Using Visual Specifications in Verification of Industrial Automation Controllers

Valeriy Vyatkin*, Gustavo Bouzon and Hans-Michael Hanisch*****

*Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand

**Automation Engineer Controle Soluções em Mecatrônica Ltda., Rua Mauro Nerbass, 72, CEP 88024-420 Lages/SC Brazil

***Institute of Computer Science, Martin Luther University Halle–Wittenberg, 06099, Halle, Germany

This paper deals with further development of a graphical specification language resembling timing-diagrams and allowing specification of partially ordered events in input and output signals. The language specifically aims at application in modular modelling of industrial automation systems and their formal verification via model-checking. The graphical specifications are translated into a model which is connected with the original model under study.

1. INTRODUCTION

Formal verification of industrial automation systems requires three constituent components: a model of the controller, a model of the uncontrolled plant and a specification of desired or forbidden plant behaviour. Generation of the two first elements can be facilitated by application of modular modelling approaches and from automatic model-generation as described in [1].

However, languages commonly used for specification, such as temporal logic, are still rarely familiar to control engineers. So, the engineers would benefit from having user-friendly means of specifying the desired or forbidden behaviour of a model.

Inspired by the timing diagram specifications explored in the domain of digital systems design (e.g. by K. Fisler [3], N. Amla et al., [4], R. Schlör [5]), a graphical language for describing the dependency of interface signal changes was proposed in [6], and some of its implementation issues were developed in [7].

In this paper we harmonize the earlier developed specification and implementation techniques aiming at a solution that can be a part of an integrated verification framework. The underlying modelling language of the framework is the modular formalism of Net Condition/Event Systems (NCES) described in [8], [9]. The proposed visual language specifies the behaviour of NCES models and the verification technique also relies on the use of NCES. We suggest two procedures for translation and checking of visual specifications: one for verifying the output behaviour, and the other for combined input-output behaviour. The paper is organized as follows. Net Condition/Event Systems are briefly introduced in Section 2. Timing Diagrams as a means for specifying desired or forbidden behaviour of NCES models of automation systems are defined in Section 3. The transformation of Timing Diagrams to NCES modules is subject of Section 4. Section 5 describes the implementation of the method in a software prototype. Some conclusions are presented in Section 6.

2. NET CONDITION/EVENT SYSTEMS

The formalism of Net Condition/Event Systems (NCES) was introduced by Rausch and Hanisch in [8] as a modular extension of Signal/Net Systems (SNS) – a place-transition formalism for discrete state, discrete time modelling. The idea of Signal/Net Systems is described as follows.

2.1. Definition of SNS

A Signal/Net System is a tuple *(P,T,F,V,B,W,S,M,m0,eft,lft), where P* is a non-empty finite set of *places*; T is a non-empty finite set of *transitions* disjoint with P; F is the set of *flow arcs*, where $F \subseteq (P \times T) \cup (T \times P)$; V maps a weight to every flow arc and $V : F \rightarrow \mathbb{N}$; B is the set of *condition arcs*, which carry condition signals and $B \subseteq P \times T$; W maps a weight to every condition arc and $W : B \rightarrow \mathbb{N}$; S is the set of irreflexive *event arcs*, which convey event signals and $S \subseteq T \times T$; M maps a event-processing mode (AND or OR) to every transition, $M : T \rightarrow \{\boxed{\wedge}, \boxed{\vee}\}$; m_0: $P \rightarrow \mathbb{N}_0$ is the initial marking of *SNS*, where for each place $p \in P$, there are $n_p \in \mathbb{N}_0$ tokens; *eft* maps the *earliest firing time* to every pre-arc $[p, t] \in F$, *eft*: $F \cap (P \times T) \rightarrow \mathbb{N}_0$; and, *lft* maps the *latest firing time* to every pre-arc $[p, t] \in F$, *lft*: $F \cap (P \times T) \rightarrow \mathbb{N}_0 \cup \{\omega\}$, where $\omega \in \mathbb{N}$ and $0 \leq eft(p, t) \leq lft(p, t) \leq \omega$. The interval $[eft(p, t), lft(p, t)]$ is called the *permeability interval*.

A state of SNS model is determined by a) m – vector of marking of its places, i.e. allocation of tokens across the places; and b) u – vector of clock values.

An example of SNS is presented in Figure 1. The model consists of four places and five net transitions. Places p_1

and p_3 have tokens at the initial marking. Besides ten token flow arcs, the transition t_2 is connected to t_4 via an event arc, and place p_2 is connected to t_5 via condition arcs.

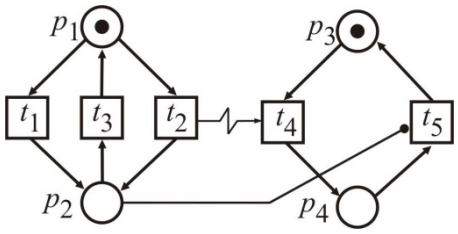

Figure 1. Signal/Net System.

2.2. State of SNS model

Places bear integer clocks whose values are denoted as u: $P \rightarrow \mathbb{N}_0$, where for each place $p \in P$, the clock reading in the place is denoted as $u_p \in \mathbb{N}_0$. All clocks have zero value at the initial state of the model. The clock of a place resets to zero anytime marking of the place changes.

A *state* in timed SNS is defined as a pair $z=[m, u]$, where m is a marking of P and u is the vector of the clock positions, such that $u(p) > 0 \rightarrow m(p) > 0$. Evolution of a SNS consists of changing its states. A state change (also called *state transition*) can consist in changing net's marking, or changing values of clocks (elapsing of time).

In every state there could be some *enabled* net transitions. If there are no enabled transitions then the clocks count (increment they value by 1) in all marked places and the SNS transitions to a new state. Otherwise, i.e. if there are some enabled transitions, then it is said that one or several enabled transitions *fire* that leads to the change of marking as explained by the firing rules. The set of simultaneously firing transitions is called *step.* In a given state there could be several different steps ready to fire, meaning that a state of SNS can have several successor states.

2.3. Firing rules

Let St denote the set of incoming event arcs of transition t: $St := \{t' | [t', t] \in S\}$. If St is empty, which indicates that no incoming event arc is associated with transition t, then t is spontaneous, otherwise it is forced. Firing of a forced transition is caused by firing of some other transition connected to it by an event arc. Both are included in the same step, i.e. fire simultaneously. Enabled spontaneous transitions can fire regardless of other transitions. For example, the transition *t4* in Figure 1 is forced and other transitions are spontaneous. Accordingly, the transition set T in can be subdivided on two disjoint sets: $T = Spont \cup Forc$, where *Spont* is the set of all spontaneous transitions of the *SNS*, and *Forc* denotes the set of all forced transitions of the *SNS*.

For any transition t, there can be three kinds of markings: the marking on incoming flow arc t^-, the marking on outgoing flow arc t^+, and the marking on incoming condition arc $\hat{t}$, defined as follows:

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

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

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

For any subset $s \subseteq T$, the marking s^- and s^+ denote the sum of markings t^- and t^+ respectively, and $\hat{s}$ represents the union of markings $\hat{t}$ for $t \subseteq s$.

The firing of a spontaneous transition is determined by the three factors listed below:

1. *Token concession*: A transition is said to have a *token concession* or is *token-enabled* when all the flow arcs from its pre-places are enabled. More specifically, a flow arc is enabled when the token number in its source place is not less than its weight, i.e. $m(p) \geq V(p, t)$. For example, given the marking m, transition t is token-enabled if $t^- \leq m$. Transitions which have no pre-places are always marking-enabled.
2. *Permeability interval*: The permeability interval defines the time constraints applied to the input flow arcs of transitions. A transition t: $\exists$ $(p, t) \in F$ is *time-enabled* only when clocks of all its pre-places have a time $u(p)$ within permeability interval of the corresponding place-transition arc: $eft(p, t) \leq u(p) \leq lft(p, t)$.
3. *Incoming condition signals*: A spontaneous transition may have incoming condition arcs. It is considered *condition-enabled* when all the condition signals on its incoming condition arcs are true, i.e. $\hat{t} \leq m$.

A spontaneous transition is eligible to fire only when it is token-enabled, time-enabled, and condition-enabled.

2.4. Step and state transitions

SNS is *executed in steps,* meaning that for each state transition there is a unique set of concurrently firing transitions $s \subseteq T$. A state is *dead* if no further step is enabled or will be enabled by elapsing time. For non-dead states, the *delay* $D(m,u)$ denotes the minimum amount of elapsed time before a step is enabled.

A step is referred as *executable at the state* $[m, u]$ if all of its constituent transitions fire after $D(m,u)$. The execution of an executable step s at state $[m, u]$ is accomplished by first elapsing $D(m,u)$ amount of time and then firing s.

The new state $[m', u']$ led by the execution of step s is determined by:

$$m' = m - s^- + s^+, \text{ and}$$

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

Subsequent step executions from the initial state construct the *reachability graph* of the *SNS* model, which illustrates the relationship of all realizable states within the state space. The reachability graph of a timed SNS can be represented as a 3-tuple:

$$RG = (Z, R, z_0),$$

where Z is a finite set of reachable states, R is a finite set of state transitions, and z_0 is the initial state $[m_0, u_0]$.

For any subsequent states $[m_i, u_i]$ and $[m_{i+1}, u_{i+1}] \in Z$, there is a state transition $\tau \in R$, such that $[m_{i+1}, u_{i+1}]$ is

reachable from $[m_i, u_i]$ via state transition τ. This state transition is also denoted as $[m,u] \xrightarrow{\tau} [m',u']$.

A. Adding modularity to SNS

A Net Condition/Event System [8], [9] is defined as a SNS augmented by interface elements: condition and event inputs and outputs, which can be connected by event and condition arcs to SNS transitions and places. NCES having no inputs is SNS.

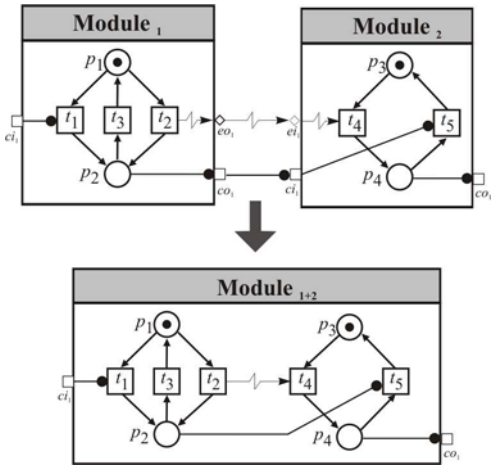

Figure 2. An example of a modular composition.

The NCES concept provides a basis for a compositional approach to build larger models from smaller components. According to the rules presented in [17], the composition is performed by connecting inputs of one module with outputs of another module as depicted in Figure 2. The modularity, introduced in NCES does not bring any semantic consequences - the model analysis is applied to the SNS resulting from the composition of several NCES modules.

The result of the composition of two NCES N_1 and N_2 is an NCES N_{1+2} obtained as a union of the components. The result of the composition again can be represented as a new module. Inputs and outputs of the "composition" are unions of the components' inputs and outputs, except for those which are interconnected to each other, and hereby "glued", i.e. substituted by the corresponding condition and event arcs. If the resulting NCES from Figure 2 is considered standalone, its condition input can be neglected making it semantically equivalent to the SNS from Figure 1.

The reachability graph of the model from Figure 2 is shown in Figure 3, assuming that the input *ci1* of the Module1 is not assigned. The transitions are shown as arcs of the graph, and are marked by names of NCES transitions simuntaneously occurred. Observing values of model parameters along a certain path in the reachability graph one can draw a timing diagram, like the one shown in the right part of Figure 3 for some outputs of the NCES modules from Figure 2 (some of which are inputs to another module).

NCES attempts to enhance the structured model development using place-transition nets. NCES models can precisely follow the structure of popular block diagram modelling and implementation languages, such as Stateflow of Matlab/Simulink and the function blocks of the IEC61499 standard – new reference architecture [15] used for modelling and implementation of distributed automation systems.

NCES were successfully used for modelling of traditional automation systems built using Programmable Logic Controllers (PLCs), as presented, for example in [1], [2], and of distributed embedded control systems following IEC61499 systems, as explained in [18].

The trend to improve structuring and composition potential of formal languages based on Petri net is seen in other dialects of Petri nets, as reported in [10] and [11].

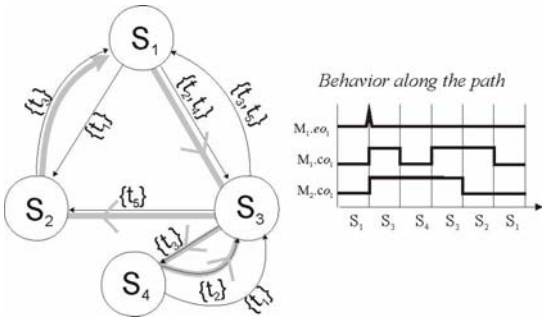

Figure 3. Reachability graph describing the complete behaviour of the model from Figure 2 and timing diagram in one of possible traces.

2.5. Integrated tools for model creation, editing and analysis

The timing diagram specification technique explained in this paper is a part of the tool chain for integrated modelling and verification of automation systems. The tool chain, described in more details in [16] consists of:

1) A graphical editor of NCES models;
2) The integrated environment for Model Assembly and Checking (VisualVerifier) that inputs model type files and is capable of assembling a composite, hierarchically organized model from modules contained in different libraries and translating the model into a "flat" NCES with the through numbering of places and transitions.

Thus, module boundaries are removed and the model-checking tools can be applied. In particular, the translator generates files in the input format of SESA model-checker [13].

Model-checkers like SESA prove properties of desired or prohibited behaviour of NCES models in their reachability space. A reachability graph like the one in Figure 3 is generated, and the properties are checked in its states or trajectories. Properties of single states can be captured in form of predicates, and properties of trajectories are usually defined in temporal logic languages, like CTL – Computation Tree Logic.

3. TIMING DIAGRAMS

3.1. Idea of use for specification

Capturing trajectory relevant properties in some formal language like CTL is quite difficult for control engineers. The idea of using timing diagrams for specification is to draw a specification graphically and then ask the model checker: if inputs behave like shown in the input diagram will outputs behave like in the output diagram? However, a single timing diagram describes only a single scenario.

Sometimes it is desirable to define a class of input scenarios with certain properties and then check if certain output patters are observed among all or any trajectories in the reachability graph. The idea is illustrated in Figure 4. The diagram consists of two parts: the upper (if) part presents the "input" part of guaranteed signals and the lower part is the "conjecture" to prove. In this example there is conditional restriction added between the rising edge of $M_1.co_1$ (event e_2) and the falling edge of $M_2.co_1$ (event e_3) – the restriction says that $e_3$occurs after e_2. Note that the signal $M_1.co_1$ belongs to both parts. In the "input" part it is specified by a single wavefront change that is simultaneous with the event $M_1.eo_1$. The waveform of the same signal in the "output" diagram is more complicated.

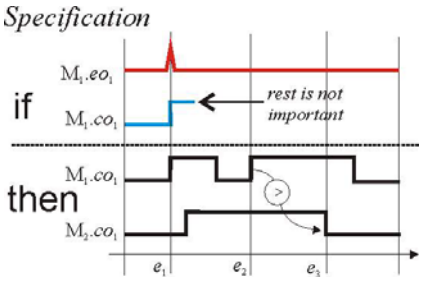

Figure 4. Timing diagram specification

Comparing the "then" part of the specification with the timing diagram of real behaviour in Figure 3 one sees that the specification holds in the given path. The idea of this paper is to enable such a check automatically using model checkers.

3.2. Definitions

The use of Timing Diagrams (TD) as a method of formal specification requires the definition of a graphical specification and its semantics.

In a diagram, sequences of changes in signal specification values are assigned to condition and event signals. Given the subsets $E \subseteq E^{in} \cup E^{out}$ and $C \subseteq C^{in} \cup C^{out}$, a specification for a signal set $A = E \cup C$ is described as a tuple $S = (A, f, g)$, where $f = f_e \cup f_c$ defines sequences of specification values: $f_e : E \to \Sigma_e^*$ with $\Sigma_e = \{noevent, maybe, always\}$ specifies sequences for event inputs and outputs, while $f_c : C \to \Sigma_c^*$ with $\Sigma_c = \{zero, any, stable, one\}$ defines values for condition signals.

The partial function $g : f(A) \times \mathrm{N} \times f(A) \times \mathrm{N} \to (>, =, \neq)$ assigns an ordering operator (precedence, simultaneity or non-simultaneity) between signal changes from different signals, in such a way that $g(a_i,m,a_j,n)$ indicates an ordering restriction between the m-th signal change of a_i and the n-th signal change of a_j.

A graphical description of a specification is illustrated in Figure 5 (for a model with outputs "FAILURE", "RESUME" and "SENS"). Signal changes at the beginning or ending of the diagram are implicitly simultaneous. Nevertheless, no further ordering is determined by the horizontal position of signal changes - therefore a timing diagram usually specifies a partial ordering among signal changes.

The semantics associated to the diagram is as follows: when the set of levels specified at the beginning of the diagram is achieved, it is required that the sequence of changes of the signals does not violate the partial ordering specified in the diagram, until a final state is reached.

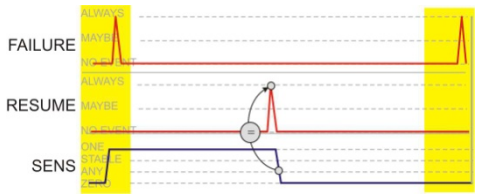

Figure 5. Specification including two event inputs, one condition output and a simultaneity operator.

3.3. Specified Signals

In order to describe specifications of NCES models, TDs must provide different representations for event and condition signals. Thus, we define the following possibilities of specification:

- in the case of a condition signal, the specification may have one of four possible levels: *zero*, corresponding to a logical zero; *any*, representing the situation where the signal might assume any logical value which can change at any state transition; *stable*, which also means undefined value, however assuming that the signal remains at a single level; or *one*, corresponding to the logical one;
- event signals are specified in two possible levels: *no event*, in the case where the occurrence of the event is forbidden, and *maybe*, meaning that the event might occur. It is also possible to specify an obligatory occurrence of the event *signal (always)*, but in this case only as a single pulse, because of the instantaneous nature of an event signal.

We define a *diagram event* as: any level change specified at a condition signal; a level change from *no event* to *maybe* or vice-versa, at an event-signal; or a specification of an obligatory occurrence of an event (*always* peak at an event signal).

3.4. Event Ordering at Different Signals

If a *partial ordering semantics is* assumed, no prior ordering of events on different signals is implicit. In other words, although each signal presents an ordering of its events, two events of different signals may occur at any sequence, except when special operators explicitly define their sequence. On the other hand, it is also possible to assume that the ordering of all events is defined through their position at the visual description. In this case, we are talking about a *strict* or *sequential ordering*.

Although more intuitive, adopting a sequential ordering would limit the representational capabilities of a diagram. Therefore, we adopt a partial ordering semantics for the TD language. In this case, the same TD represents a set of possible behaviours of the system, each one represented by a different event chain on the modelled system. Each chain is called *scenario,* and the set of scenarios defined by the diagram is named *diagram language*.

In Figure 6 (A) we observe the specification of two signals $\mathbf{s_1}$ and $\mathbf{s_2}$. Had we adopted a sequential ordering semantics, only one scenario would compose the diagram language: $\mathbf{s_2^+ s_1^- s_2^-}$. As the temporal dependence among events from different signals is not predefined (assumed partial ordering semantics) the same figure represents a TD with the following scenarios: $(\mathbf{s_2^+}, \mathbf{s_1^-})\mathbf{s_2^-}$; $\mathbf{s_2^+}(\mathbf{s_1^-}, \mathbf{s_2^-})$; $\mathbf{s_1^- s_2^+ s_2^-}$

and $s_2^+ s_2^- s_1^-$. Figure 6(B) indicates the timing diagram that, based on the adopted semantics, accepts as its only scenario $s_2^+ s_1^- s_2^-$, by introducing operators that indicate the obligatory ordering among events from different signals. The meaning of these operators will be stated in the next section.

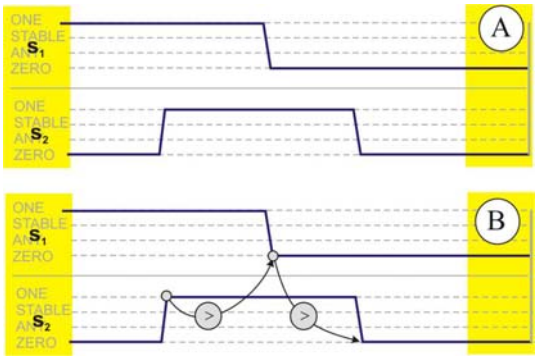

Figure 6. Temporally independent signals (a) and event ordering (b).

In order to constrain the ordering of two events from different signals, we define the following precedence operators:

≠ : events are not allowed to occur simultaneously;
= : events must be simultaneous;
> : event from the first signal must occur prior to the event from the second signal.

3.5. Specification of Finite Behaviour

The TD represents a finite behaviour that must be satisfied by the model. The satisfaction of a TD is evaluated from the moment when all specified signals are in their initial levels and some of them execute an initial transition, as indicated at the beginning of the diagram. The verification process ends when all signals achieve their final state, indicated in the end of the diagram. The initial part of the diagram, denominated *precondition*, corresponds to a condition, whose satisfaction by the model indicates that we must start comparing the model's behaviour with the remaining part of the TD. The comparison ends up when the final part of the diagram, called *postcondition*, is reached. Both pre- and postcondition are highlighted at the diagram (Figure 7).

When a TD specifies a finite behaviour, different interpretations are possible:

Existence of a scenario (from the diagram language): here we require that at least one of the specified scenarios will occur at the model. In other words, there is a path at the state tree of the model, where the precondition is satisfied and the behaviour of the model does not contradict the specification.

Existence of all scenarios: the existence of each scenario must be tested inside the state space of the model.

Generality of a single scenario: here a single scenario, from the set of scenarios specified at the diagram, must be recognized in every path, indicating a situation that has to occur in the future, regardless of which path is taken by the model.

Generality of the diagram's language: the behaviour specified at the diagram will eventually occur, no matter which scenario, in each path from the state tree of the model. Notice that, in this case, the existence of a path with no occurrence of the precondition would already be a counter-example.

Satisfaction of a single scenario: every satisfaction of the precondition must be followed by the satisfaction of the same scenario, among those that are possible according to the specification. This corresponds to an assume-guarantee clause, where the precondition plays the role of an assumption that, when fulfilled, guarantees the occurrence of a given sequence of events.

Satisfaction of the diagram: the specified behavior must not be contradicted, which means that every occurrence of the precondition at the model leads to a behaviour that is accepted by the diagram language. As a particular case, a model that presents no occurrence of a given precondition satisfies every specification starting with this precondition. The following topics will be based on this interpretation of the TD.

3.6. Specification of infinite behaviour

The timing diagram could also correspond to a specification to be satisfied from the time when the precondition occurs, without the need to specify a postcondition. In this case, the final state specified at the diagram would correspond to a restriction that must not be violated.

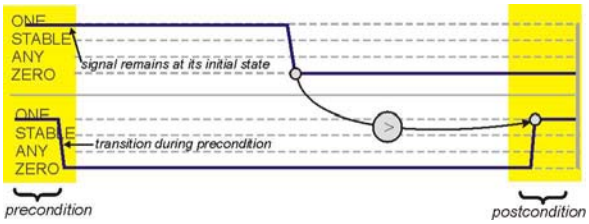

Figure 7. Pre- and postcondition.

The absence of a specification for the precondition could indicate that the initial state of the model should comply with the levels specified at the beginning of the diagram. Although these two approaches also present a practical appeal, the absence of postcondition or precondition will not be issued in the work, as a matter of simplicity.

In order to allow the translation of the timing diagram into a formal model, some requirements have to be done in respect to the events presented in each signal. Diagrams satisfying the requirements are said to be *feasible*.

4. NCES MODEL OF TIMING DIAGRAMS

When verifying autonomous NCES models without inputs, each signal specification is translated into a NCES supervisor module comprising two basic submodules: an **event generator** creates sequences of transitions, one for each change of level specified for the signal. Each transition stimulates, through an event arc, the corresponding event input of a **signal generator**, which causes the output of the signal generator to recreate the signal according to the input stimulated. Ordering operators are translated into special places and transitions that create interdependency of event generators.

The verified module is then connected through event arcs to the event generators of the corresponding signals, in such a way that every change of signal in the first is reported to the latter. Along with the translation of the

specification into NCES modules, a set of automatically generated temporal-logic statements is created. The composite module is then model-checked against these statements to verify if each transition at the supervisor always fires whenever the corresponding transition at the verified module is fired.

The graphical specification also provides automatic test possibilities for input/output behaviour or non-autonomous NCES modules. In this case, the NCES supervisor modules that describe input signals are used for generating the specified sequences of input signal changes, while the output signals are again verified as described before. The components of the NCES model of the timing diagram are detailed in the following sections.

4.1. Event Generator

The main part of the NCES model for the specification is called *event generator* and consists of a set of parallel *processes* (sequences of transitions and places), started simultaneously by the firing of a transition denoted $\mathbf{t_{start}}$. Each process is responsible for reproducing the behavior specified for one signal. Events on the signals are translated into transitions at the processes.

For each signal **i**, there is a place $\mathbf{p_{notstart,i}}$ which is a preplace of $\mathbf{t_{start}}$ and postplace of the last transition of the corresponding process. The transition $\mathbf{t_{start}}$ indicates the beginning of the timing diagram. The situation where the diagram language is not being executed corresponds to the marking $\mathbf{p_{notstart,i}}$=1 for every signal **i**.

In the case that at least a signal **j** has the marking $\mathbf{p_{notstart,j}}$=0, the marking $\mathbf{p_{notstart,i}}$=1 for a signal **i** indicates that this signal has already achieved the last level specified at the diagram.

The precedence relationships among events of different signals are mapped to special interconnections among the corresponding processes, as shall be detailed in the following section.

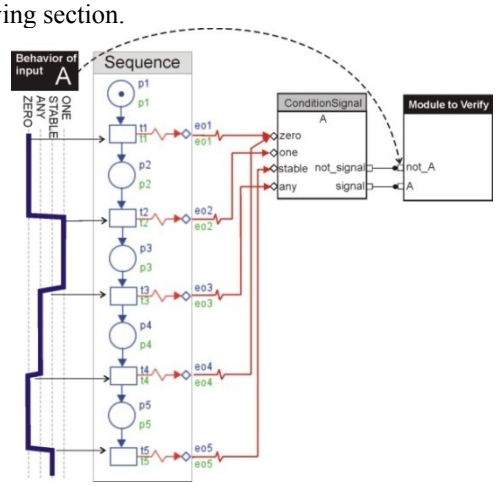

Figure 8. Translation of a single specification for a condition output, and linking to the verified model.

4.2. Signal Generation Module

For each specified signal, we create a signal generator module which reproduces, at its output, the possible values for the signal, according to the level specification stimulated at its input. Each event on the timing diagram (modelled by the firing of a transition at the event generator) stimulates, by an event arc, the corresponding change at the signal generator, which guarantees that the NCES module, resulting from the combination of the event generator with the signal generators, will reproduce at its output the diagram language. The idea is illustrated in Figure 8. To each condition signal included at the specification is assigned a signal generator module with four event inputs, corresponding to the four possible specification levels, and two condition outputs, indicating the two possible values assumed by the condition signal (*zero* or *one*).

Figure 9 shows the structure of a signal generator for a condition signal.

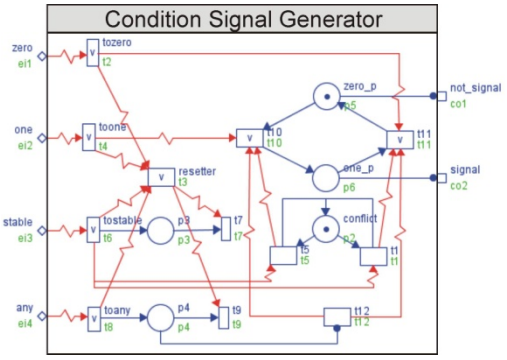

Figure 9. Generator of condition signals.

The transitions **tozero**, **toone**, **tostable** and **toany** receive event arcs, respectively, from the **zero**, **one**, **stable** and **any** event inputs.

Firing one of these transition means that the corresponding signal has changed its specification level to, respectively, *zero*, *any*, *stable* or *one* – in other words, a diagram event has occurred. The condition outputs **not_signal** and **signal** are linked to the internal places **zero_p** and **one_p**. The remaining transitions and places implement the desired non-deterministic behaviour - after the firing of **tostable** and **toany**, the marking of places **zero_p** and **one_p** should be non-deterministic, and may change randomly in the latter case, until another input event is stimulated. The place p2 always has a conflict with respect to transitions t5 and t1 leading to non deterministic choice in case of the signal 'to stable' (i.e. the stable value can be assigned either to 0 or to 1).

Figure 10 presents the internal structure of a signal generator for an event signal.

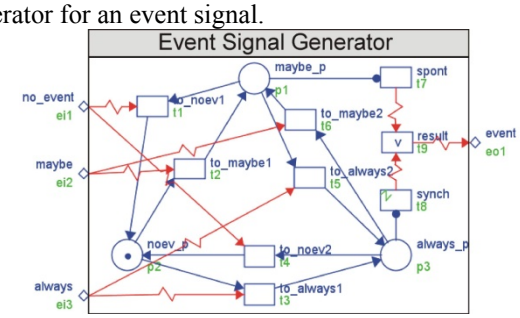

Figure 10. Generator of event signals.

Event signals are represented by modules with three event inputs, corresponding to the three possible specification values, and an event output, whose firing corresponds to the generation of the event. Internally, this generation corresponds to the firing of the **result** transition.

The transitions **to_noev#** (1 and 2), **to_maybe#** (1 and 2) and **to_always#** (1 and 2) are fired by stimulating the **no_event**, **maybe** and **always** inputs respectively. Every diagram event leads to the firing of at least one of these

transitions – actually, an *always* peak at the specification, followed by the specification of a new level, implies that both the **result** and the transition that leads to the new level specification (**to_noev#** or **to_maybe#**) will be enforced to fire.

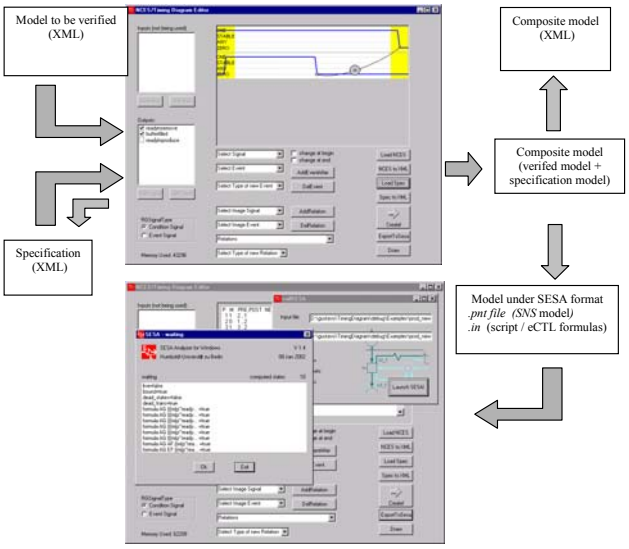

Figure 11. User interface of the TDE tool and file formats adopted for data storage

5. PROGRAM IMPLEMENTATION

The Timing Diagram Editor (TDE) is an application developed with the aims of providing the following functionalities:

- create, edit, save and load specifications of function blocks whose internal logic is specified by means of a NCES. These specifications are generated and visualized graphically as timing diagrams, while each signal at the timing diagram may be of one of the following types: event signals and condition signals; the signal levels allowed for each type of signals that were presented above.
- translate the combination of a function block and the behaviour specified for it into a composite finite state model (NCES) and temporal propositions written in the eCTL [12] format, in such a way that the composite model, and consequently the original function block, can be verified formally with the aid of the SESA tool [14]. If all the generated eCTL propositions evaluate to true with regard to the composite model, we conclude that the behaviour of the original model satisfies the specification.

The TDE tool uses XML as a storage format for both timing diagrams and NCES models and converts them to the input formats of the SESA model checker as illustrated in Figure 11.

CONCLUSION

The paper presented the idea of visual specification language to be used with modular discrete models, in particular of plant-controllers systems. Future work will include integration of this language to the visual verification framework [16].

ACKNOWLEDGEMENTS

The work of some authors was supported in part by the cooperative project VAIAS funded by the German Ministry for Education and Research (BMBF) and industry, and by the Deutsche Forschungsgemeinschaft under the reference Ha 1886/10-2 and Ha 1886/12-2 and by the University of Auckland (grant UARC 3607207).

REFERENCES

[1] H.-M. Hanisch, J. Thieme, A. Lüder, O.Wienhold, Modeling of PLC Behaviour by Means of Timed Net Condition/Event Systems, 6th IEEE Conference on Emerging Technologies and Factory Automation, Los-Angeles, 1997

[2] H.-M. Hanisch, A. Lobov, J. L. Martinez Lastra, R. Tuokko and V. Vyatkin: *Formal Validation of Intelligent Automated Production Systems towards Industrial Applications,* International Journal of Manufacturing Technology and Management, Volume 8 No. 1/2/3, 2006

[3] Fisler, K.: *Timing diagrams: Formalization and algorithmic verification*. Journal of Logic, Language, and Information, 8(7), July 1999.

[4] Amla, N., Emerson, E., Kurshan, R., and Namjoshi, K: *Model checking of synchronous timing diagram*, Conf. on Formal Methods in Computer Aided Design, Nov. 2000

[5] Schlör, R., Allara, A. and Comai, S.: *System Verification using User-Friendly Interfaces*. In *Design, Automation and Test in Europe,* pp. 167-172. IEEE Comp. Soc. Press, 1999

[6] Vyatkin, V. and Hanisch, H.-M.: *Application of Visual Specifications for Verification of Distributed Controllers*, Proc. of IEEE Systems, Man, and Cybernetic Conf, pp. 646-651, Tucson, 2001

[7] G. Bouzon, V. Vyatkin, H.-M. Hanisch: *Timing Diagram Specifications in Modular Modelling of Industrial Automation Systems*, IFAC World Congress, Prague, 2005

[8] Rausch M. and Hanisch H.-M.: *Net condition/event systems with multiple condition outputs*. Symposium on Emerging Technologies and Factory Automation, Paris, France, October 1995, Proc., Vol.1, pp. 592-600, INRA/IEEE

[9] Hanisch, H.-M. and Lüder, A.: *Modular Modelling of Closed-Loop Systems*, Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Germany, October 21-22, 1999, Proc, pp. 103-126

[10] L. Gomes, J. P. Barros, "*Structuring and Composability Issues in Petri Nets Modeling*", *IEEE Trans. on Industrial Informatics*, Vol. 1, No. 2, pp.112-123, 2005

[11] N. Hagge, B. Wagner, "*A New Function Block Modeling Language Based on Petri Nets for Automatic Code Generation*", IEEE Trans on Industrial Informatics, 1-4, 2005

[12] Roch, S.: *Extended Computation Tree Logic*, in Proc. of Workshop on Concurrency, Specification and Programming, Berlin, 2000

[13] P. Starke, S. Roch, K. Schmidt, H.-M. Hanisch, and A. Lüder, *Analysing Signal-Event Systems, Technical report, Humbold,*[Online]:http://www.ece.auckland.ac.nz/~vyatkin/tools/modelchekers.html

[14] SESA – Signal/Net System Analyzer, [Online]. Available: http://www.ece.auckland.ac.nz/~vyatkin/tools/modelchekers.html

[15] Function Blocks for Industrial Process Measurement and Control Systems, International Electrotechnical Commission, Part 1: Architecture, Geneva, 2005

[16] Visual Verification Framework, [Online], http://www.fb61499.com/valid.html

[17] Thieme, J. *Symbolische Erreichbarskeitanalyse und automatische Implementierung struktuirter, zeitbewerter Steuerungsmodelle*, Dissertation zur Erlagung des Grades Dr.-Ing., Berlin: Logos Verl., 2002

[18] Vyatkin, V. and Hanisch H.-M.: *Verification of Distributed Control Systems in Intelligent Manufacturing,* Journal of Intelligent Manufacturing, special issue on Internet Based Modelling in Intelligent Manufacturing*, vol.14, N.1, 2003, pp.123-136*