# OPEN OBJECT-ORIENTED MODELLING AND VALIDATION FRAMEWORK FOR MODULAR INDUSTRIAL AUTOMATION SYSTEMS

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

*Dept. of Engineering Sciences,*
*Martin Luther University Halle–Wittenberg,*
*06099, Halle, Germany*

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

Valeriy.Vyatkin@iw.uni-halle.de
Hans-Michael.Hanisch@iw.uni-halle.de
gbouzon@das.ufsc.br

Abstract: This paper introduces a framework for formal modelling and validation of automation systems intended to be used by control engineers. The framework is based on a graphical, modular, and typed formalism of Net Condition/Event Systems.
This allows for modelling of realistic hierarchically organized industrial automation systems in a closed loop. The framework consists of methodologies and tools which enable formal analysis of automation systems.
The framework will be used to improve safety, reliability and robustness of automation systems predicting potential faults and deadlocks. 

Keywords: Verification, PLC modelling, manufacturing systems, discrete event systems.

## 1 INTRODUCTION

Modern production systems need to be more flexible and re-configurable. For this reason they are built from standardized processing modules. Their software is also organized in a modular form and is executed on distributed control devices.
When new configurations of production systems are formed from the modular components, the testing becomes a bottleneck for quick commissioning. Formal validation can reduce the time-consuming testing and commissioning phases of system's development and deployment. As the functionality of such systems is determined by cooperation of entities of heterogeneous domains, e.g. mechanical, electric, automation hardware and software, the validation has to take into account the relevant properties from all these domains.
Formal modelling of automation systems proved to be helpful for validation of automation systems by simulation or by formal verification of static and dynamic properties. In automation systems the software represents a variable part, while the models of equipment can be reused through the engineering cycle. Once developed by a machine vendor, the models may follow the equipment, enabling the machine users (e.g. system integrators) to validate new configurations of the machines re-using the models of their components.
This vision, however, requires a more systematic approach to the modelling, than that can be seen now. Models in most of the formalisms such as Petri nets or finite automata lack integrating capabilities: while they may cope well with the modelling of a particular process, building the overall model of a system comprising several processes is difficult.
An opposite example make the modelling techniques based on the Unified Modelling Language (UML). The UML is getting increasingly popular also in automation for its ability to describe systems in object-oriented form. However, the UML lacks a formal background and can hardly be used for deep analysis of the systems.
This paper tries to sketch another approach for systematic modelling of systems by means of a modular modelling formalism.
The paper is organized as follows: Section 2 discusses some extensions to the formalism. Section 3 introduces the approach to modelling and describes the framework of tools supporting it. Section 4 elaborates more on the application scenarios of modelling for better automation systems. Section 5 considers an example of a machine modelling, and Section 6 provides some considerations on the validation. The paper is concluded with discussion of future research.

## 2 STRUCTURAL EXTENSIONS AND LIMITATIONS

### *2.1 Modelling formalism*

In this work we use the formalisms of Net Condition/Event Systems (NCES) (*H.-M. Hanisch and A. Lüder, 1999*). The modular capabilities have encouraged further development of a systematic approach to modelling of industrial systems. The formalism is:

**Modular**, i.e. it provides encapsulation of place/transition models into modules, connected to each other by condition and event arcs.

**Graphical**, that simplifies understanding of the model's semantic and facilitates application by engineers.

**Distributed state**, that helps to cope with the complexity of model-checking, especially when decentralized systems are modelled.

**Discrete time**, that allows to add new, time dimension to the discrete modelling.

Supported by the model-checking tool SESA (*Starke, Roch et al.*), NCES were applied in a number of studies on formal validation of automation systems (e.g. *H.-M. Hanisch, T. Pannier et al, 2000*).

However, the borders of the NCES formalism have been reached. In particular, questions of creating nested modular models (i.e. encapsulation of a network of modules inside a module) theoretically consideration of which started in *(Thieme, 2002*), needs to be continued and reflected in implementations.

The automation becomes a field of distributed architectures (e.g. IEC61499) where automation systems are represented as networks of function blocks. Evident similarities between the NCES modelling concept and the function block architectures have motivated further development (*Vyatkin V., Hanisch H.-M. et al. 2003*) of the NCES formalism that targets:

- Definition of model types to simplify encapsulation and reuse of the models;
- Less restricted rules for interconnections of modules (multiple input and output links are allowed).

The formalism is supported by the corresponding tools and methodologies.

### *2.2 Model type definition*

In the formalism discussed in this paper a basic unit of model is called module. A module is defined by its **interface** and **content**. The interface contains a type name, names of event inputs and outputs, and of condition inputs and outputs. The content can be either a NCES, i.e. places, transitions and arcs as described in the previous section, or a network of model instances, i.e. of other modules, interconnected via event and condition arcs.

A NCES has to be encapsulated in a module. The corresponding model type is called **basic**. A **basic** model type is defined by a graphical representation of a NCES module, i.e. by definition of its interface (event and condition inputs and outputs), and of its place/transition model. The module identifier serves as the type identifier. The values of inputs influence the model's dynamic as they are connected to the model's transitions by condition and event arcs. The model's states and transitions may be reflected at the outputs as they are connected to places/transitions of the model by condition and event arcs.

A **composite** model type includes a description of block's interface, instances of constituent modules that are instances of other model types (basic or composite), and connection arcs.

The definition above makes the NCES "compatible" with other kinds of object-oriented modelling, for example using Unified Modelling Language (UML). Several works appeared recently on application of object-oriented modelling to machines and production systems (e.g. *M. Bonfè, C. Fantuzzi*, *2003*). The UML class diagrams are used in these works to represent the structure of production objects as composed from more elementary ones, that also paves the way to hierarchical models. However, application of UML for formal analysis is difficult as it lacks formality. Thus, the approach presented in this paper bridges the gap between the expression power of UML and the formal semantics of NCES.

## 3 MODELLING APPROACH AND TOOL FRAMEWORK

### *3.1 Closed-loop modelling*

The control systems is considered as composed of two independent components: object and controller, connected in a closed loop by control signals and process data. Modelling according to this view requires to model uncontrolled reactive behaviour of objects.

It is worth mentioning that the closed-loop approach to the modelling enables expression of the specifications directly in terms of the machine behaviour (not only I/Os of the controller).

### *3.2 Integrated tools for model creation, editing and analysis*

The modelling approach explained in this paper is supported by a number of software tools:

The graphical editor provides full graphical authoring and editing of the models. The editor uses an open XML-based data format for basic and composite NCES models. The data format of composite model blocks intentionally was made identical with that of IEC61499 function blocks, supported by tool (*FBDK*).

The integrated environment for Model Assembly (iMA) inputs the model type files given in XML and is capable of:

1) Assembling a composite, hierarchically organized model from modules contained in different libraries. The component model types are instantiated into NCES modules.
2) Translating the model into a "flat" NCES with the through numbering of places and transitions. The inter-module connections are converted into event and condition arcs between places and transitions. Thus the 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.

## 4 MODELLING OF AUTOMATED PLANTS

Benefits of the typed modelling are well visible in the following example of object modelling. The automated lifter (product of Flexlink Automation Oy., FINLAND) as shown in **Figure 1** is used in production of electronic components. The lifter can be controlled by two different controllers: an OMRON PLC programmed in ladder logic and Nematron SoftPLC programmed in Visual Flow Chart language. Though both controllers achieve similar control goals, the internal logic of control algorithms and even the logic of program execution are completely different (cyclically scanned vs. sequential). However, both controllers eventually deal with the same object.

When the closed-loop plant-controller systems are validated, the model of the lifter can be reused over and over again in connection with models of controllers of different types.

The lifter consists of three transporters, one of which is mounted on a vertically moving platform driven by a step motor as schematically represented in **Figure 2**.

**Figure 1.** The lifter.

The structure of the model type "Lifter" is defined by means of UML class diagrams as shown in Figure 3.

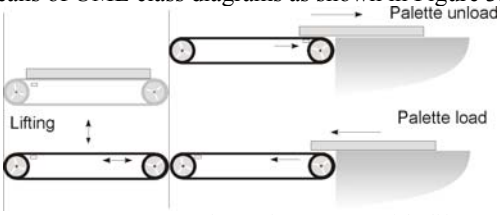


**Figure 2.** Structure and operation sequence of the lifter.

The definition literally says that the object "Lifter" consists of 4 elements. The loading and unloading one-directional conveyors are identical but turned in opposite directions. The corresponding models are of type **Conveyor**. The vertically moving platform (an object of type **StepMotor**) has a moving belt that moves pallets in both directions (modelled as an object of type **Conveyor2D**).

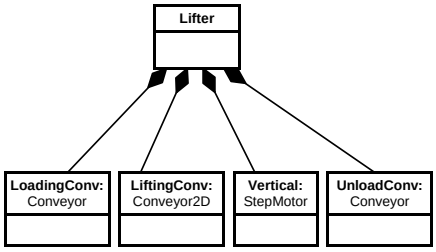


**Figure 3.** Definition of the model type (class) "Lifter" by means of UML class diagrams.

Note that the model in **Figure 3** does not define an interface of the lifter, nor dependencies between its constituent parts. These dependencies can be reflected in modular models by event and condition connections between the corresponding modules as exemplified in **Figure 4**.

Let us consider the model of a Conveyor. In our example two different types of conveyors are used – capable to move only in one direction, and those moving in both directions. The model of a more complex conveyor can be created based on the simple model using the mechanism of inheritance.

The interface of the model type "**Conveyor**" can be seen in **Figure 4**. The model itself can be conceptually divided onto three elements: Status, Position, and Sensor as shown in the class diagram in **Figure 5**, left. The Status element of type **MovingStatus** models the behaviour of the motor that drives the conveyor and converts the logic control signals into one of the states "Moving" or "Standing still" (that corresponds to the one-directional conveyor). Input "PRESENT" indicates if a pallet is present, and input "FORCED" is used to indicate influence of a neighbour belt on the movement of the pallet.

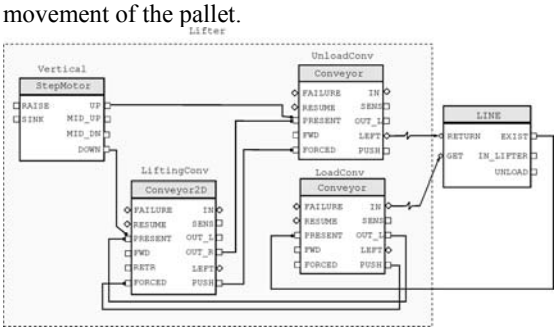


**Figure 4.** A model of Lifter represented as a network of NCES modules.

The output condition FW_ST is used by the model of belt position.

The structure of the model of the bi-directional conveyor is identical to that of the uni-directional. The difference is in the model **Status** that has type **MovingStatus2D** that inherits the interface properties of the one-directional **MovingStatus** and extends them with one more input and output for the retracted movement. This is shown in **Figure 5** (right). All transporters are equipped with a single position sensor indicating presence of the pallet (fully loaded on the conveyor).

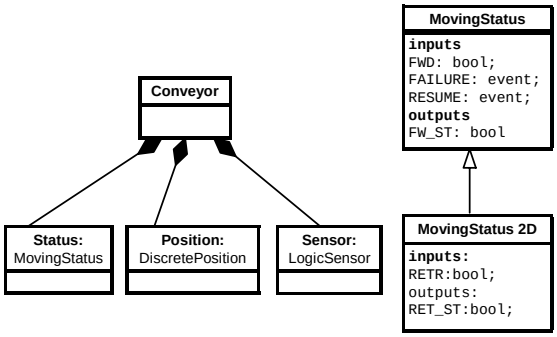


**Figure 5.** Model type definition of the conveyor and inheritance of the MovingStatus model types.

The data/event flow connections between the sub-models constituting the model of the conveyor are represented in **Figure 6**.

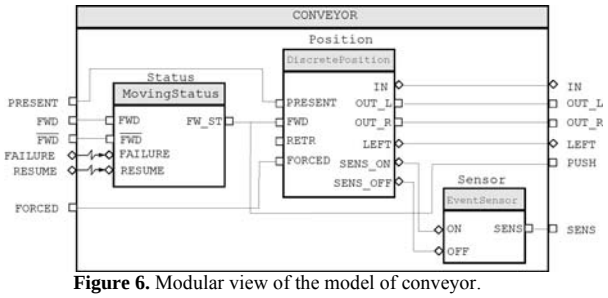


**Figure 6.** Modular view of the model of conveyor.

The basic models can be described further in form of NCES modules. **Figure 7**,**a** shows an implementation of the **MovingStatus** in NCES. The model receives the control signal FWD and transforms it into the state of the belt: place $p_2$ corresponds to the state "belt stands still", place $p_1$ – belt moves and $p_3$ to the state indicating a failure. The belt moves when the control signal FWD is ON, and stops when the signal goes OFF (i.e. negation of the signal FWD goes on).

An occurrence of a failure is indicated by an external event that may come from the corresponding model. For example, it can be a stochastic model of failures. Note that the model is sensitive to failures only when the belt moves, i.e. when the place $p_1$ is marked. It is assumed that the failure can be fixed by an external interaction indicated by event input RESUME.

The model **MovingStatus2D** for the bi-directional moving belt is shown **Figure 7**. It models one additional state for moving backwards, and correspondingly more transitions between all possible states.

The position of a pallet on the belt can be modelled with different precision. A qualitative model in **Figure 8** distinguishes only 3 states of a pallet on the belt: no pallet, pallet on the belt with its front edge between the belt's ends, and pallet's front edge is beyond the right end of the belt.

A more precise modelling of the position can be done using the arc-timed version of NCES. Let us assume that the belt is 3 units long and the pallet is two units long as shown in **Figure 9**. The speed of the belt is 1 unit of length per second. Then it will take 3 seconds for a pallet to reach the right end of the belt and 2 more seconds to leave the belt completely.

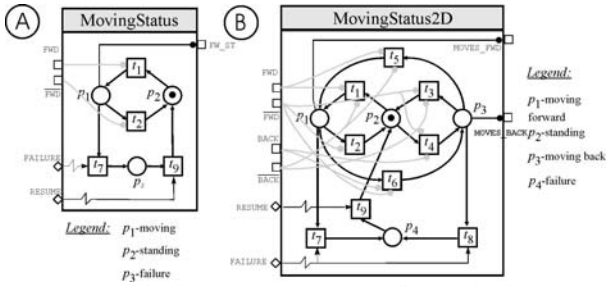


**Figure 7.** Models of the moving status for uni-directional and bi-directional belts.

Place $p_1$ corresponds to the state "No pallet". When a pallet appears (input condition "Present") and the state of the moving belt is "Moving forward" (indicated by the input condition FWD) then the transition $t_1$ occurs and the token goes to place $p_2$.

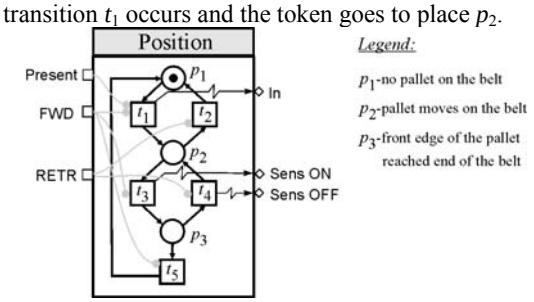


**Figure 8.** A qualitative non-timed model of the pallet's position**.**

This place indicates the state "Front edge of the pallet is in the interval 1 of the conveyor". Another reason to transfer to this state is the presence of the input condition "Forced". This condition indicates that the pallet is pushed onto the belt by some external force that maybe another moving belt positioned backwards to this one. This option is modelled by transition $t_{12}$. In general, moving in this case is slower than if driven by the own motor of the belt. The presented model, however, does not cover with enough precision the case when both forces are present simultaneously. Note that the transition from $p_1$ to $p_2$ (either via $t_1$ or $t_{12}$) is a qualitative one and does not take time (more precisely has zero delay).

The places $p_2$-$p_4$ correspond to the location of the pallet (again front edge!) in the intervals 1-3 respectively. A transition from interval $i$ to interval $i$+1 occurs in either case "FWD" and "Present" or "Forced" and "Present".

The latter, however, works only till less than the half of the pallet is on the belt – beyond this point friction would not let the pallet move driven only by the external force. The moving to the next interval takes 1000 ms if driven by the own motor of the belt or twice as long under the external force. The backward moving from interval $i$+1 to interval $i$ occurs if the combination of input conditions "RETR" and "Present" are true. It also takes 1000ms under assumption that the speed of the moving belt in both directions is the same.

Arriving of the pallet to the 3rd interval is indicated by the sensor. This is modelled by two event outputs "Sens ON" and "Sens OFF" associated with firing of transitions $t_8$ and $t_9$ or $t_{11}$, respectively. The sensor goes off when either the front edge of the pallet moves backward to the interval 2, or when the back

edge of the pallet leaves the belt in forward direction (and the pallet completely disappears from the belt).

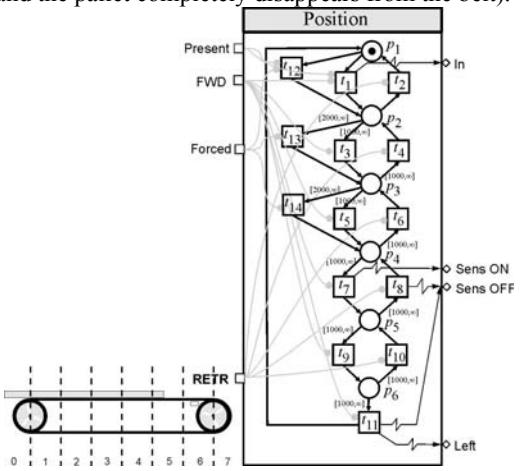


**Figure 9.** Model of the position of the pallet on the conveyor discretized on 3 intervals**.**

This model can represent the state of the pallet on the belt with better precision. However, it has its own limitations and drawbacks. In particular, let us consider how the alternative kinds of movement are modelled. A place indicating a position (e.g. p3 indicating interval 2) has several outgoing arcs (p3-t4, p3-t5 and p3-t14) marked with non zero time delays ([1000,∞], [1000,∞], [2000,∞]). Transitions that are targets of these arcs have condition input signals that represent alternative control signals (RETR, FWD, Forced). Either of transitions will fire when it is enabled by marking, conditions and time. It is important that all these conditions are mutually orthogonal (alternative) and they never change within the minimum delay of the place (1000ms in our case). Otherwise the model will not work as intended.

## 5 VALIDATION

The validation of automation systems modelled by NCES can be performed by simulation and formal verification via model checking.

Simulation usually follows a limited number of scenarios in the system's behaviour. In contrast, the model-checking studies multiple scenarios caused for example by some unpredictable factors, such as variable durations of some operations, communication delays, malfunctions, etc.

In the described framework, the model-checking can be conducted either by means of SESA, or by an embedded model-checker. In the latter case, the results of the model-checking, such as a reachability space (full, or generated until an example/counterexample is found) can be visualized as state/time diagrams of relevant values (e.g. represented as marking of certain places, or firing of certain transitions).

The verification consists in proving specifications with respect to the dynamic behaviour of the model. The specifications can be given either in form of second order predicates, or in form of temporal logic expressions. Terms of the expressions can be formed from inputs, outputs and internal variables of the controller or variables of the model of plant. The latter have to be eventually expressed via marking of places in the model.

More examples of formal verification can be found in (*Vyatkin V., Hanisch H.-M, 2003*). In the example mentioned in the previous section of particular interest were:

1. Robustness of the system in case of malfunctions of some sensors;
2. The control programs in VFL are branching. Formal verification helps to prove that the response time is never exceeded in any feasible IO combination.
3. Quality assurance: the lifter must never allow the situations when the pallet leans or jumps. It may be caused by inexact synchronization of conveyors' levels which may be a result of synchronization of control programs;

The overall model after assembly encountered 571 places and 828 transitions. However, the model-checking of a normal behaviour (without modelling malfunctions in sensors) resulted in the reachability space not exceeding 40000 states. This result reflects the efficiency of distributed state modelling with NCES.

## 6 GRAPHICAL SPECIFICATION

Although specification of properties for model-checking in temporal logic may be easily created by experts on verification, engineers would benefit of having user-friendly means of specifying the desired behaviour of a module. Inspired by timing diagrams, well-known in the hardware branch, a graphical language for describing the dependency of interface signal changes has been proposed (Vyatkin2001, Bouzon 2002).

A graphical description of a specification is illustrated in **Figure 10**. 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 semantic 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 at the signals does not violate the partial ordering specified at the diagram, until a final state is reached.

Translation and verification approaches differ slightly depending on whether the verified module has inputs. When verifying NCES (autonomous) modules, 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.

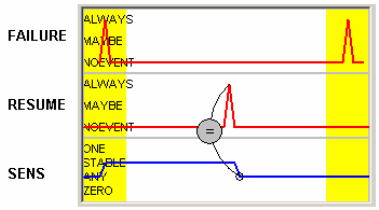


**Figure 10.** Specification including two event inputs, one condition output and a simultaneity operator.

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.

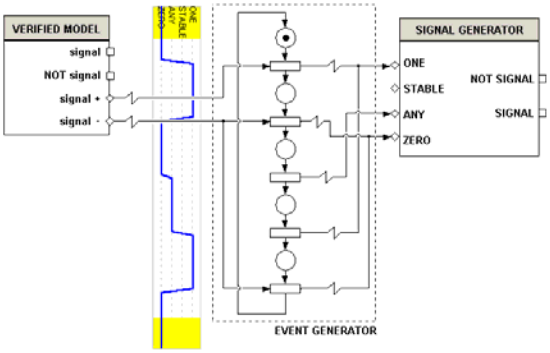


**Figure 11.** Translation of a single specification for a condition output, and linking to the verified model.

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.

## 7 CONCLUSION

This paper provided a short overview of an extended modular modelling paradigm which combines the ideas of object-oriented typed modelling of the mainstream UML with benefits of modular place-transition nets. Though the UML support is not integrated to the tool framework supporting NCES so far, however, the similarities with IEC61499 function blocks allow for using the CORFU environment (*Thramboulidis, 2002*) in connection with Rational Rose in order to define the model's structure and interfaces and convert them to the form of interconnected NCES modules. This approach allows for taking advantage of formal verification methodologies and tools with NCES.
Generation of NCES models from UML state charts will be a subject of future research.

## 8 ACKNOWLEDGEMENTS

The work was supported by the Deutsche Forschungs gemeinschaft under reference Ha 1886/12-2.
The closed-loop model of Lifter was developed following the methodology presented in this paper (Section 4) in MOVIDA-1, a project funded by the National Technology Agency in Finland -TEKES. (A. Lobov, et al.).
The authors thank Reijo Tuokko, Jose L Martinez Lastra and Andrei Lobov from Tampere University of Technology (TUT), as well as all MOVIDA-1 partners, for providing the opportunity to apply the object-oriented version of NCES to real automated equipment and for fruitful discussions on the presented modelling approach.

## 9 REFERENCES

H.-M. Hanisch and A. Lüder: *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

P. Starke, S. Roch, K. Schmidt, H.-M. Hanisch, A. Lüder: Analysing signal-event systems, Technical report**,** *Humboldt Universitat zu Berlin*, Institut für Informatik, http://www.informatik.hu-berlin.de/lehrstuehle/automaten/tools/, July, 1999

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

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

*IEC 61499 - Function Blocks for Industrial Process Measurement and Control Systems*. Publicly Available Specification, International Electrotechnical Commission, Tech. Comm. 65, Working group 6, Geneva, 1998.

Vyatkin V., Hanisch H.-M., Pfeiffer T., "*Modular typed formalism for systematic modelling of automation systems*", 1st IEEE Conference on Industrial Informatics (INDIN'03), Proceedings, Banff, Canada, August 2003

Vyatkin V., 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*

Vyatkin V.: *Intelligent Mechatronic Components: Control System Engineering using an Open Distributed Architecture*, IEEE Conference on Emerging Technologies in Factory Automation (ETFA'03), Proceedings, Lisbon, September, 2003

M. Bonfè and C. Fantuzzi: *Design and Verification of Industrial Logic Controllers with UML and Statecharts*, submitted to the IEEE Conference on Control Application 2003, June 23-35, Istanbul, Turkey

*FBDK - Function Block Development Kit at www.holobloc.org*

K. Thramboulidis: *Development of Distributed Industrial Control Applications: The CORFU Framework*, 4th IEEE International Workshop on Factory Communication Systems, August 2002, Vasteras, Sweden

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

G. Bouzon: *Development of a visual specification language for verification of Distributed Controllers*, Final Paper (Bachelor Degree), Dept. of Automation and Systems, Federal University of Santa Catarina, Florianópolis, May 2002 (in portuguese)

A. Lobov, J. L. Martinez Lastra, R. Tuokko, V. Vyatkin: *Methodology for Modelling Visual Flowchart Control Programs using Net Condition/Event Systems Formalism in Distributed Environments*, IEEE Conference on Emerging Technologies in Factory Automation (ETFA'03), Proceedings, Lisbon, September, 2000