TIMING DIAGRAM SPECIFICATIONS IN MODULAR MODELING OF INDUSTRIAL AUTOMATION SYSTEMS

G. Bouzon*, V. Vyatkin, H.-M. Hanisch****

**Dept. Automation and Systems, CTC*
CP 476, Federal University of Santa Catarina
88040-900, Florianópolis, SC, Brazil
gbouzon@das.ufsc.br

***Dept. of Engineering Sciences,*
Martin Luther University Halle–Wittenberg,
06099, Halle, Germany
Valeriy.Vyatkin@iw.uni-halle.de
Hans-Michael.Hanisch@iw.uni-halle.de

Abstract*:* This paper deals with further development of the timing-diagram based specification language destined for application in modular modeling of industrial automation systems. The results of this work are intended for application in formal verification of software intensive automation systems.

Keywords: formal verification, visual specification, timing diagrams, Net Condition/Event systems

1. INTRODUCTION

Formal validation 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 behavior.
Generation of the two first elements can be facilitated by application of modular modeling approaches and from automatic model-generation as described in (Vyatkin, Hanisch, 2003).

However, the 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 behavior of a model.
Inspired by timing diagrams, well-known in the hardware domain (e.g. explored in works (K. Fisler,1999), (N. Amla et al., 2000), (R. Schlör, 1999)), a graphical language for describing the dependency of interface signal changes has been proposed in (Vyatkin and Hanisch, 2001).
In this paper we proceed with the issues that are specific for application of timing diagrams for specification and verification purposes of some classes of industrial automation systems.
The work is based on the model representation in the formalism of Net Condition/Event Systems (NCES) as described in (Rausch and Hanisch, 1995), (Hanisch, Lüder, 1999).
The basic idea of the formalism is to encapsulate a representation of discrete event dynamic behavior in modules with input and output signals of two types, namely

- condition signals that provide information about a state of a module, and
- event signals that provide information about a state transition of a module.

This distinction is essential for the sequel of this contribution. More details about the modeling formalism are provided in (Thieme, 2002).

This paper suggests two procedures for translation of visual specifications that differ slightly depending on whether the verified module has inputs.
It is therefore organized as follows. Timing Diagrams as a means for specifying desired or forbidden behavior are defined in Section 2. The transformation of Timing Diagrams to NCES modules is subject of Section 3. Section 4 describes the implementation of the method in a software prototype. Some conclusions are presented in Section 5.

2. TIMING DIAGRAMS

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 S is illustrated in Figure 2. 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.

2.1. *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 might assume four possible levels: *zero*, corresponding to a logical zero; *any*, representing the situation where the signal might assume any logical value; *stable*, which also means undefined, 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 impulse, because of the instantaneous nature of an event signal.

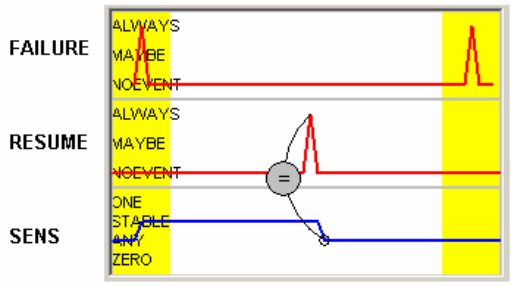

Figure 2. Specification including two event inputs, one condition output and a simultaneity operator.

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).

2.2. *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 explicit 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 behaviors of the system, each one represented by a different event chain on the modeled system. Each chain is called *scenario*, and the set of scenarios defined by the diagram is named *diagram language*.
In Figure 1 (a) we observe the specification of two signals **s1** and **s2**. If we have adopted as our

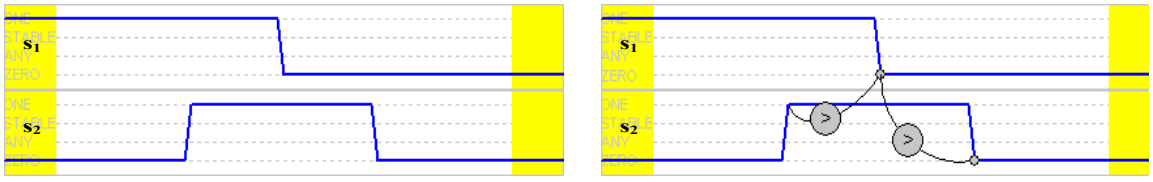

Figure 1. Specifying temporally independent signals (a) and event ordering (b).

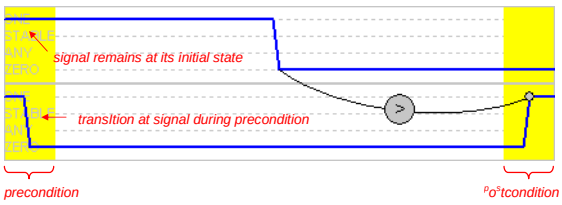

Figure 3. Pre- and postcondition.

convention 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: $\mathbf{(s_2^+, s_1^-) s_2^-}$; $\mathbf{s_2^+ (s_1^-, s_2^-)}$; $\mathbf{s_1^- s_2^+ s_2^-}$ and $\mathbf{s_2^+ s_2^- s_1^-}$. Figure 1(b) indicates the timing diagram that, based on the adopted semantics, accepts as its only scenario $\mathbf{s_2^+ s_1^- s_2^-}$, 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.

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.

2.3. Specification of Finite Behavior

The TD represents a finite behavior 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 behavior 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 3).

When a TD specifies a finite behavior, 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 behavior 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 behavior 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 behavior 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.

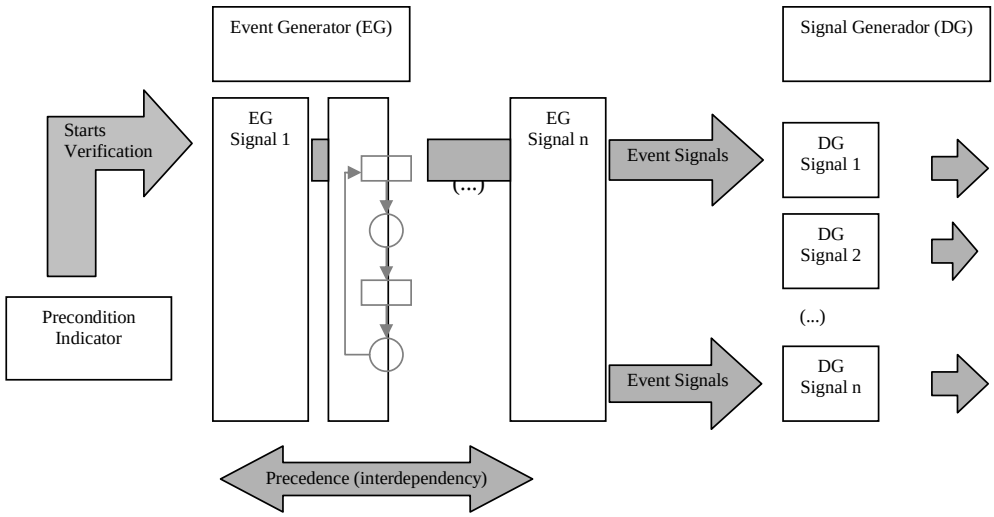

Figure 4. Diagram representing the main functionalities of the specification model.

2.4. *Specification of Infinite Behavior*

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.

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.

2.5. *Feasibility of Finite Behavior Diagrams*

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.

3. 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 behavior 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.

3.1. *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 $\mathbf{t_{start}}$

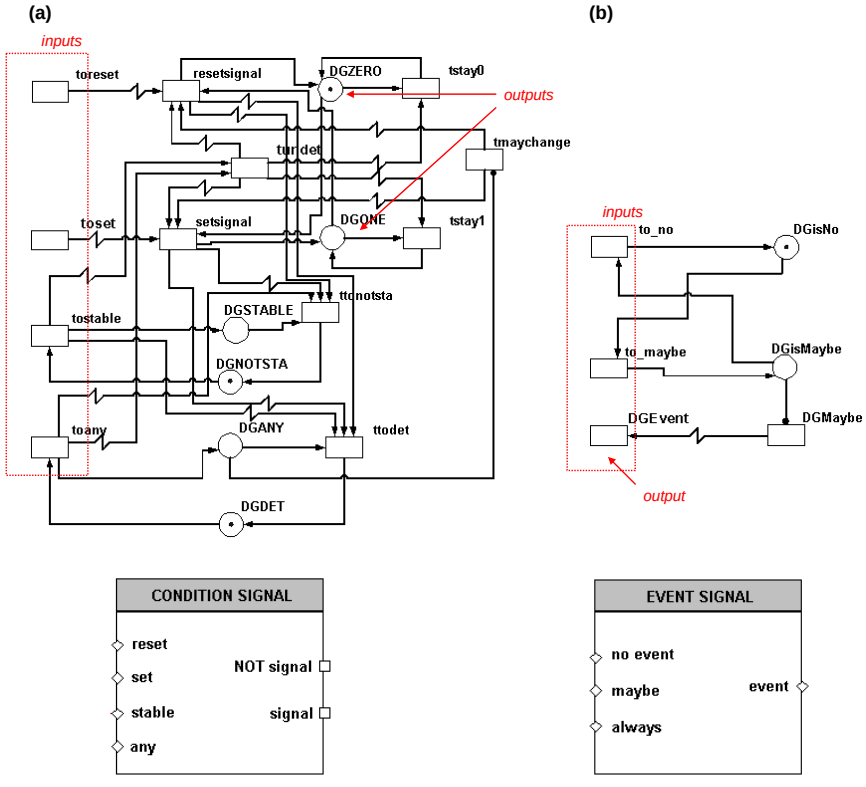

Figure 5. Signal generator for condition signals (a) and event signals (b).

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

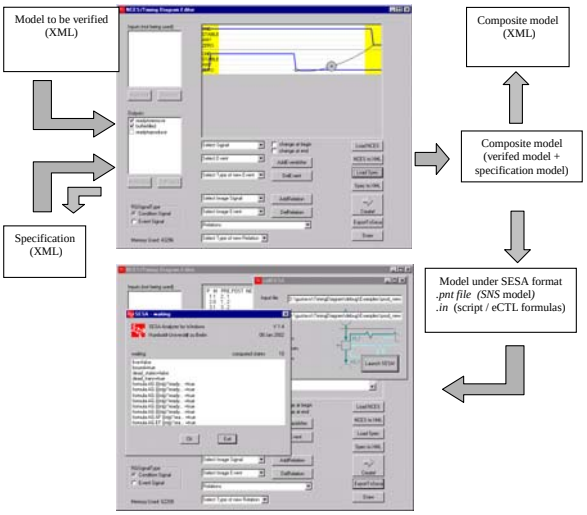

Figure 6. User interface of the TDE tool and file formats adopted for data storage

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.

3.2. 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 (modeled 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.

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 5 (a) shows the structure of a signal generator for a condition signal. The transitions **toreset**, **toset**, **tostable** and **toany** receive event arcs, respectively, from the **reset**, **set**, **stable** and **any** event inputs[1]. 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 **NOTsignal** and **signal** are linked to the internal places **DGZERO** and **DGONE**. The remaining transitions and places implement the desired undeterministic behavior - after the firing of **tostable** and **toany**, the marking of places **DGZERO** and **DGONE** should be non-deterministic, and may change randomly in the latter case, until another input event is stimulated.
Figure 5 (b) presents the internal structure of a signal generator for an event signal. 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 **DGEvent** transition. The transitions **to_no**, **to_maybe** and **DGEvent** 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 **DGEvent** and the transition that leads to the new level specification (**to_no** or **to_maybe**) will be enforced to fire.

4. 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 (Roch, 2000) 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 (SESA, 2004). If all the generated eCTL propositions evaluate to true with regard to the composite model, we conclude that the behavior of the original model satisfies the specification, as described in Chapter 5.

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 6.

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

[1] Although we can imagine the interface of an NCES module, it is not really implemented by the set of software tool used at this work. Therefore, linking an external transition to an event input is done by linking the transition, through event arcs, directly to the internal transitions that are supposed to be linked to the event input. Similar approach is used for module outputs.

Forschungsgemeinschaft under the reference Ha 1886/10-2 and Ha 1886/12-2.

REFERENCES

Amla, N., Emerson, E., Kurshan, R., and Namjoshi, K: *Model checking of synchronous timing diagram,*. Conference on Formal Methods in Computer Aided Design (FMCAD), Proceedings, November 2000

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

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, Proceedings, pp. 103-126

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

Rausch, M. and Hanisch, H.-M.: *Net condition/event systems with multiple condition outputs*. In Symposium on Emerging Technologies and Factory Automation, volume 1, p. 592-600, Paris, France, October, 1995. INRIA/IEEE

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

SESA – Signal/Net system analyzer. Humboldt Universität zu Berlin, Institut für Informatik, http://www.informatik.hu-berlin.de/lehrstuehle/automaten/tools/

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

Vyatkin, V. and Hanisch, H.-M.: *Application of Visual Specifications for Verification of Distributed Controllers*, Proceedings of the 2001 IEEE Systems, Man, and Cybernetic Conference.

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*