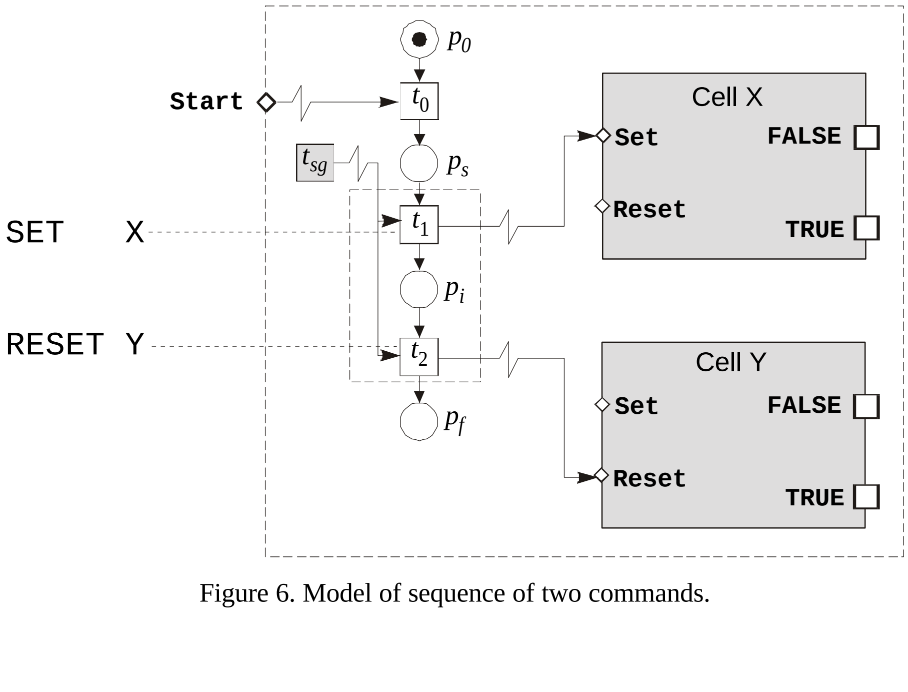


Figure 6. Model of sequence of two commands.

Sequence of transitions t_1, place p_i and transition t_2 can be considered as sort of transition from place p_s to place p_f. Then the model of sequence can be seen as an SNS module, forced to start by transition *t*, as depicted in the Figure 7.

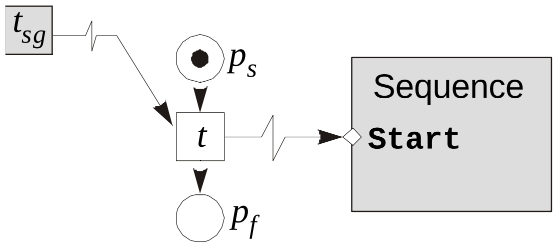


Figure 7. Model of sequence initiated by one transition.

CONDITIONAL CHOICE

Conditional choice of type **IF X THEN Sequence A ELSE Sequence B** can be modelled in SNS as shown in Figure 8. Transition t_A has incoming condition arc which relays value of X, and t_B has incoming condition arc marked with not X. Since the conditions are orthogonal, only one of the transitions is able to fire.

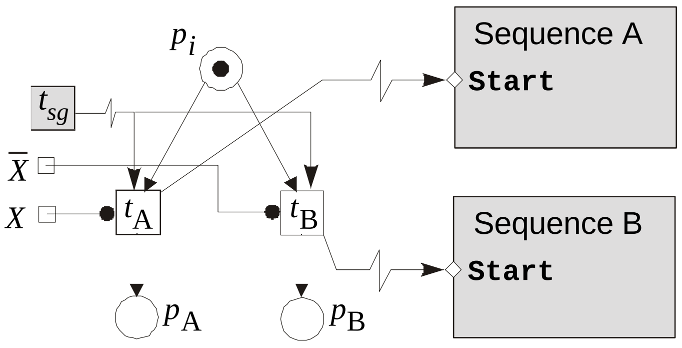


Figure 8. Model of conditional choice

BOOLEAN OPERATIONS

Since every Boolean variable is modelled by two places (as was shown in Figure 5), we do not need a specific model for getting negation of a Boolean variable.

As for AND, and OR operations, they can be modelled as shown in Figure 9, a) and b) correspondingly. Both models have two incoming event signals: *compute* and *reset*. Computation of the result takes one state transition.

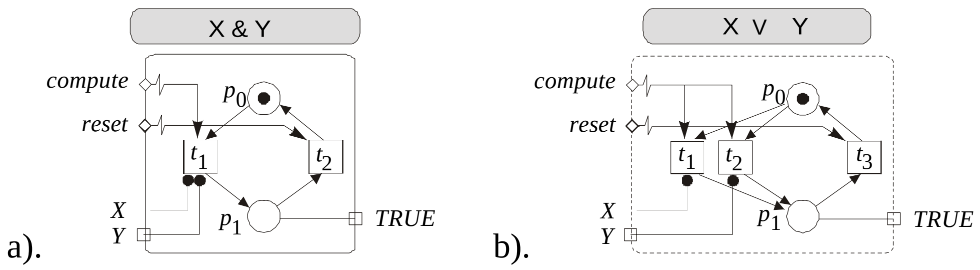


Figure 9. Models of Boolean operations.

The over examples of modelling serve to illustrate basic principles of mapping from commands of programming language to SNS models and back. In the models presented below we would not provide the exact correspondence between SNS models and algorithmic equivalents, relying on the presented examples.

MODELING OF BASIC FUNCTION BLOCK

The standard provides certain hierarchy of program structures, which implies the corresponding strategy of modelling - we begin with the simplest basic functional blocks, gradually extending the modelling 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 modelling of the component algorithms as long as they do not strongly concern the execution logic of the block. Calls of the algorithms can be modelled by the corresponding state or time delays when it is essential. This agrees with the practical view on modelling - to get more or less valuable results we need to concentrate only on the important issues sacrificing the neglectable ones.

As for the variety of data types defined by the standard, we divide them to the four categories: event signals, which are modelled mainly by event arcs of NCES; Boolean variables which can be modelled 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:

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

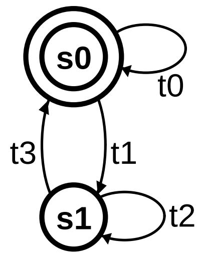


Figure 10. Event input state machine

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 element which exhibits the behaviour defined by the state machine in Figure 10. Its transition arcs are marked both with conditions of transitions and with operations, executed upon the transitions as listed in the Table 1. Though it is not directly stated in the draft of the standard, we assume that the state machine defines a Mealey automaton with input and output symbols associated with arcs.

Transition	Condition	Operation
t0	Map input (1)	None
t1	Event arrives	ECC invocation request (2)
t2	Event arrives	Implementation dependent
t3	Map input (1)	Set EI variable (3)

NOTES:

1. This condition is issued by the ECC state machine shown in Figure 2.2.2.2-2.
2. This operation consists requesting the resource to invoke the ECC.
3. This operation consists of setting the value of the corresponding EI input variable to TRUE (1) and *sampling* the input variables associated with the event input by a WITH declaration as described in 2.2.1.1. This sampling shall be a *critical operation*.

Table 1 - Transitions of event input state machine

The SNS model of event input is shown in Figure 11. It includes model of the storage element (places p_1-p_2, transitions t_1-t_4), and model of the variable (places p_3-p_4, transitions t_5-t_6).

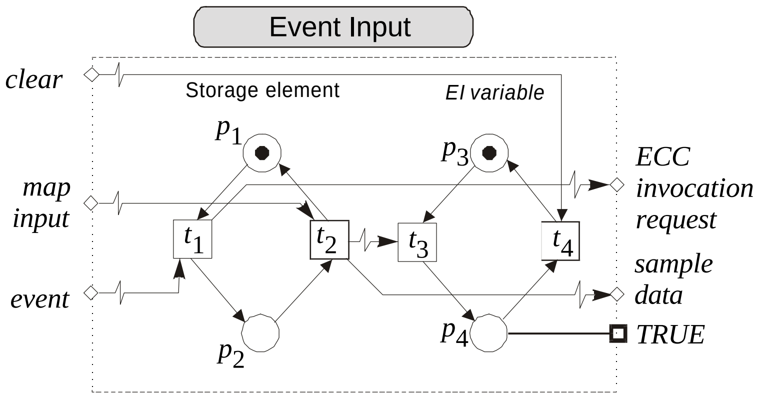


Figure 11. Model of event input variable.

Places in the model have the following meaning:

p1- "Ready" - is ready to detect new event;

p2- "Locked" - event occurred, but its processing by Execution Control has not started yet;
p3- Value of the event input variable is FALSE;
p4- Value of the event input variable is TRUE;

The operations consist of the issuance of event signals "ECC invocation request" at t1 and "Set Event Input variable" (t2,t5). The storage element ensures the correct detection of events and the correct sequence of their processing, which is achieved in co-operation with the further described execution control state machine. Initial state of the model is defined by the marking {p1,p3}. Event input forces transition t1 which changes the state to {p2,p3} and causes the event signal "ECC invocation request". Processing of this signal is performed by resource and will be explained below in the corresponding section.

It implies that only those event inputs are set to TRUE which have been mapped by the execution control. This happens when the execution control starts to process these events. According to the standard, once event input variable is set to TRUE it can be reset only by the execution control, and only in the case if a transition in the execution control chart clears. Until a transition clears all the event input variable remain to be in their state. Event output is also implemented as an NCES module, containing two states and two transitions forced by event signals "Set event output" and "Issue output events".

EXECUTION CONTROL OPERATION STATE MACHINE (ECO SM)

Operation of the Execution Control Chart is described in the IEC1499 by means of the state machine (ECC SM) shown in Figure 12,a with transitions and actions defined as in the table 2.

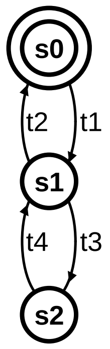


Figure 12.

Operation of the Execution Control Chart is described in the IEC1499 by means of the state machine (ECC SM) shown in Figure 12,a with transitions and actions defined as in the table 2.
Multiple actions, associated with arcs of the automaton, are interpreted as executed sequentially. Thus, actions associated with t1 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.

Transition	Condition	Operations
t1	Invoke ECC (1)	map inputs (2), evaluate transitions (3a,b)
t2	no transition clears	Issue events (4)
t3	a transition clears	Schedule algorithms (5)
t4	Algorithms complete (6)	Clear EI variables(7), set EO variables (8), evaluate transitions (3)

Table 2. Transitions of ECC operation state machine

NOTES TO TABLE 2 (shortened from the original version):

1. This condition is issued by the *resource* at an **implementation-dependent** time after it has received one or more "ECC invocation requests" from one or more of the event input state machines specified in Figure 10 and Table 1.
2. This operation consists of issuing "map input" commands to all of the event input state machines of the function block.
3. (a) This operation 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. (b) *Software tools* may provide means for determining the order in which the EC transitions following an active EC state are to be evaluated.
4. This operation consists, for each event output for which the value of its associated *EO variable* is TRUE (1), of *sampling* each *output variable* associated with the event output by a WITH declaration, then issuing an event at the event output followed by resetting the value of the associated EO variable to FALSE (0). This sampling shall be a *critical operation*.
5. This operation consists of requesting the resource to schedule for execution the algorithms named in the EC actions associated with the active EC state.
6. This condition consists of the completion of execution of all the algorithms associated with the active EC state (always TRUE for an EC state which has zero associated algorithms).
7. 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.
8. This operation consists of setting to TRUE (1) the value of the EO variables named in the EC action blocks of the active EC state.

Basic structure of ECO-SM model 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". We introduce also an additional "transitional" place p_2 in the ECO-SM model, which ensures that variable setting is completed before the transition evaluation. Since the transition from p_2 to p_3 is not forced by any event and the net is assumed to be timed, it fires as soon as p_2 becomes marked. The same role plays the place 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 t4: "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."

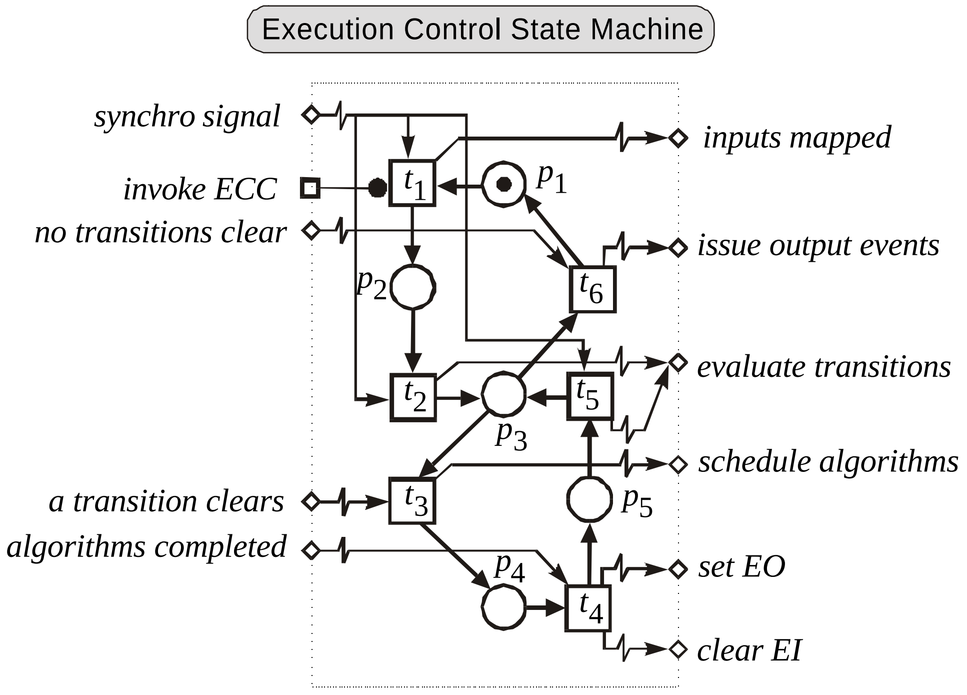


Figure 13. Model of the execution control state machine

To 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 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 include 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 14-a. Desired behaviour 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."

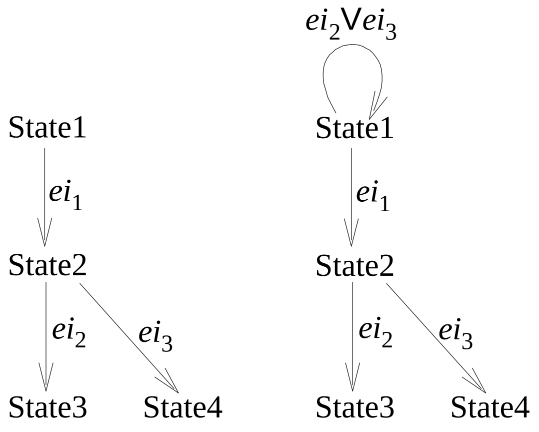
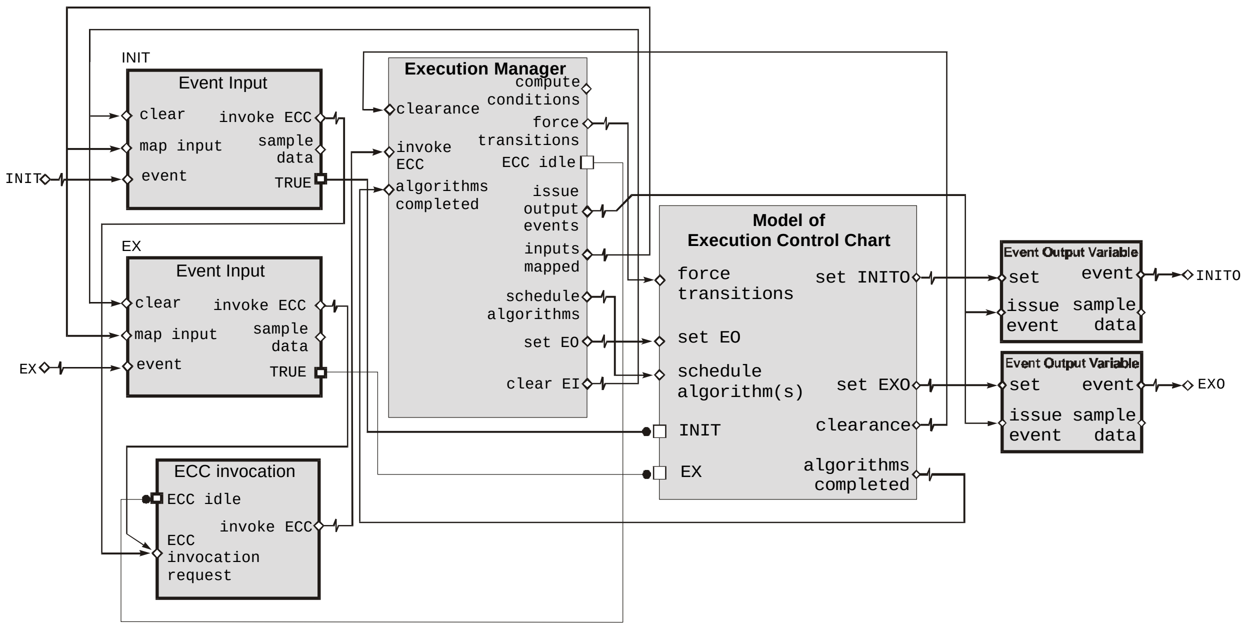


Figure 14. a).ECC which accepts sequence ei3,ei1 as ei1,ei3 b).Corrected ECC

To get the required behaviour, the correct ECC must look like one in Figure 14-b. In this case ei3 will be cleared if occurred at the state 1.



MODELING OF A COMPOSITE BLOCK

MODEL 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,* 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, synchronising 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 resource containing two function blocks is given in Figure 15. Processing of invocation requests is modelled 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 (Figure 13), 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.

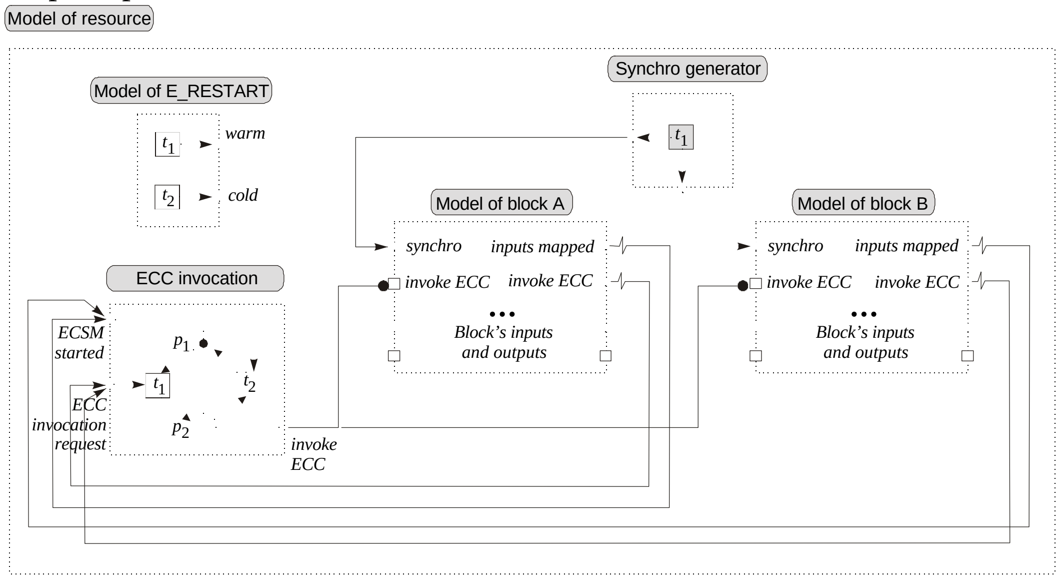


Figure 15. Sketch model of resource containing two function blocks

Scheduling function of the resource is described in the standard as a *function* which selects *algorithms* or *operations* for *execution,* and initiates and terminates such execution, based on the following:
1. the occurrence of events;
2. function block interconnections; and
3. scheduling information such as task scheduling periods and priorities;
4. possible interactions with scheduling functions of other resources.

The working draft of the standard does not specify at the moment the two latter means of providing of the scheduling requirements. That is the reason why in the current models we made the following limitation assumptions:

1. Algorithms have no priorities or task scheduling periods;
2. Resource is capable to execute simultaneously as many algorithms as required;
3. Several copies of the same algorithm can be executed independently;

Synchronisation of the blocks within the resource is required 1) to reflect the synchronous nature of the computing device; and 2) to model possible sequencing of events caused by the fact that some blocks of the application reside in one resource, and some in the other resources. For the latter purpose we introduce the synchroniser, which is a synchronous transition, forcing all the transitions in the model of the blocks, which correspond to the commands executed by the resource.

EXAMPLES

When the interconnected plant/controller system is modelled one has to take in account the following considerations:

1. Model of plant has to reflect asynchronous nature of the real plant. Thus if two units co-exist 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}.
2. Model of the event-driven controller has to provide the 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 behaviour 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.
3. 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 modelled. Consider the sketch example given in Figure 16. The "controller" part of the system consists of three components: Controller 1 which registers the event 1 executing the sequence S_{11},S_{12},S_{13}, and generating the output event 3; 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.
4. 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 “synchroniser” transition of each controller in every state transition.

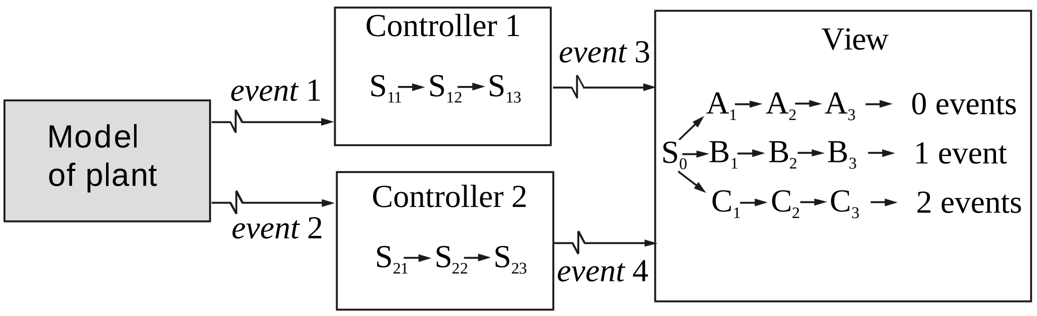

Model of plant
event 1
event 2
Controller 1
$S_{11} \rightarrow S_{12} \rightarrow S_{13}$
Controller 2
$S_{21} \rightarrow S_{22} \rightarrow S_{23}$
event 3
event 4
View
$A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow$ 0 events
$S_0 \rightarrow B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow$ 1 event
$C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ 2 events

Figure 16. Sketch example of an interconnected system

Certainly this behaviour can be somehow modelled 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 the various complications.

REFERENCES

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

[2] Function Blocks for Industrial Process Measurement and Control Systems
International Electro technical Commission, Tech.Comm. 65, Working group 6, Committee draft

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

[4] P.Starke, S.Roch, K.Smidt
Analysing signal-event systems
Humboldt Universitat zu Berlin, Institut fur Informatik

[5] R.S.Sreenivas and B.H.Krogh
On condition/event systems with discrete state realisations

Discrete Event Dynamic Systems: Theory and Applications, 2(1):209-236,1991

[6] V. Vyatkin, H.-M. Hanisch
A modelling approach for verification of IEC1499 function blocks using net condition/event systems
ETFA-99,pp.

[7] V. Vyatkin, H.-M. Hanisch, P. Starke, S. Roch,
Systematic modelling of discrete event systems with signal/event nets and its application to verification of iec1499 function blocks,
Submitted to the special issue "Discrete Event Systems" of IEEE Trans. on Man, Machine and Cybernetics.