Otto-von-Guericke Universität Magdeburg
Fakultät Elektrotechnik
Institut für Automatisierungstechnik (IFAT)

Humboldt-Universität zu Berlin
Institut für Informatik

Modelling and Verification of Execution control of the Function Blocks following to the standard IEC 1499 by means of Net Condition/Event Systems (NCES)

Technical report

Abstract- *This report provides a modeling framework for verification of the event-driven function blocks for industrial process measurement and control systems following to the new international standard IEC1499 . The modeling is based on the representation of the components of the modeled system in the modular way as Net Condition Event Systems. The entire model is obtained connecting the component modules by condition and event arcs into a Signal/Event net. The model retains the original structure of the modeled block. The model in terms of the Signal/Event nets can be exposed to various model checking procedures provided by the net analyzing tools which include analysis of the reachability graph, proof of temporal logic specifications, etc. The modeling approach is illustrated in the paper on two examples of the function blocks, which represent as general real-time applications as well as discrete event controllers. In the latter case the modeling framework includes also the model of the controlled plant. The paper reflects the state-of-the-art in the current work on the verification of the IEC1499 applications and outlines its prospects.*

1 Introduction

An inherent feature of present real-time systems development is their formal verification. Theoretically it is based on the works of E. Clarke [10], J. Ostroff [14, 15], Z. Manna and A. Pnueli [13], R. Alur et al. [7, 8]. In particular, verification of programs for Programmable Logic Controllers (PLC) was studied in [9, 19, 12].

A typical framework for the 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.

The latest trends in the development of discrete control systems are concentrated in the developing new international standard on function blocks for measurement and control systems IEC 1499. In this paper we are trying to come up with suitable models for the function blocks following IEC1499 keeping in mind the requirements stated above. 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.

The remainder of this paper is organized as follows. Section 2 refers to some basic notions of function blocks as defined in IEC 1499. This motivates the modular modeling technique chosen to map function blocks into formal or semi-formal models, which is introduced in Section 3. Section 4 presents a definition of the modeling formalism of Signal/Event nets. In Section 5 we discuss the modeling components required to build a model of a function block. Sections 6,7 provide some examples for modeling and verification of execution control of function blocks. An outlook of further work that needs to be done concludes the paper.

2 Description of IEC 1499

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 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. 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 the head is connected to the event flow, and the body to the data flow. The algorithms included in the block are programmed in terms of IEC1131.

An application which is defined as a collection of interacting function blocks connected by event and data flows, can be distributed over multiple devices which is the significant difference of this new approach in contrast to IEC 1131.

For example, a control application may include a function block, implementing the controller itself, as well as a block responsible for operator interface (displaying the 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 a communication interface of devices where the blocks reside. For instance, the controller may be placed in one device (e.g. 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 the system architecture on the logic level, which includes the notions of system, device, resource, subapplication, 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 the notion of execution control for a single function block, as well as a 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.

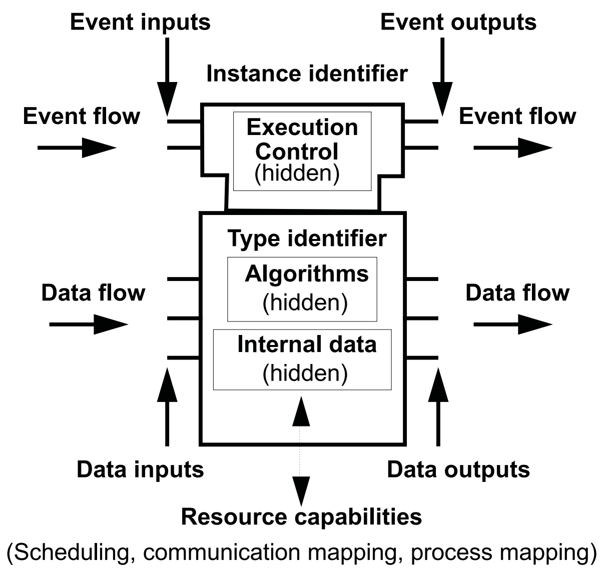


Figure 1: Characteristics of function block.

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. 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.

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.

Farther in the paper we present a couple of examples of the function blocks, following IEC 1499. Thus, Figure 9 shows an example of both the external interface of the block INTEGER-REAL, and its execution control chart.

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.

An inevitable precondition for verifying a controller design which follows the IEC 1499 is a way of modeling which is closely related to the modular way the design is performed. We will outline a modular framework for modeling discrete event systems in a block-diagram way in the following section.

3 Modular Modeling of Discrete Event Systems

Inspired by the idea of Condition/Event systems as defined by Sreenivas and Krogh in 1991 [4] the prior work of some of the authors dealt with providing a modeling formalism which preserves the graphical notation and the non-interleaving semantics of Petri nets and extends them with a clear and concise notion of signal inputs and outputs. These models were called Net Condition Event Systems (abbr. NCES), and they have been applied to a few discrete event system problems, including controller synthesis, verification, and performance analysis and evaluation. A bunch of specific NCES "dialects" have been developed to cover this wide range of problems.

From today's point of view, the general idea which is common to each particular "dialect" is quite simple, namely the way of thinking of and modeling a system as a set of modules with a particular dynamic behavior and their interconnection via signals. This way of modeling is a very intuitive one, and the modules can be pre-tailored and used over and over again. Roughly spoken, the behavior of a module can be described by a Petri net in the classical sense or even by a NCES.

Each module is equipped with inputs and outputs which are of two types:

1. Condition inputs/outputs carrying state information, and
2. Event inputs/outputs carrying state transition information.

This way the over extension clearly reflects the duality of Petri nets, namely the clear distinction between states and states transitions with their own graphical representation, semantics, and formal properties.

An illustrative example of the graphical notation of a module is provided in Figure 2.

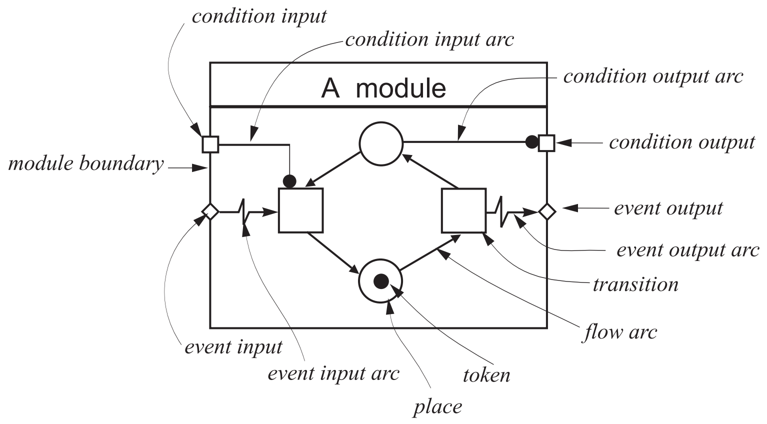


Figure 2: Graphical notation of the module.

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 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 "gluing" inputs of one module with outputs of another module as depicted in Figure 3.

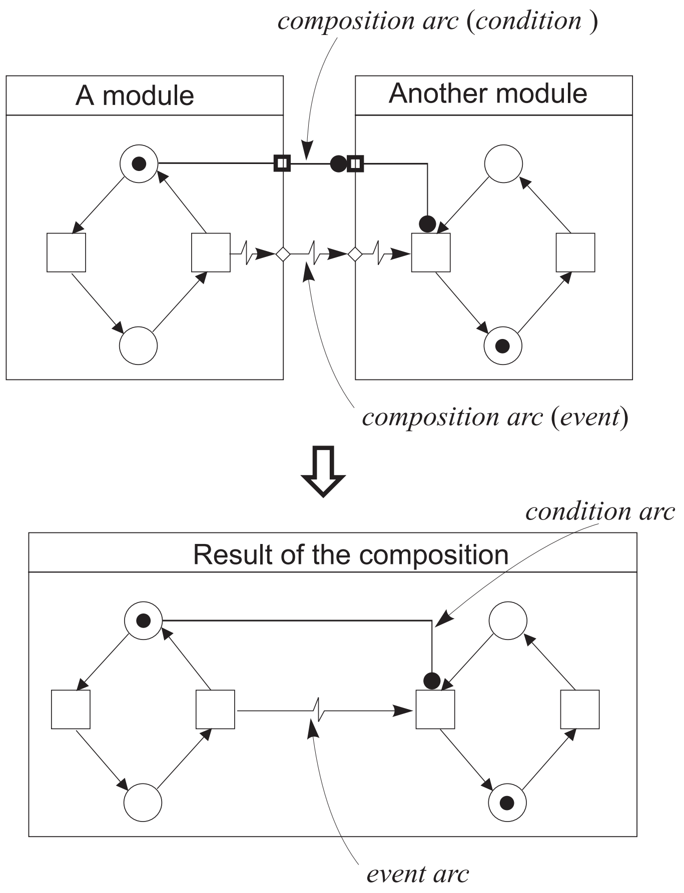


Figure 3: Example of modular composition

This "gluing" is graphically represented by composition arcs. The semantics of these arcs is obvious to any engineer who has ever modeled a system in a block-diagram oriented way. It is further obvious that only inputs and outputs of the same type (conditions or events respectively) can be interconnected. The low part of the Figure 3 shows the result of the composition. One sees that we get two kinds of new arcs, namely condition arcs and event arcs. Their influence on the dynamic behavior is formally described in Section 4. If such a new module is equipped with inputs and outputs, it can also be interconnected over and over again.

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. Some aspects are worth mentioning before we come to the formal definition of the model for an interconnected, autonomous system. These aspects are:

1. We have an intuitive way to model a system. This way closely corresponds to the design techniques as

they provided by IEC 1499.

2. The modeling technique supports a bottom-up modeling as well as top-down strategy. One could start with a set of modules and create a larger model by composing them. On the other hand, one could start with a larger system and could decompose it to a set of subsystems (expressed by modules) and a set of interconnections (expressed by composition arcs).

3. Even after composition, the state of the system is distributed, and the original structure of the modules is preserved. Even removing a module and replacing it by another module with the same input/output interface would be a local operation over the structure of the model. Hence, composition is far less complicated as building the cross product of automata or the interleaving language. This allows us to build models of realistic scale efficiently.

On the other hand, local changes in the behavior of a single module may lead to global changes in the behavior of the interconnected system. Hence, one needs a formal model for expressing and analyzing the global behavior of an interconnected system. Such a formal model is the model of Signal/Event nets as described in the next section.

4 Signal/Event nets

Signal/Event net is an autonomous NCES, i.e an NCES without input and output signals, which can be obtained as a result of composition of several NCES modules.

4.1 Untimed Models

Let P be an arbitrary non-empty set. A mapping $m : P \to \mathbb{N}_0$ is called a *marking of* P (or a *multiset over* P). For $p \in P$, the number $m(p) \in \mathbb{N}_0$ often is referred to as the *number of tokens* on p or as the *multiplicity of* p *in* m.

For markings m and m' (of the same set) we define the *sum* $m + m'$, the *difference* $m - m'$ and the relation $m \leq m'$ pointwise. Moreover, reminding that markings are multisets, we define the *union* $m \cup m'$ and the *intersection* $m \cap m'$ by

$$\begin{aligned} m \cup m'(p) &:= \max(m(p), m'(p)), \\ m \cap m'(p) &:= \min(m(p), m'(p)). \end{aligned}$$

$N = [P, T, F, B, S, M, m_0]$ is a *signal-event net* (*SE*-net for short) iff:

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 flow relation, the set of flow arcs),

4. B is a subset of $P \times T$ (the set of condition arcs),

5. S is a subset of $T \times T$, the irreflexive signal (flow) relation,

6. M is a mapping which attaches a (signal-processing) mode to every transition ($M : T \to \{\boxed{\wedge}, \boxed{\vee}\}$), and, finally,

7. m_0 is a marking of P called the initial marking.

The sets P, T and F, and the mapping m_0 are interpreted in the usual 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 1 token. We consider condition arcs $[p,t]$ as leading 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 t sends a *signal-event* to t'. We assume the signal relation to be irreflexive since it is not meaningful for a transition to send to itself a signal-event.

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 is $\boxed{\vee}$. Consider a transition t which is the target of signal arcs coming from $t_1, \ldots, t_n$, i.e. $[t_1,t], \ldots, [t_n,t] \in S$. If $M(t) = \boxed{\vee}$ then to fire t it is necessary that at least one signal arc $[t_i,t]$ leads a signal-event, i.e. 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 lead 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 t is called *spontaneous*, otherwise *forced*. By *Spont* we denote the set of all spontaneous transitions of N, by *Forc* the set of all forced transitions.

For any t we define the markings t^-, t^+, $\hat{t}$ as follows:

$$t^-(p) \quad := \quad \begin{cases} 1, & \text{if } [p,t] \in F \\ 0, & \text{else} \end{cases} \quad ,$$

$$t^+(p) \quad := \quad \begin{cases} 1, & \text{if } [t,p] \in F \\ 0, & \text{else.} \end{cases}$$

and

$$\hat{t}(p) \quad := \quad \begin{cases} 1, & \text{if } [p,t] \in B \\ 0, & \text{else.} \end{cases}$$

For any subset $s \subseteq T$ the markings s^- resp. s^+ are the sum of the markings t^- resp. t^+ for $t \in s$, and, $\hat{s}$ is the union of the markings $\hat{t}$ for $t \in s$.

Given a marking m, we say that t has *token-concession* at m iff every (pre-)place p such that $[p,t]$ is in F holds at m at least one token, i.e. $t^- \leq m$.

A transition t is said to be *enabled at the marking* m iff t has token-concession, its conditions are satisfied, i.e. $\hat{t} \leq m$, and, (if St is not empty) the necessary signal-events are present.

SE-systems are executed in steps, i.e. sets of transitions fire simultaneously. The *firing rule* says, roughly speaking, that executable steps are formed by first picking up a nonempty set of enabled spontaneous transitions 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 a spontaneous transition. To make this more precise we define the *signal-completeness* of transition sets inductively:

Basis: Every subset $s \subseteq Spont$ is *signal-complete*.

Step: If $s \subseteq T$ is *signal-complete*, $t \in Forc$ and ($M(t) = \boxed{\vee}$, and $St \cap s \neq \emptyset$) or ($M(t) = \boxed{\wedge}$, and $St \subseteq s$) then $s \cup \{t\}$ is *signal-complete*.

Obviously, the empty set is signal-complete and $\emptyset$ is the only signal-complete set containing no spontaneous transition. A signal-complete set of transitions may fire simultaneously as far as signal-events are concerned.

A subset $s \subseteq T$ is said to be a *step of* N iff

1. $s \cap Spont \neq \emptyset$

   (i.e. there is at least one spontaneous transition in s), and

2. s is signal-complete

   (i.e. all necessary signal-events will occur).

A step s of N is called *enabled at the marking* m iff

3. $\hat{s} \leq m$

   (i.e. the conditions of all $t \in s$ are satisfied), and

4. $s^- \leq m$

   (i.e. the transitions in s are concurrently enabled w.r.t. tokens).

A step s of N is said to be *executable at the marking* m iff s is enabled at m and there is no forced transition $t \in Forc$ such that $s \cup \{t\}$ also satisfies 1-4 (i.e. s is maximal with respect to inclusion of forced transitions.)

A forced transition t with $M(t) = \boxed{\wedge}$ appears in an enabled step only if it receives signals from all its signal sources. Otherwise, a forced transition t with $M(t) = \boxed{\vee}$ appears in an enabled step if it receives a signal from at least one of its signal sources.

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 reachability relation is defined as usual: let $R_N(m)$ denote the set of all markings m' such that a finite sequence of executable steps leads from m to m'.

4.2 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 refered to as *permeability* interval.

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 $N = [P, T, F, B, S, M]$ 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. Let $[m,u]$ be a state.

A step s of N is said to be *enabled at the state* $[m,u]$ of TN iff

3. $\hat{s} \leq m$

4. $s^- \leq m$ and for every pre-place p of $t \in s$ it holds $eft(p,t) \leq u(p) \leq lft(p,t)$.

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*.

The state $[m,u]$ of an arc-timed signal-event net may change not only by execution of a step but also 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 step is enabled or can become enabled by elapsing of time 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.

Let $[m,u]$ be a non-dead state. Following the *earliest firing rule* we call a step s to be *executable at the state* $[m,u]$ iff s is enabled after elapsing of $D(m,u)$ time units and s is not contained properly in a step s' which is enabled after elapsing of $D(m,u)$ time units.

The execution of an executable step s at the state $[m,u]$ then is done by first elapsing $D(m,u)$ time units and then firing s. The state $[m',u']$ reached by the execution of s is determined by
$m' = m - s^- + s^+$,

$$u'(p) \quad := \quad \begin{cases} u(p)+D(m,u), & \text{if } m(p) > 0 \wedge m'(p) > 0 \wedge p \notin (Fs \cup sF), \\ 0, & \text{else.} \end{cases}$$

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

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

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.

The following is worth mentioning as the timed models concerned:

1. In a timed NCES if a pre-arc has no explicitly designated permeability interval, it is assumed to be $[0,\omega]$.

2. As follows from the definitions above, a spontaneous transition enabled by marking must fire as soon as it becomes enabled by time.

4.3 Timed Computation Tree Logic

In their paper [5] Emerson, Mok, Sistla and Srinivasan propose an extension of the *Computation Tree Logic* CTL that allows the melding of qualitative temporal assertions together with real-time constraints. They call their logic *Real-Time Computation Tree Logic* and they show that several practically useful correctness properties can be expressed in RTCTL.

The extension essentially consists in attaching a number k to the modalities which serves as a time bound, e.g. $EF_k\varphi$ is satisfied by the state s_0 iff there is fullpath starting at s_0 such that the first state which satifies the formula φ is reached after at most k units of time.

We here go one step further and use intervals $[l, h]$ with $0 \le l \le h \le \omega$ as time constraints, but attach them only to the modalities X, F and U. Hence, a formula from *Timed Computation Tree Logic* TCTL is obtained from a CTL-formula by attaching intervals to some of these modalities. 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 such that
3. $\forall z(z \in Z \rightarrow \exists z'(z' \in Z \ \wedge \ z \Rightarrow z'))$, i.e. no state is dead,
4. $D : Z \rightarrow \mathbb{N}_0$ is the state delay.

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 elaps at z before a step can be executed.

A *fullpath* σ in Σ is an infinite sequence $(z_i), i \in \mathbb{N}_0$ of states such that $z_i \Rightarrow z_{i+1}$ holds for all $i \in \mathbb{N}_0$. 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.

Let φ and ψ be arbitrary formulas of TCTL and z be a state. Then we put:

$z \models EX[l, h]\varphi$ iff $\exists z'(z \Rightarrow z' \ \wedge \ l \le D(z_0) \le h)$,
$z \models AX[l, h]\varphi$ iff $\forall z'(z \Rightarrow z' \ \rightarrow \ l \le D(z_0) \le h)$,

$z \models EF[l, h]\varphi$ iff there exists a fullpath σ starting at $z_0 = z$ and a number
$j \in \mathbb{N}_0$ such that $z_j \models \varphi$ and $l \le D(\sigma, z_j) \le h$;
$z \models AF[l, h]\varphi$ iff for all fullpathes σ starting at $z_0 = z$ there exists a number
$j \in \mathbb{N}_0$ such that $z_j \models \varphi$ and $l \le D(\sigma, z_j) \le h$;

$z \models E[\varphi U[l, h]\psi]$ iff there exists a fullpath σ starting at $z_0 = z$ and a number
$j \in \mathbb{N}_0$ such that $z_j \models \psi$, $l \le D(\sigma, z_j) \le h$, and,
$\forall k(k < j \ \rightarrow \ z_k \models \varphi)$;
$z \models A[\varphi U[l, h]\psi]$ iff for all fullpathes σ starting at $z_0 = z$ there exists a
number $j \in \mathbb{N}_0$ such that $z_j \models \psi$, $l \le D(\sigma, z_j) \le h$, and,
$\forall k(k < j \ \rightarrow \ z_k \models \varphi)$.

Some examples:

1. A state z is called a time-deadlock iff all states z' which are reachable from z have the delay $D(z') = 0$. This is expressed by
$$z \models AGEX[0,0]\top \text{ or } z \models \neg EF[1,\omega]\top.$$

2. Let φ be a formula which is satisfied exactly by the state z. From the state z_0 in any case the state z is reached after at most d units of time: $z_0 \models AF[0,d]\varphi$.

3. Let φ be a formula which is satified exactly for states $z = [m,u]$ with $m(p_1) = 3$ and let ψ be satisfied iff $m(p_2) = 4$. Then
$$z \models \neg EF(\varphi \ \wedge \ \neg AF[0,d]\psi)$$
holds from any state z where p_1 has three tokens within at most d units of time a state is reached where p_2 has four tokens.

We will not use all the features of Signal/Event nets here (for example arc weights and multiple tokens on places), but the interested reader will see the essential new ingredients, namely the event arcs that synchronize transitions, are part of the formalism of Signal/Event nets. Therefore, Signal/Event nets and appropriate analysis and model checking techniques provide a formal basis to formally verify models which have been built by using the NCES methodology.

The following section presents application of the Signal/Event nets to the modeling of basic components of the function blocks following IEC1499. The following restrictions to the Signal/Event nets have been applied.

1. The nets are timed, though in the most components of the models (except for the model of uncontrolled plant behavior in Section 7) the timing is not shown explicitly, so that the permeability intervals are assumed to be $[0,\omega]$. This also refers to the specification language, which at the current stage of the work is mostly restricted to the CTL rather than RTCTL.

2. All the transitions have the signal processing mode $M(t) = \boxed{\wedge}$;

5 Modeling and verification of control systems following IEC 1499

5.1 General considerations

Two questions are important when a discrete control system is modeled:

1. How to build a model more or less adequate to the behavior of the system under consideration?

2. How to utilize the model to analyze the behavior of the object and answer the questions of interest?

Description of a discrete control system includes a controller, presented in some formalism, and a specification of the controlled plant. As IEC 1499 presents new formalisms for the controller specification, the following verification issues can be of interest as applied to the discrete control systems:

- Verification of the formalisms defined by the IEC 1499 itself;

- Verification of arbitrary function blocks and applications following IEC1499, in particular controllers and their parts;

- Verification of models of uncontrolled plant behavior;

- Verification of the control systems consisting of both controller and plant models;

In the case when a standalone application is verified, its inputs are modeled using spontaneous transitions of Petri nets. Evolution of the model is determined by all possible combinations of the spontaneous transitions firing. In other words, each path in the reachability graph corresponds to a certain combination of the spontaneous transitions firing. Thus, the analysis of the application behavior is based on the analysis of the reachability graph which contains all possible states where changes of inputs may lead to. Dimension of the space of reachable states, which may exponentially depend on the number of spontaneous transitions in the model, may become so large that analysis would be impossible. However when controller is modeled together with plant, models of plants usually restrict the space of reachable states.

A possible modeling framework might include not only the model of a function block (say a controller), but also a 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 a 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. This problem, however can be overcome using *event generator* module, as we will show further.

As for the formal verification engine, the models are supported by the tool SESA developed at Humboldt University of Berlin [11], and by the simulator, developed in Magdeburg University. SESA allows to perform quite sophisticated analysis of models, 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 model and its processing, such as finding fullpaths 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 model's evolution, expressed in the Timed Computation Tree Logic. Atoms of the TCTL formulas are predicates over the marking of the net which have the following general form $low \leq m(p_i) \leq high$, being TRUE iff marking of the place p_i is within the low and $high$ bounds. In this paper we deal only with the nets having 0/1 marking, so for simplicity we use the notation p_i equivalent to $m(p_i) = 1$.

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).

5.2 Modeling of IEC 1499 components

The hierarchy of structures provided in the standard implies the corresponding way of modeling - we 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.

As for 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 a 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).

5.3 Modeling of Input and 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 element which exhibits the behavior defined by the state machine in Figure 4,a. Its transition arcs are marked both with conditions of transitions and with operations, executed upon the transitions as listed in the following table:

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

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.

The NCES model of event input is shown in Fig.4,b. 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 $t_5 - t_6$).

Places in the model have the following meaning:

p_1	-	"Ready" - is ready to detect new event;
p_2	-	"Locked" - event occured, but its processing by Execution Control has not started yet;
p_3	-	Value of the event input variable is FALSE;
p_4	-	Value of the event input variable is TRUE;

The operations consist of the issuance of event signals "ECC invocation request" at t_1 and "Set Event Input variable" (t_2, t_5). The storage element ensures the correct detection of events and the correct sequence of their processing, which is achieved in cooperation with the further described execution control state machine. Initial state of the model is defined by the marking $\{p_1, p_3\}$. Event input forces t_1 which changes the state to $\{p_2, p_3\}$ and causes the event signal "ECC invocation request".

As the standard sets, processing of the invocation requests is performed by resource. 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. This behavior is modeled by the NCES shown in Figure 4,c, which sets a Boolean flag (token in the place p_2) after at least one ECC invocation request occured when the execution control is busy processing another event. When the execution control becomes idle, t_2 fires and invokes the execution control again. Once processing of the event is started by the execution control (as will be shown later in Figure 6,b), it issues the

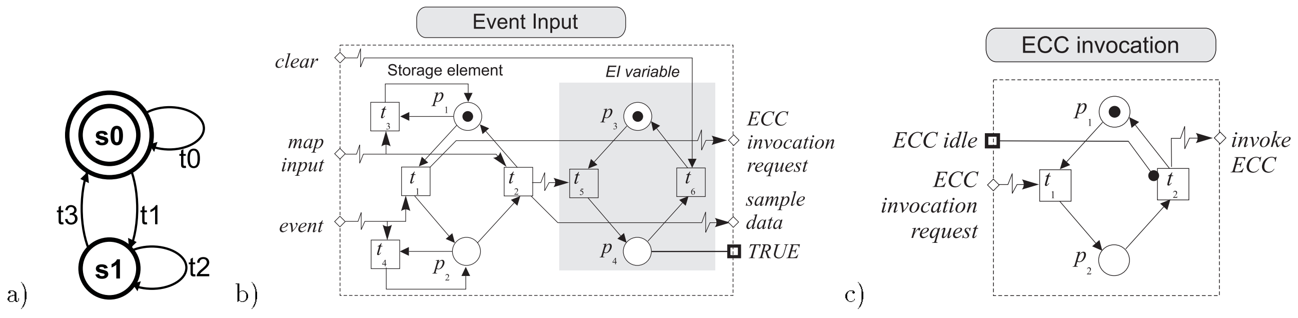


Figure 4: a)Event input storage element state machine; b) NCES model of event input including EI variable; c) Model of the ECC invocation processing by resource.

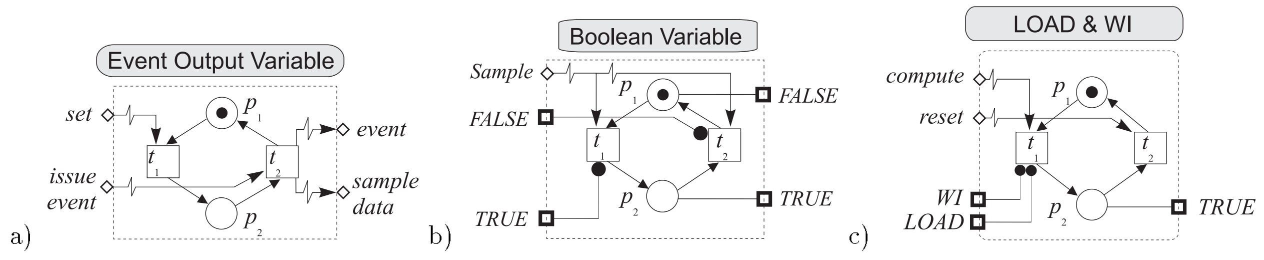


Figure 5: a)Event output variable model; b)Boolean variable model; c)NCES implementation of the Boolean expression: $LOAD \wedge WI$.

signal "Input mapped" to the models of event input, which forces t_2, returns a token from p_2 to p_1, and then forces t_5.

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 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" as shown in Figure 5,a.

5.4 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 6,a with transitions and actions defined as in the table:

Transition	Condition	Operation
t1	invoke ECC	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

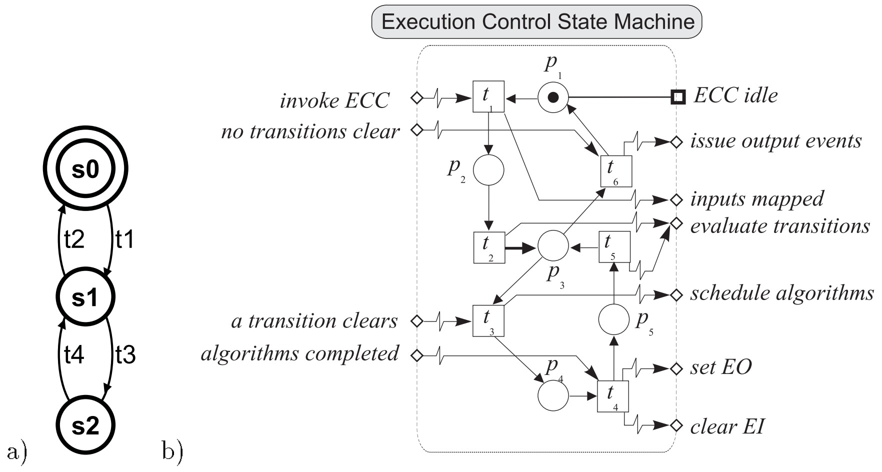


Figure 6: a)State machine of execution control operation and b) its NCES model.

Behavior of the ECO SM 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 $t1$ is driven by the event "Invoke ECC" and has the following associated operations:

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.

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.

5.5 Model of Execution Control Chart

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".

Language of ECC description 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 NCES model which reflects the structure of the initial ECC, where each state of the ECC is mapped in to a place in the model, each transition of the ECC transformed into a correspondingly connected NCES transition. Besides additional modules are provided to model actions and output signal issuance of each state, as well as the modules modeling Boolean conditions of the transition. The place in the model of ECC which corresponds to the initial state has initally one token, while the other places are empty. Thus, the model exhibits a non-deterministic behavior in case if a state in ECC has more than one successor. This allows to analyze all possible scenarios of the ECC evaluation, though the standard states that the latter might be implementation dependent. In this case our model should be properly adjusted according to the particular evaluation discipline.

Evaluation of the ECC transitions is controlled by the execution control state machine which however requires either of the signals "A transition clears" or "No transitions clear" upon the evaluation. The latter are provided by the module named Transition Evaluation Monitor (further abbr. TEM).

5.5.1 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.

Modeling of the ECC transitions includes modeling of Boolean condition computation, and modeling of the explained above evaluation discipline. Computation of Boolean expressions can be modeled to the various extent of detail, dependent on how precise modeling is required. For example, NCES implementation of Boolean expressions up to the level of machine commands (IEC1131-3 Instruction List) was in detail considered in [3]. If less precise implementation would be enough, then each non-trivial Boolean condition can be modeled by a single NCES module, having two places, corresponding to the FALSE and TRUE values of the condition. Initially the value is assumed to be FALSE. Transition from FALSE to TRUE is driven by the event signal "Evaluate conditions". Thus, computation of all conditions takes only one (multi-) transition. The place, corresponding to the TRUE value of the condition is connected to its corresponding transitions by a condition arc. It is essential, therefore, to attempt to fire transitions only after the computation of conditions has been completed. Example of a Boolean condition implementation is shown in Figure 5,c.

A transition has one forcing event input "Force transitions", which is issued by TEM, and an event output, which is connected to the "Clearance" input of the TEM.

5.6 Transition evaluation monitor

The main purpose of TEM is to detect a situation when no ECC transition clears in response to the forcing signal "Evaluate transitions" and to issue the corresponding signal. Structure of the TEM is shown in detail in Figure 7,a. Places in the TEM have the following meaning:

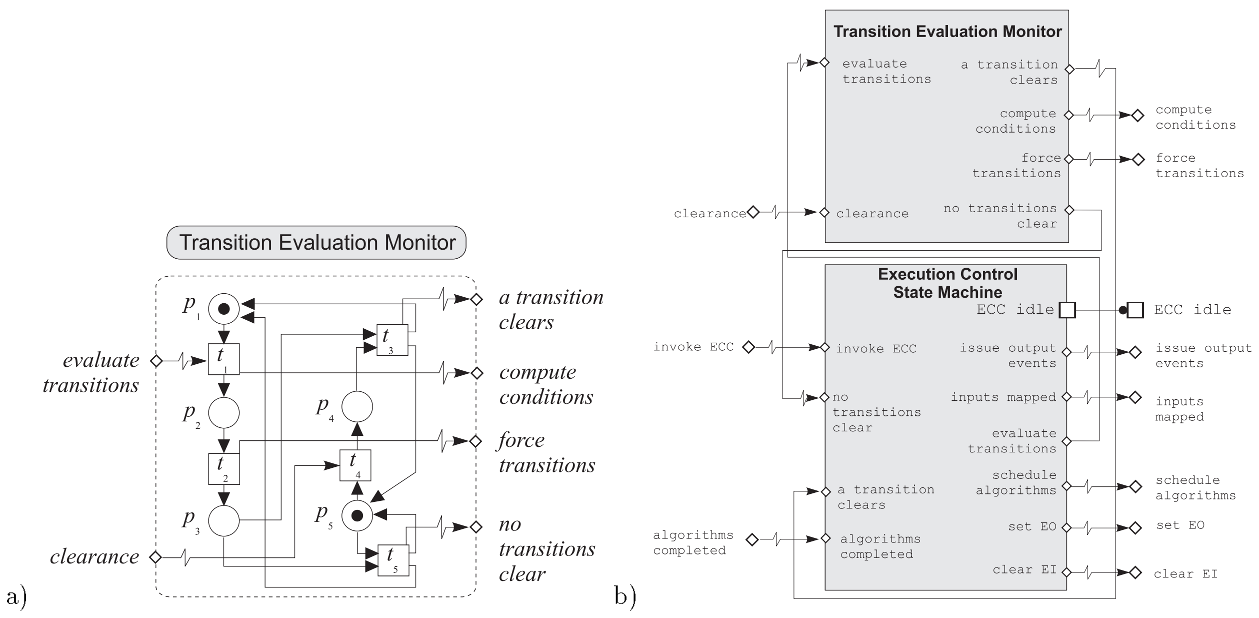


Figure 7: NCES implementation of the Transition Evaluation Monitor a) and Execution manager b)

p_1 - ECC is ready to "Evaluate transitions"

p_2 - Boolean conditions of transitions have been evaluated;

p_3 - Transitions attempted to be fired;

p_4 - A transition cleared;

p_5 - No transition cleared so far;

Transitions in the TEM have the following notation:

t_1 - TEM turns into evaluation mode, compute conditions;

t_2 - Transitions are forced to fire;

t_3 - Transition clears, return to the initial state;

t_4 - Clearance is detected;

t_5 - No clearance of transition is proved;

TEM starts its operation driven by the "Evaluate transitions" event, issued by the EC SM. This event fires the transition t_1 which moves token from p_1 to p_2 and issues the signal "Evaluate conditions". In the next state transition t_2 becomes enabled and fires issuing the event signal "Force transitions", which is connected to every ECC transition.

Every such a transition has a single output event link, all of which are merged at the input "Clearance" of TEM (transition t_4). In case if any of the ECC transitions clears, it forces t_4 to fire and moves a token from p_5 to p_4. It occurs simultaneously with the token moving from p_2 to p_3.

Thus, in the next state places p_3 and p_4 is marked and t_3 becomes enabled and is forced to fire since it is a spontaneous one. Therefore, it fires issuing event $TransitionClears$ and returning tokens to the initially marked places p_1 and p_5.

Otherwise, if the signal "Force transitions" causes no subsequent firing of ECC transitions, a token moves from p_2 to p_3, but another token remains in p_4. It will cause t_4 to fire and issue the signal "No transitions clear".

Since the TEM and ECO SM together provide the execution control of the ECC we combine them in the single module denoted as "Execution Manager" as shown in Figure 7,b.

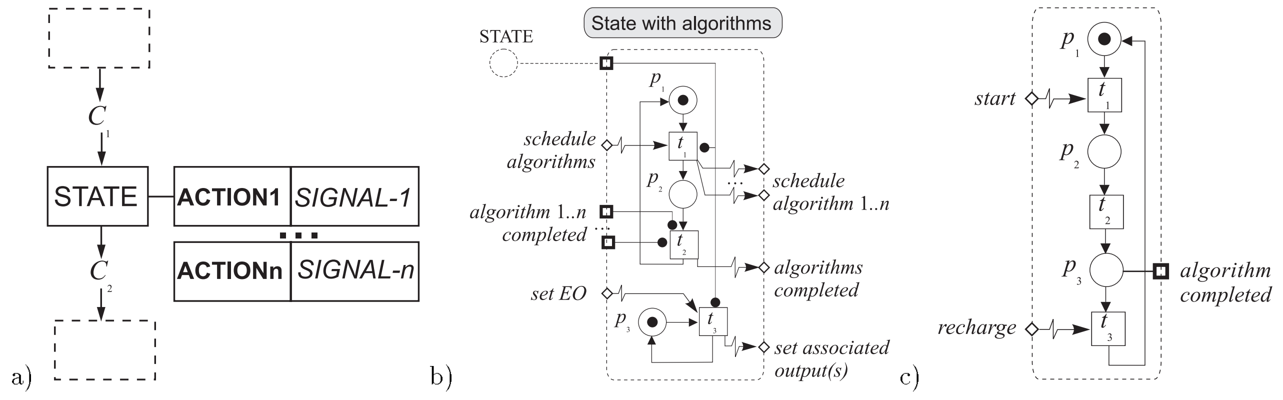


Figure 8: Example of ECC state with n actions (a), its NCES implementation (b), and model of an algorithm.

5.6.1 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.

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.

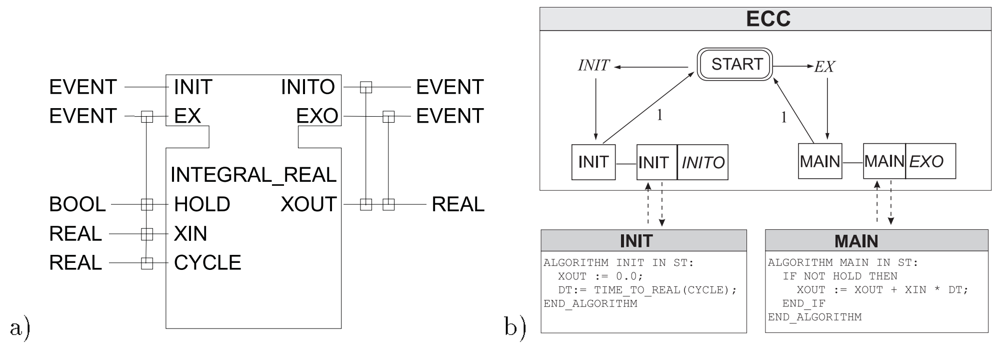
A model of such a state is shown in Figure 8-a,b. Models of each action can be further detailed using definition of its particular algorithm. In the NCES in Figure 8-b it is assumed, that conditions "ACTION-i completed" are issued by the corresponding NCES models of algorithms. If a state has no associated actions then just no incoming condition arcs would be attached to the transition t_2. If detailed models of algorithms are not available, they can be substituted by a simple "one-transition" models as in Figure 8,c, having the states: "Algorithm idle", "Algorithm active" and "Algorithm completed", defined by the marking of places $p_1, p2, p_3$ respectively. Transitions (t_1) from the "idle" to "active" is forced by event signal "start" which is connected to the "Schedule algorithm-i" of the state model. Transition from the active to the completed state is modeled by the spontaneous transition t_2, so the active state lasts no longer than one state of the model. Otherwise, a permeability interval with the delay, to model the execution of algorithm can be assigned to the arc (p_2, t_2). Transition t_3 returns the model to the "idle" state, driven by the "recharge" signal which can be connected to the "Algorithms completed" output of the state model.

6 Modeling and verification of IEC 1499 application

6.1 Building the model

First we illustrate the modeling of IEC1499 applications using a simple example of function block INTEGRAL-REAL which is borrowed from the draft of IEC 1499 (Table 2.2.1 and Annex H). Type declaration of the block is given in Figure 9,a, and Figure 9,b shows its Execution Control Chart and internal algorithms. Functionality of the block is quite simple - it integrates the function given by the value of the input XIN (of type REAL) driven by the event input EX, and provides 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



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

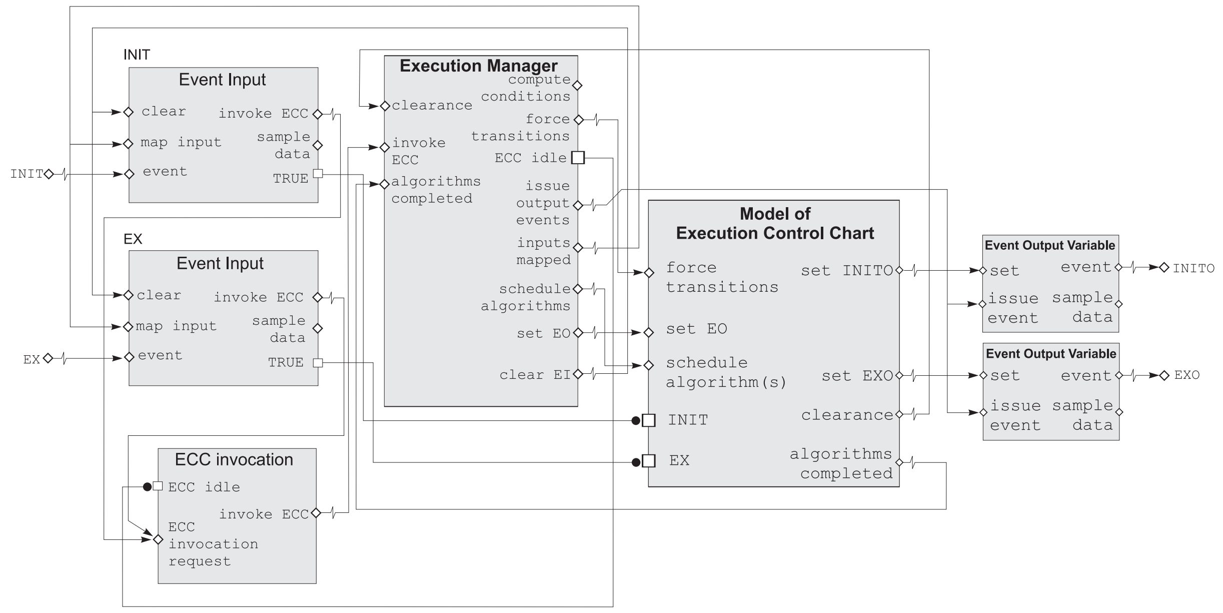


Figure 10: Modular NCES model of the INTEGRAL-REAL function block.

blocks to the model and check the general correctness of our modeling approaches, especially an adequate understanding of the execution rules, provided by the standard as well as correctness of the latter.

Again, we would like to point out that our modeling is quite limited. Thus we do not pursue the goal of checking how correct is the computed result (XOUT). The only thing we would be able to do is to make sure that execution logic of the block is correct.

The model is built from the modules described above, which are interconnected by means of event and condition arcs as shown in Figure 10. Connection of several function blocks in a composite function block is also realized in the similar way.

Model of the execution control chart is composed of the ECC model itself, models of states and models of algorithms INIT and MAIN. In detail it is presented in Figure 11. 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 the state INIT with the call of INIT algorithm. Module p_8, t_9 is responsible for the issuance 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 action is associated with a state (like in case of the START state), the corresponding "Algorithm completed" condition

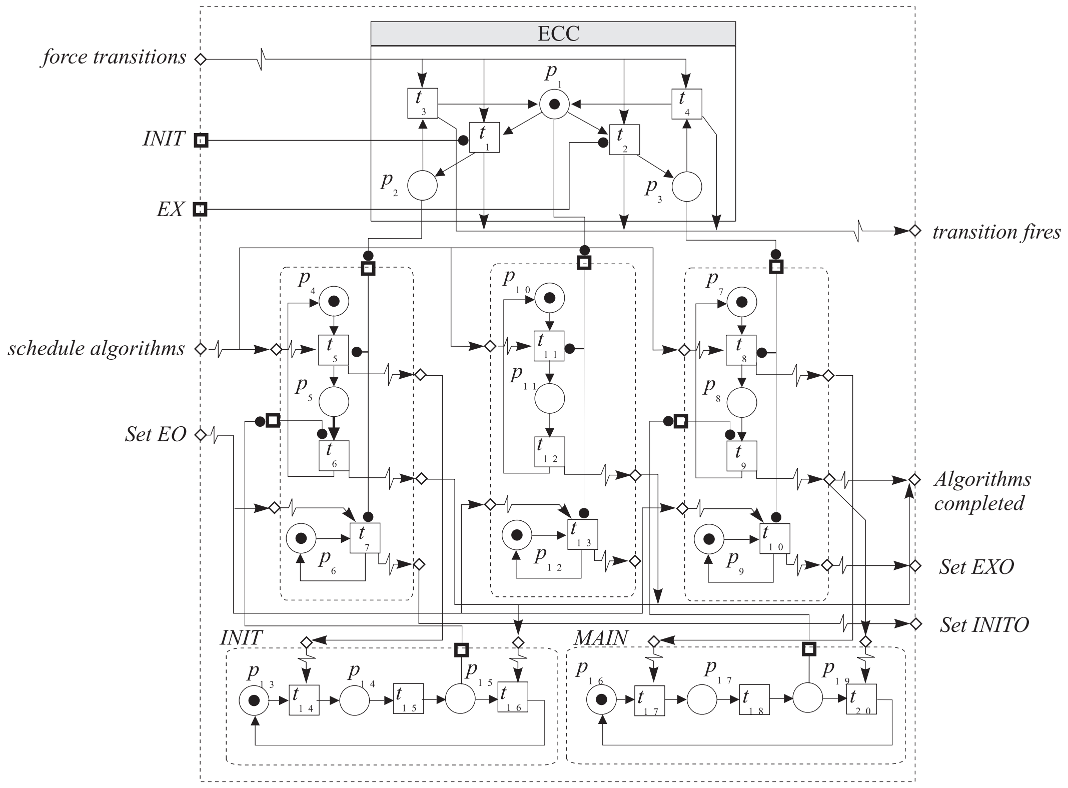


Figure 11: NCES model of the Execution Control Chart of the INTEGRAL-REAL function block.

is always true. This is modeled by the module $p_{11} - p_{13}, t_{12} - t_{14}$ which issues the signal "Algorithms complete" provided the signal "Schedule algorithms". The algorithms INIT and MAIN are modeled as the "one transition" automata in modules $p_{13} - p_{15}$, and $p_{16} - p_{19}$ correspondingly. The resulting model of the INTEGRAL-REAL function block in detail is shown in Figure 12.

Execution control state machine is sensitive to the event inputs when it is in the "idle" state $S0$ which in the model we determine by predicate $P_{idle}^{ECC} = p_{14}$. If an event occurs when the execution control is sensitive, its processing begins immediately, otherwise it is postponed until the state $S0$ is reached again.

Due to the "cooperative work" of event input and execution control state machines the model can be either in "Signal-ready" or in "Signal-locked" state. We denote this as predicates P_{ready}^{signal} and P_{locked}^{signal}. In our model of INTEGRAL-REAL $P_{ready}^{INIT} = p_1$, $P_{ready}^{EX} = p_5$, $P_{locked}^{INIT} = p_2$, and $P_{locked}^{EX} = p_6$. If the execution control state machine comes to the idle state when some signals are locked, they will be mapped into the corresponding EI variables and processing of the locked event inputs will start.

6.2 Analysis

Verification of IEC1499 real-time applications is a broad issue to be covered in the limited space of this paper. Besides, the work on the verification is still underway, so we illustrate just a few opportunities provided by the NCES modeling and supporting tools.

When an event-driven function block is verified, the following questions related to its execution logic might be of interest for the developer of the block:

1. Make sure that execution of the block terminates correctly on all possible combinations of driving inputs;

2. Estimate the response time of the block, measured either in number of transitions required to get the

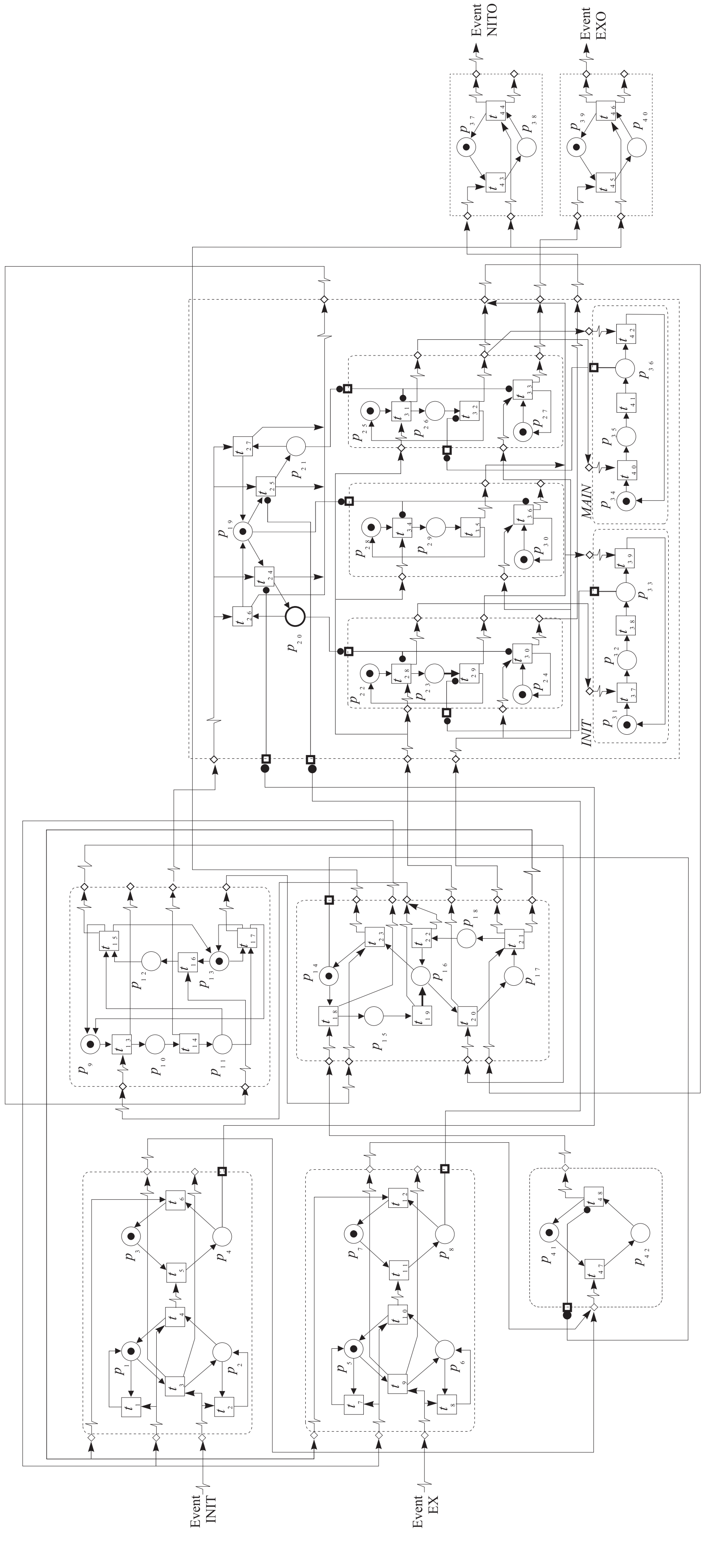


Figure 12: Detailed NCES model of the INTEGRAL-REAL function block.

output given the input, or in time units, when more detailed information on timing is both required and provided;

In context of this work we call this as a verification of "computational" properties as opposite to the verification of technological properties, i.e. those in control applications which directly related to the behavior of the controlled plant.

For the purposes of the computational verification we dip the model into the "verification framework", which consists of the generator of events and bi-stables for each input and output as shown in Figure 13. The bi-stables for events are required to express the fact of event also by a predicate, i.e. via marking of the places. Thus event input INIT is denoted by predicate $P^{INIT} = p_{i1-2}$, event input EX is by predicate $P^{EX} = p_{i2-2}$, event output INITO by predicate $P^{INITO} = p_{o1-2}$, and event output EXO by predicate $P^{EXO} = p_{o2-2}$.

The generator provides "testing" of the model on all possible combinations of the input signals. The model extended in such a way is exposed then to the NCES analyzing tools, first of all to SESA. Using SESA it is possible to built up the reachability graph of the model. Using the reachability graph we can, for example, find and analyse dead states (if any), and check the validity of properties, such as "Prove that for all trajectories input signal 1 is followed by the output signal 1", etc. For the model of the INTEGRAL-REAL function block we found that its reachability graph consists of about 600 states and there are no terminal (or dead) states. However many loops in the reachability graph require more thorough check of the behaviour, which is possible by means of proving temporal logic properties.

The following properties in our opinion would help to ensure the correct behavior of a function block:

6.2.1 Guaranteed response

Each event input eventually is followed by the corresponding event output. Such a sequence of states is called the correct event processing sequence. For example, consider the event input INIT. We are going to check whether every occurrence of the INIT is eventually followed by the event output INITO. We are checking the following temporal statement:

$$AG(P^{INIT} \rightarrow EFP^{INITO}),$$

which means: in every state occurence of INIT is eventually followed by the INITO output.

This formula however is proved in general to be FALSE. The subsequent analysis of the counter examples revealed that this is due to the situation when the event INIT occurs simultaneously with the other event input EX. In this case the model might call either INIT or MAIN algorithm, which is followed by the corresponding event output (either INITO or EXO). The other event input gets lost in such circumstances.

6.2.2 Return to the sensitive state

This property is formulated as follows: from every reachable state every fullpath eventually comes to the state characterized by the predicate P^{ECC}_{idle}. We proved this statement checking the temporal formula $AGAFP^{ECC}_{idle}$.

6.2.3 Event processing always terminates by an event output

If the ECC is not idle that means it is processing some event. We prove that when the event processing is ended (i.e. next time P^{ECC}_{idle} becomes true again), one of either event outputs is already occurred:

$$AG(\neg P^{ECC}_{idle} \rightarrow E[(P^{INITO} \vee P^{EXO})BP^{ECC}_{idle}]).$$

This formula is also proved to be TRUE.

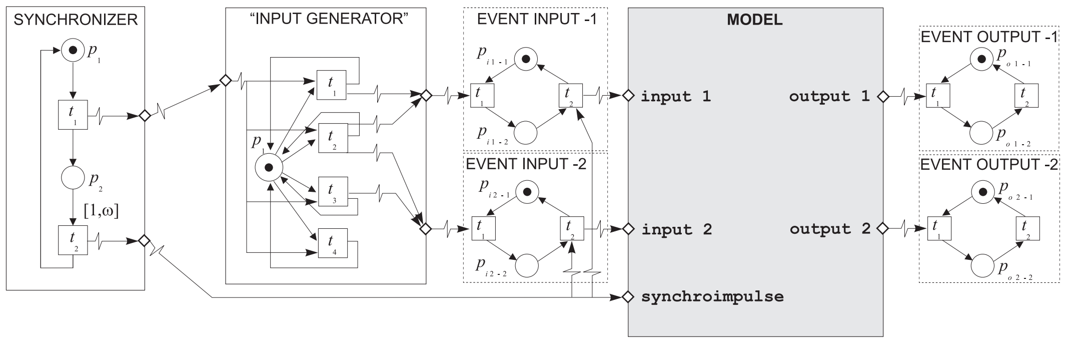


Figure 13: Framework for verification includes generator of events and bi-stables for each input and output.

6.2.4 Correct processing of simultaneous event inputs

A simultaneous occurrence of two or more event inputs might require the usage of several algorithms which in turn would ask several transitions to be evaluated using the values of corresponding event input variables. Behavior of the block in such cases must be explicitly defined. The standard, however, does not say much about the issue. To clarify, how a function block would behave in such a case, we have studied consequences of the simultaneous occurrence of two event inputs INIT and EXO. The analysis revealed that such an input is never followed by the simultaneous output of INITO and EXO. As applied to the INTEGRAL-REAL, this is obviously correct behavior.

In general, regardless of how many inputs triggered simultaneously, only one transition of ECC with non-constant condition would fire. As a matter of fact, simultaneously means during the time when execution control is busy processing prior events. This behavior is due to the fact that processing of an event is ended by the corresponding "Clear EI variables" signal which also erases traces of all other events.

6.2.5 Measurements

Rough estimation of the block timing, for example in terms of number of transitions, required to get the output for a corresponding input is also possible using analysis with SESA. It allows, for example, to find shortest or longest path between event-input and corresponding event-output states in the reachability graph.

More precise estimation of the timing is possible if we assign realistic values of delays for the operations in all parts of the model by means of the timed arcs.

7 Modeling and verification of a simple control system

Consider more sophisticated example of IEC1499 application: controller of a simplified transfer stage. This example originally is borrowed from the web site of the Allen-Bradley Company dedicated to the IEC 1499, where it serves as an illustration of programming within IEC1499 framework. The original example consists of controller, model of the object and human-machine interface components.

Here we use a slightly modified controller part of the example. We verify the correctness of the controller function block using an TNCES model of the plant which has behavior similar to the specified in the original. Using our modeling and verification technique, we spotted some faults in the original controller which could lead to an incorrect behavior of the object.

7.1 Description of the object and specification of the desired behavior

The transfer stage (TS) is meant for transferring workpieces between two positions, denoted as HOME and END. TS consists of STATION, where the workpiece rests, and turning mechanism which moves the station between the positions. The transfer stage is connected to the INPUT TRAY, which is the source of workpieces, and to the "OUTPUT TRAY", i.e. the place where workpieces have to be delivered.

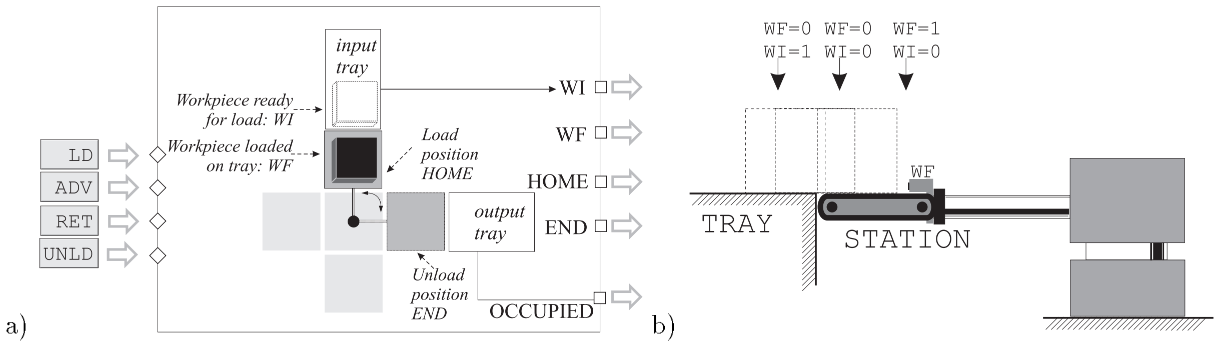


Figure 14: Transfer stage

7.1.1 Sensors (condition and events)

HOME - station is in the loading position;

END - station is in the unloading position;

WI - workpiece is in the input position (HOME). Sensor is ON from the position when the workpiece is ready to be loaded until the workpiece is completely loaded on the station of TS;

WF - workpiece is on the station of TS. Is TRUE when the workpiece is completely loaded on the station. WI and WF are not overlapped, i.e. while a workpiece moves from the input tray to the station of the transfer stage, the following sequence of states unfolds:

1. $WI = 1$ and $WF = 0$ - the workpiece is on the input tray;
2. $WI = 0$ and $WF = 0$ - the workpiece moves from the input tray to the stage;
3. $WI = 0$ and $WF = 1$ - the workpiece is on the stage;

OCCUPIED - occupancy status of the output tray;

7.1.2 Actors

Actions of the device are driven by the following impulse (event) signals:

LD - loading of the station. On this signal mini-transporter of the station starts moving to the right, pulling the workpiece, if it is present. Time of the loading if the workpiece was initially present at the tray is denoted as T_{LD}.

UNLD - unloading the station. On this signal mini-transporter moves the workpiece out of the station. If applied when not in the extreme positions may cause the lost of the workpiece. Time of the loading is denoted as T_{UNLD}.

ADV - moves the station from HOME to END, takes T_{ADV} time;

RET - moves the station from END to HOME, takes T_{RET} time; When an extreme position is reached, the corresponding moving terminates.

In the closed loop control structure presented in Figure 15 the sensors become inputs of the controller and the actors - its outputs. Besides the controller has "external commands" LOAD and UNLOAD which can be interpreted as coming from buttons or from other machines which interact with the transfer stage. Note that some outputs of the controller are not connected to the plant but reserved for the information exchange with other controllers.

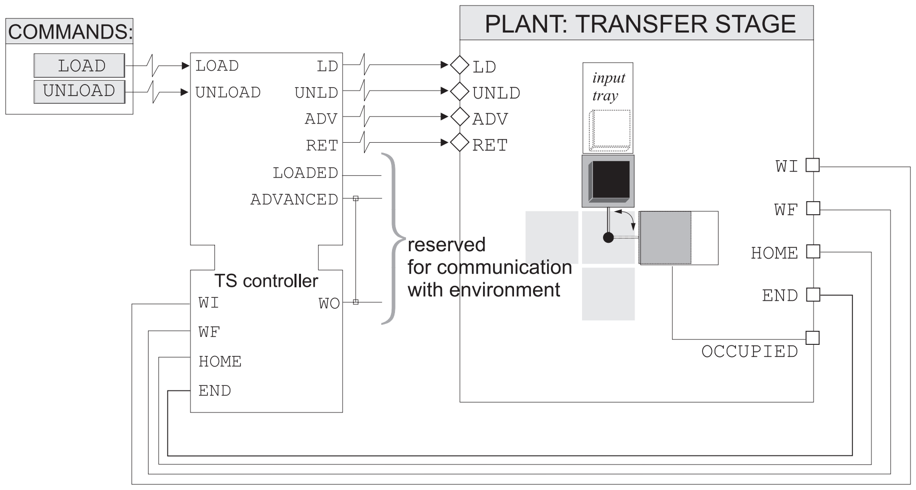


Figure 15: Controller and plant structure

The desired behavior of the system is explained as follows. Initially the transfer stage is in the HOME position. When the station does not contain a workpiece and the LOAD command occurs when a workpiece is available for loading at the input tray, the transfer stage must load the workpiece and move it to unloading position. There workpiece must be unloaded to the output tray upon the UNLOAD command comes. Then the transfer stage returns to the HOME position.

7.2 Definition of the controller as IEC1499 function block

Controller of the transfer stage, whose external interface is presented in Figure 15, is implemented as a state machine, which is shown in Figure 16.

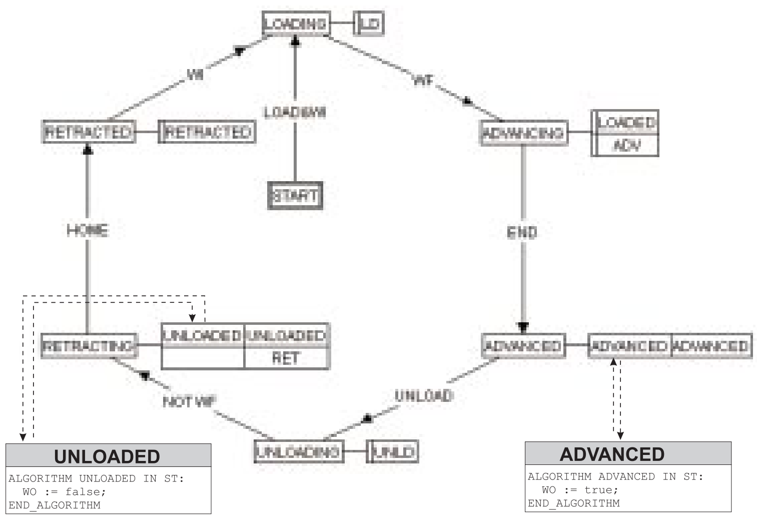
As it is clear from the Figure 15 some inputs and outputs of the controller directly correspond to those of the plant, while some other outputs require additional explanation:

LOADED - this output event signal is issued when the workpiece is loaded on the stage and advancing is started;

ADVANCED - this output event signal is issued when the stage arrives to the unloading position;

WO - workpiece is available for unloading, set TRUE when the stage with workpiece comes to the unloading position;

Since all the actors require impulses rather than Boolean conditions, controller can be implemented by the same structures which are normally used to define ECC as shown in Figure 16. Assigning of non-event output variables is not allowed in ECC, so external algorithms are used for this purpose. In the controller of transfer stage this is used to set/release the WO variable.



```
ALGORITHM UNLOADED IN ST:
  WO := false;
END_ALGORITHM
```

```
ALGORITHM ADVANCED IN ST:
  WO := true;
END_ALGORITHM
```

Figure 16: Execution control chart of the controller.

7.3 NCES model of the controller function block

The NCES model of the controller (Figure 18) is built according to the same rules as in the previously considered case of INTEGRAL-REAL. The only major difference is that conditions of transitions are more complicated which requires models for Boolean expressions.

Model of the ECC (Figure 17) is obviously also more complicated since the ECC itself has more states. Besides the model of ECC incorporates also models of algorithms ADVANCED and UNLOADED, which set and reset the output variable WO. Inputs $TC_1 - TC_7$ represent the conditions of the ECC transitions.

7.4 Model of uncontrolled plant behavior

Now consider discrete modeling configuration which is presented in Figure 19, where controller is connected to the discrete event model of the plant. All parts of the modeling configuration are TNCES modules connected by event and condition arcs.

The developed model of the transfer stage bases on the following assumptions about the object's behavior:

1. Loading is possible only in HOME and unloading only in END position.
2. Loading and unloading both activated by impulse for the designated time period (T_{LD} and T_{UNLD} respectively). No termination signal is required. Repeated signal while the previous is active does not lead to any action.
3. Both advancing and retracting automatically stop when the extreme position is reached.
4. An attempt to perform the unloading while moving (advance or retract), as well as to start moving while loading or unloading, leads to the fault situation. However ongoing signals LD or UNLD which arrive while the station is empty (even on the move) do not lead to any trouble.

The model presented in Figure 20 is built as a timed NCES which consists of two modules: one models position of the stage and the other - its loaded/unloaded state.

The former consists of 4 states p_1-p_4, where p_1(stage in the HOME position) and p_3(stage in the END position) are stable, and p_2, p_4 - are transient ones, corresponding to the moving HOME to END and vice versa.

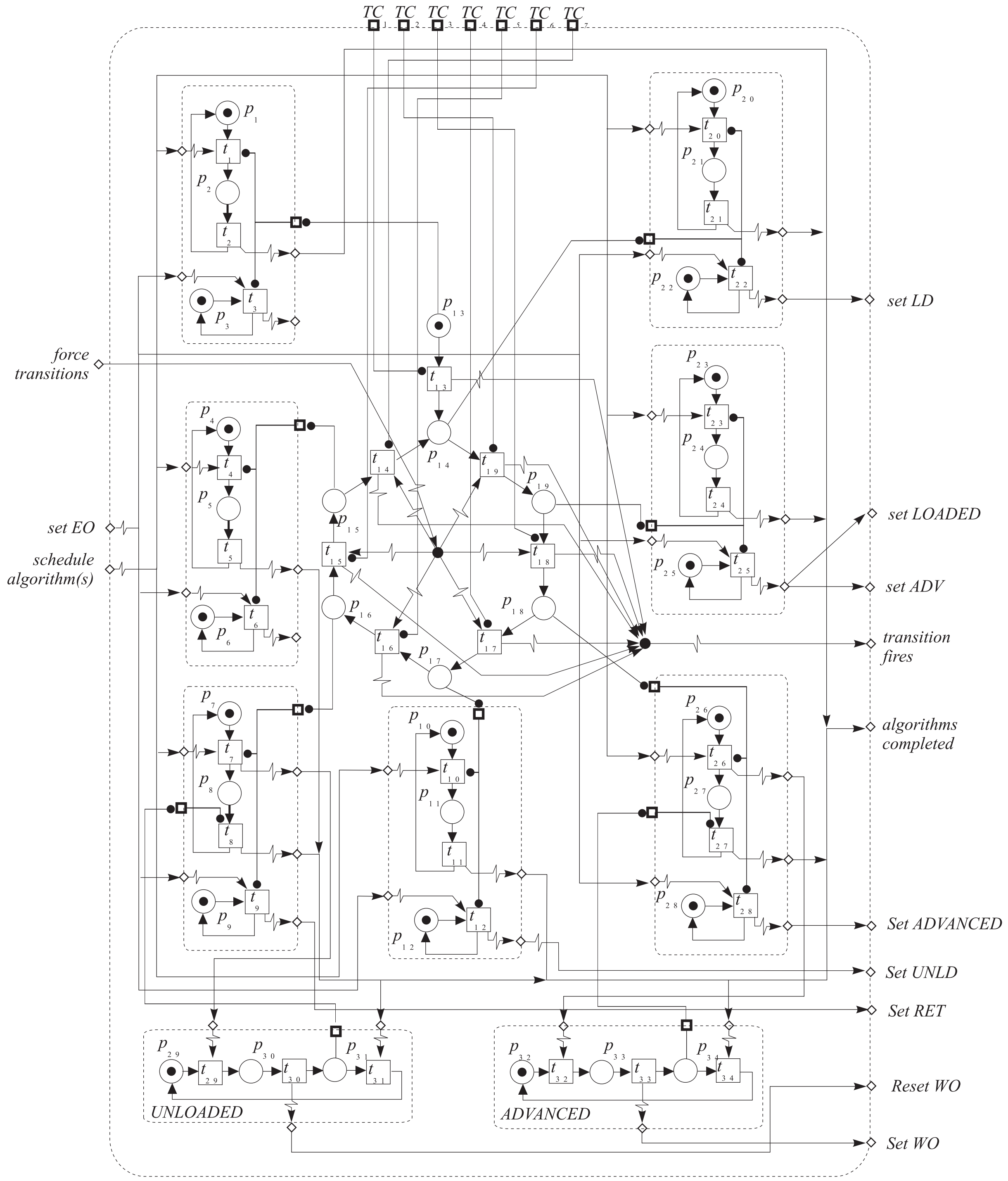


Figure 17: Model of the controller's ECC including algorithms.

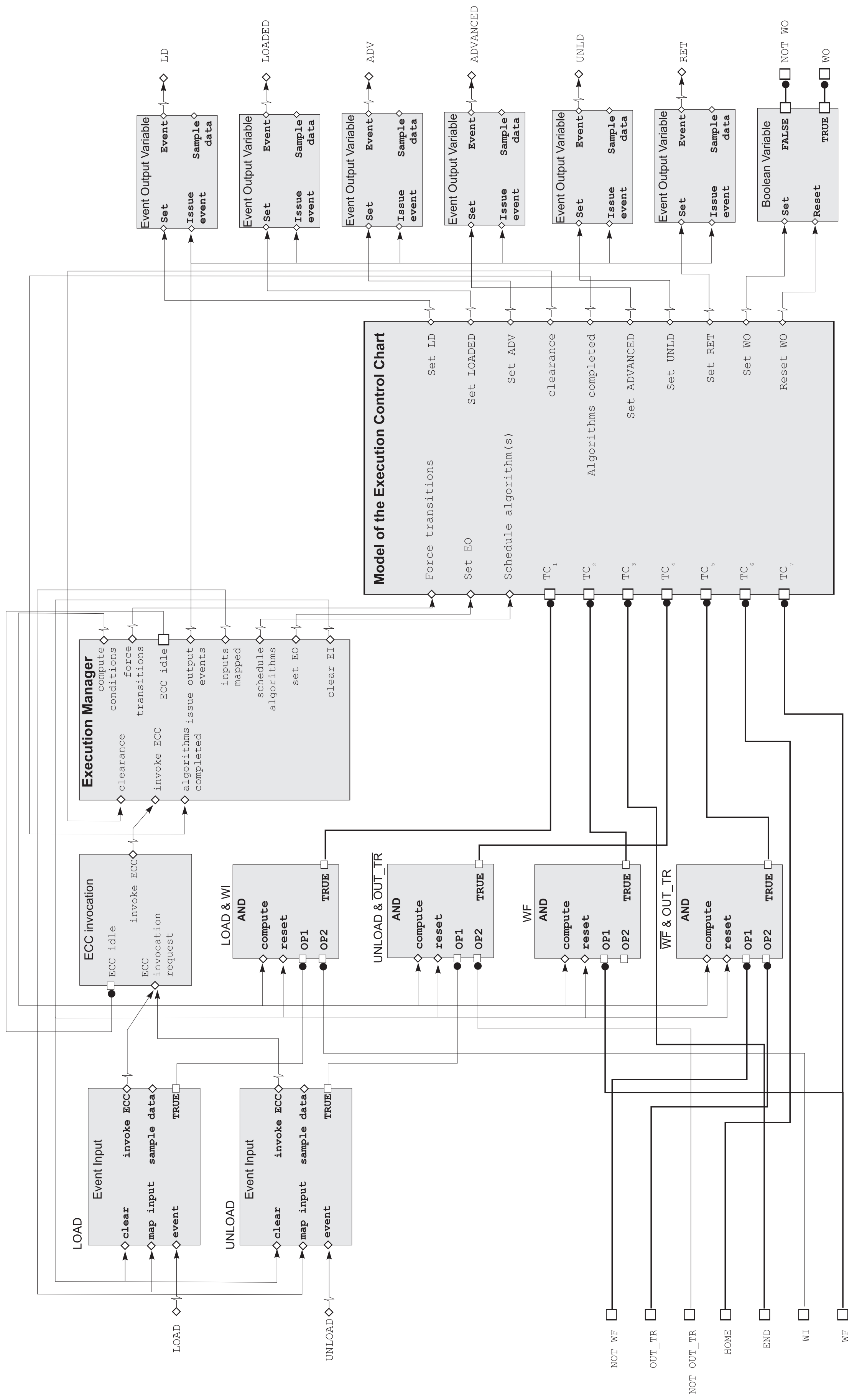


Figure 18: Modular NCES model of the transfer stage controller

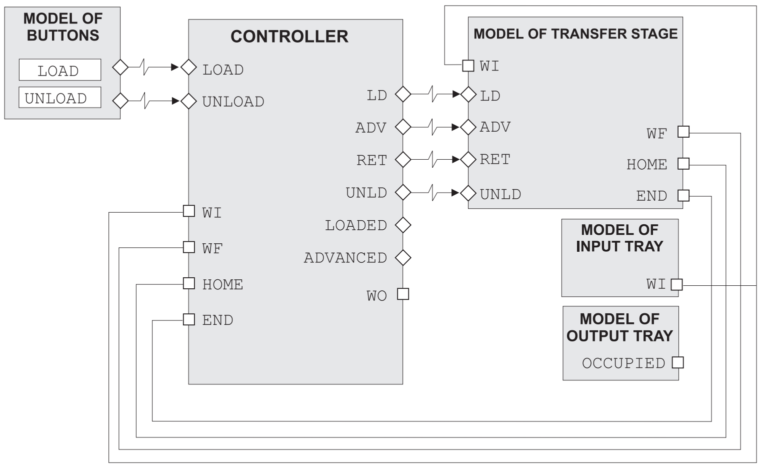


Figure 19: Discrete modeling configuration

The latter has the similar structure. Station becomes loaded only if a workpiece was provided and it passed through the loading in HOME position. Loading takes time T_{LD} upon which the sensor WO becomes showing "ON". It goes off immediately after the unloading starts.

States of the transfer stage can be specified by predicates in terms of marking of the model as follows. For example unloading in the end position is specified by the predicate

$$P_{unl.end} = p_8 \wedge p_3.$$

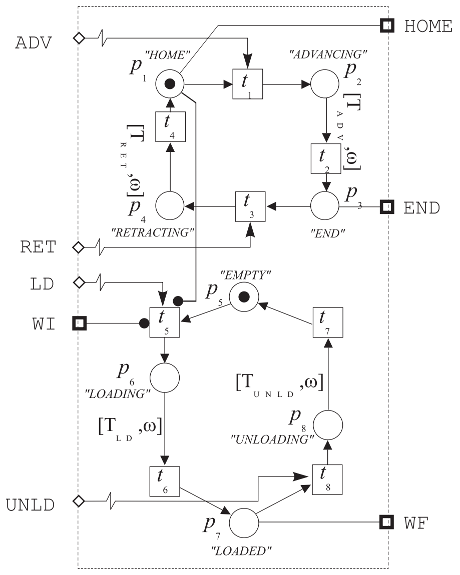


Figure 20: Model of uncontrolled plant behavior (transfer stage only).

7.5 Verification using the controller/plant model

The whole controller/plant model was exposed to the discrete event analysis: liveness, absence of dead transitions, analysis of the reachability graph, etc.

The most common technique of verification of controllers is checking various safety properties that express freedom from the fault states of the plant among the all reachable states of the plant/controller joint model. In our case we check the following fault states:

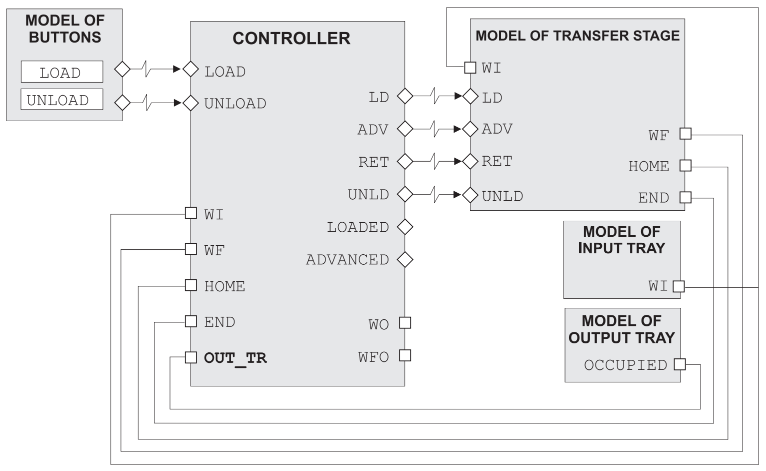


Figure 21: New configuration using the controller modified according to the verification results (input OUT_TR) is added and connected to the OCCUPIED output of the output tray model.

1. Stage starts to move (advance) before loading is completed. This property is checked by the following temporal formula:

   $$EF(P_{loading} \wedge P_{advancing}).$$

   The formula is proved to be FALSE.

2. Unloading is attempted while another workpiece is on the output tray. This fact is equivalent to the formula:

   $$EF(P_{out.tr} \wedge P_{unloading}),$$

   which is proved to be TRUE.

   This situation can be eliminated if we use additional information about the state of the output tray. For this purpose we add additional input OUT_TR to the controller function block, and connect this input to the $OCCUPIED$ output of the plant, as shown in Figure 21.

   The transition condition $UNLOAD$ from ADVANCING to UNLOADING is correspondingly substituted with the fortified condition $UNLOAD \vee \neg OUT\_TR$ to ensure that UNLOADING never starts if the output tray is still occupied.

3. Stage attempts to retract before unloading is completed. This fact is equivalent to the formula

   $$EF(P_{retracting} \wedge P_{unloading}).$$

   The formula is proved to be TRUE. This is due to the weak transition condition from UNLOADING to RETRACTING. To eliminate the dangerous state we also use the new input OUT_TR. The condition is transformed from $\neg WF$ to $\neg WF \wedge OUT\_TR$ which ensures that the transfer stage never starts to move back until reception of the workpiece is confirmed by the output tray.

7.6 Verification of controllers for composite plant

One of the declared key features of the IEC1499 is support of so called "no master controller" principle of building controller of a composite system having the controllers of components. The controllers of parts are just connected to each other to get the controller of the whole thing.

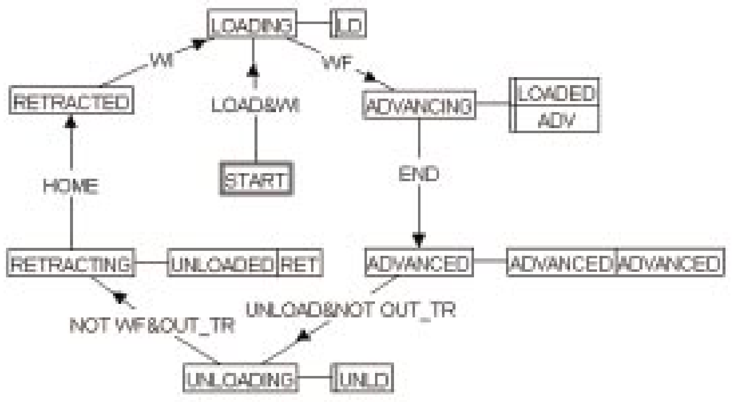


Figure 22: Modified controller of the transfer stage (condition of transition ADVANCED $\rightarrow$ UNLOADING is fortified with $\neg OUT\_TR$, and condition of transition UNLOADING $\rightarrow$ RETRACTING is fortified with OUT_TR).

We illustrate this principle building the composite plant from two transfer stages, where the first stage takes the workpiece from the input tray, then moves it to the unloading position, where passes it to the second transfer stage. The latter moves the workpiece to its unloading position on the output tray.

Controller of this object is obtained as a net of two single transfer stage controllers as shown in Figure 23. Output signal ADVANCED of the first controller is used to initiate the loading of the second by connecting it to the LOAD input. And output signal LOADED of the second controller is used to initiate the unloading of the first controller by connecting it to its input UNLOAD. Output WO of the first controller is used as input WI-"Workpiece available" of the second controller. Output WO "Workpiece available" of the second controller is used as input OUT_TR of the first.

We verified this configuration on compliance with the above stated properties. Also, we checked combined properties, which ensure the correct loading/unloading from one stage to another. For example, we checked the property, expressed by the formula:

$$EF(P^1_{unloading} \wedge P^2_{advancing}),$$

which reads "there is no state where second stage starts advancing before unloading of the first has been completed". We proved that this formula is FALSE.

Another question which concerns the interaction of the two stages at the unloading/loading point is to ensure that loading of the workpiece from one stage to another nds up, i.e. first stage returns to the initial position, while the second - advances to its unloading position. It can be checked by the temporal formulas

$$AG((P^1_{unloading} \rightarrow AF(P^1_{retracting}) \wedge (P^2_{loading} \rightarrow AF(P^2_{advancing})).$$

Surprisingly, this formula turned to be FALSE, and as it was then revealed due to the condition of transition from the UNLOADING to RETRACTING state of the controller. Instead of the WO of the second controller, which is not set immediately after the workpiece is on the transfer stage 2, we need to use its WF as the OUT_TR input of the first controller. To provide the correct style of data access, we do not attach the WF_2 directly to OUT_TR_1, but add an additional output $WFO = WF$ - "actual presence of the workpiece on the stage" to each controller. The resulting configuration looks as shown in Figure 24. Verifying the former formula on this configuration gives the positive answer.

8 Conclusion and outlook

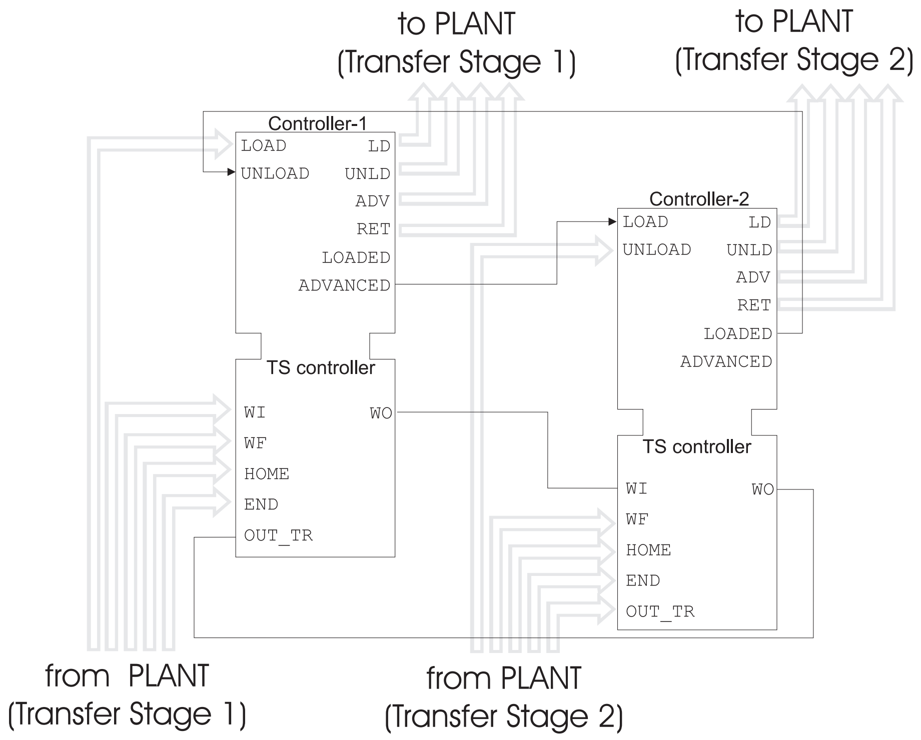


Figure 23: Composite controller of two transfer stages obtained as connection of two controllers.

Implementing the preliminary results presented in this paper into the practice would require as making them more understandable by the engineers, as well as extending the modeling framework.

To our opinion the following goals have to be fulfilled first:

- **Implementation of event operations.** The standard provides a set of predefined function blocks for operation with events. Implementation of the event blocks by NCES is required to model an arbitrary IEC1499 application.

- **Automatic model generation.** Develop formal rules of transformation of IEC1499 function blocks into NCES. This would allow to build a tool for converting IEC1499 applications into NCES models.

- **Extending the modeling framework.** So far we have studied only modeling of IEC1499 function blocks. The standard however provides the variety of hierarchical structures for building applications. Figure 25 shows the hierarchy of IEC1499 structures along with modeling components and properties which is possible to verify on the corresponding levels of the hierarchy.

  Thus, the level of function block and subapplication (composite function block distributed over several devices) allows to perform a ”rough” verification of the control strategy neglecting the dynamics and delays of the controller and communication networks. Taking these properties into an account would require modeling of the resource and device components as well.

- **Translation of practical specifications.** Probably the most difficult goal is to make a bridge between the practical specifications and requirements and the formal form of their presentation appropriate for the verification tools. This partly includes formulation of the specifications which is infamous for being a rather tricky informal art than a discipline.

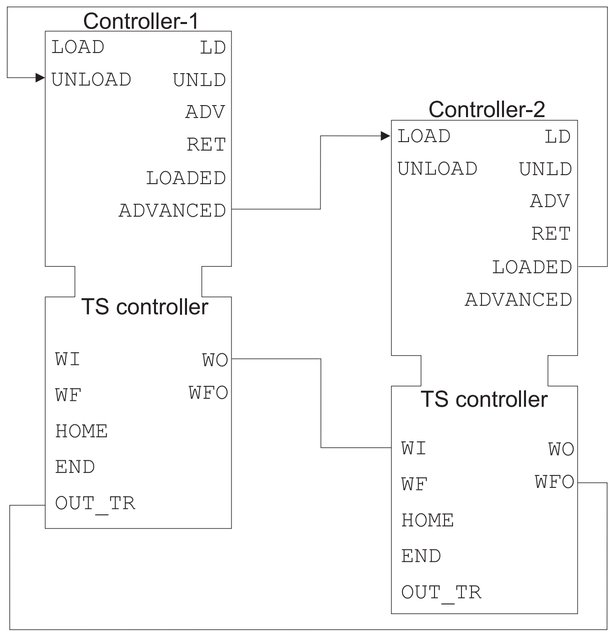


Figure 24: Corrected composite controller of two transfer stages (output WFO is added to each component controller, and WFO of the second controller is connected to OUT_TR input of the first one.)

9 Acknowledgement

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

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] R.S.Sreenivas and B.H.Krogh On condition/event systems with discrete state realizations *Discrete Event Dynamic Systems: Theory and Applications*, 2(1):209-236,1991

[5] Emerson, Mok, Sistla, and Srinivasan Quantitative temporal reasoning *Lecture Notes in Computer Science*, Vol.531:pp.136-145,1991

[6] 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

[7] 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.

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

[9] 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

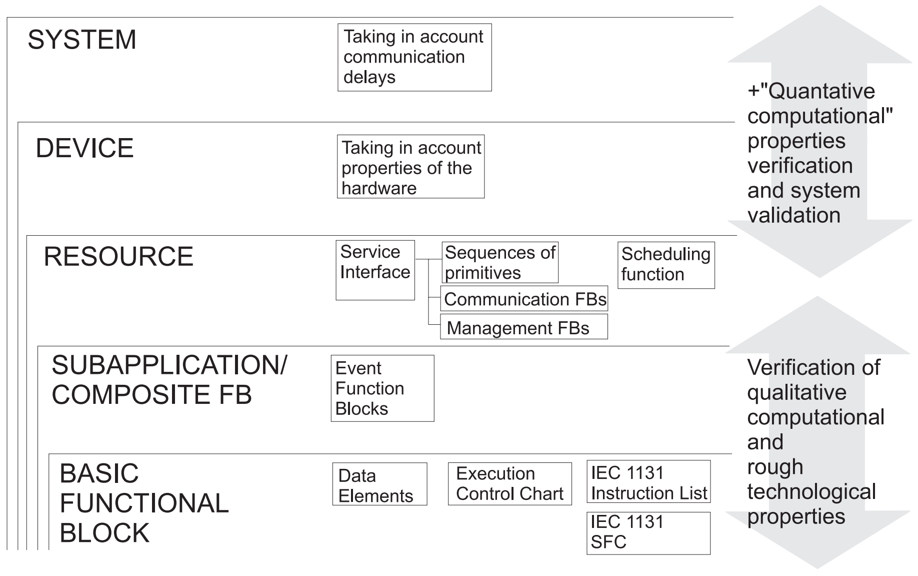

SYSTEM
Taking in account communication delays
DEVICE
Taking in account properties of the hardware
RESOURCE
Service Interface
Sequences of primitives
Communication FBs
Management FBs
Scheduling function
SUBAPPLICATION/ COMPOSITE FB
Event Function Blocks
BASIC FUNCTIONAL BLOCK
Data Elements
Execution Control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

Figure 25: Stages of modeling connected to hierarchy levels of IEC 1499 structures

[10] 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.

[11] S.Roch, P.H.Starke. *INA - Integrated Net Analyzer*, Version 2.2, Manual *Humboldt-Universit"at zu Berlin*, Berlin, 1999 http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/

[12] 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

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

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

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

[16] 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.

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

[18] 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.

[19] 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.