Modeling and Verification of Distributed Control Systems

Hans-Michael Hanisch and Valeriy Vyatkin
Martin Luther University of Halle-Wittenberg, Dept. of Engineering Sciences
D-06099 Halle, Germany
Email: {Hans-Michael.Hanisch|Valeriy.Vyatkin}@iw.uni-halle.de

1 INTRODUCTION

The field of automation technology is an engineering discipline that is highly interactive and covers subjects like manufacturing and process systems, electrical and computer engineering, and computer science. The major responsibility of an automation engineer is to design and implement a control system that interacts with the object of control in a closed loop and that ensures that the controlled object behaves safely and efficiently. Although automation technology is deeply influenced by information technology in that sense that the control system itself is an information processing system, automation technology is not identical with information technology. The major point of concern in automation technology is the object that is controlled, and the control system serves only as a means to reach the goals coming from the controlled object.

Therefore, any keen and scientific methodology for design and verification of control systems must take the behavior of the controlled object, along with its distributed and hierarchical structure, into consideration. Having only a model of the controller is in general not sufficient to prove the correctness of the specifications. We therefore focus in this contribution on a methodology that includes models of the controller as well as models of the controlled object.

This contribution will give an outline of an appropriate methodology for design and verification of distributed systems in control. It is organized as follows.

Section 2 will focus on some basic problems that come from the particular context of automation engineering. Section 3 will discuss the current transition from centralized to distributed control systems. As a result, we will present a verification methodology for distributed systems in Section 4. Finally, we will draw some conclusions on what we have so far and will give an outlook to future work in this field.

2 PROBLEMS IN CONTROL

As mentioned in the introduction, the design of a control system is driven by goals that come from outside, namely from the object that is to be controlled. We call this object "plant". A plant in the cases we are interested in is a manufacturing system or even a process system with some kind of hybrid (discrete and continuous) characteristics. Such systems are designed and operated by engineers who are not experts in automation technology. They, however, define the goals and the behavior of the plant that must be realized by means of control. Hence, the specifications of the desired or forbidden behavior of the plant are given in terms of the plant characteristics and not in terms of the control system. They are usually written down in textual form, and there is no guarantee at all that they are complete, exact and free of contradictions.

Figure 1 shows as an example a part of a manufacturing system in laboratory scale. The purpose of the testing station is to determine the material characteristics of a workpiece, check the workpiece height and either reject a workpiece or make it available to a subsequent station. Main parts of the testing station are the detection, lifting, measuring, and slide module.

The workpiece is transferred by a pneumatic handling device to the detection module. Materal/colour detection is effected with the help of 3 proximity sensors with digital output. The lifting module lifts the workpiece from the material detection module to the measuring module. A lifting cylinder and an ejecting cylinder are used as actuators. End position sensing is effected via magnetic or inductive proximity sensors.

The slide module is used for transporting workpieces with correct characteristics to the next station. A pneumatic stopper is mounted at the upper slide. By means of this stopper the workpiece is stopped before it is transferred to the subsequent station. If a workpiece with incorrect characteristics is detected, it is moved back to the bottom and is ejected.

One clearly sees that everything that has to be performed to run the system in the desired way is formulated in terms of the plant and its behavior an not in terms of a controller behavior.

The task of the control engineer is to understand these specifications and to design a control system that controls the plant in such a way that the specifications are fulfilled. It is more than obvious that such an error-prone way of designing a system will by no means result in a predictable behavior of the plant. Hence, time- and cost-intensive tests are usual.

Things are getting even more complicated if distributed systems have to be designed. The transition from the sequential, time-driven execution model of today's Programmable Logic Controllers towards the distributed, concurrent and event-driven execution marks a change of paradigms in the whole engineering process. Up to now, there is very little experience how to design systems in such a new way. All these particular aspects give rise to the need to have some formal methodology at hand that would support the engineering process of distributed control systems.

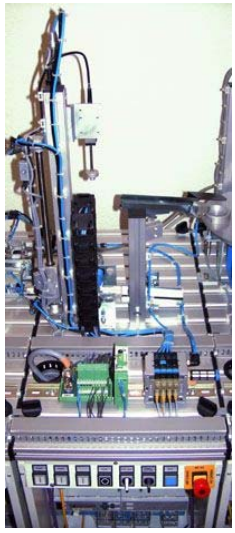

Figure 1: Example

3 RECENT DEVELOPMENTS IN CONTROL SYSTEMS

Today's daily practice in controller development is dominated by the International Standard IEC 61131 [IEC1131]. It has reached the end of its technological lifecycle. There are very few means to design programs in a way that reflects the new developments in software engineering. The centralized, time-driven execution of the control code does not sufficiently support encapsulation, modularity, and distribution. The newly emerging International Standard IEC 61499 [IEC1499,

Lew01, Hlbl] is an attempt to overcome these limitations and to provide concepts for specification and implementation of distributed control systems. It is therefore based on encapsulation of control software and on event-driven execution mechanisms. The basic software pattern is a so-called Function Block.

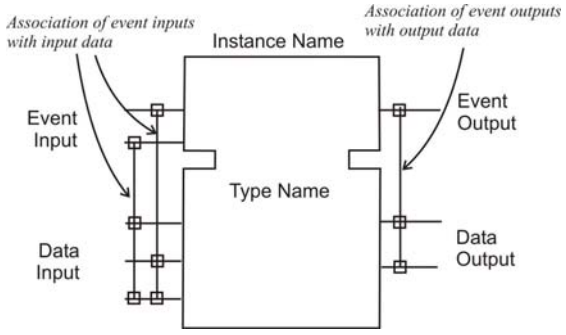

Figure 2. Function block interface.

Figure 2 shows the interface of a Function Block. The standard clearly distinguishes between events that trigger the execution of algorithms and associated data that are inputs or outputs of the algorithms. The algorithms themselves are encapsulated in the Function Block and hidden to its environment. Their execution is triggered by events coming from the environment of the Function Block. The Function Block notifies its environment about termination of the algorithm's execution by sending events as well. By this mechanism, the environment is informed that the output data of the Function Blocks are valid and can be used for further processing.

The standard also defines Composite Function Blocks, whose functionality, in contrast to Basic Function Blocks, is determined by a network of interconnected function blocks inside. Figure 3 shows the principle.

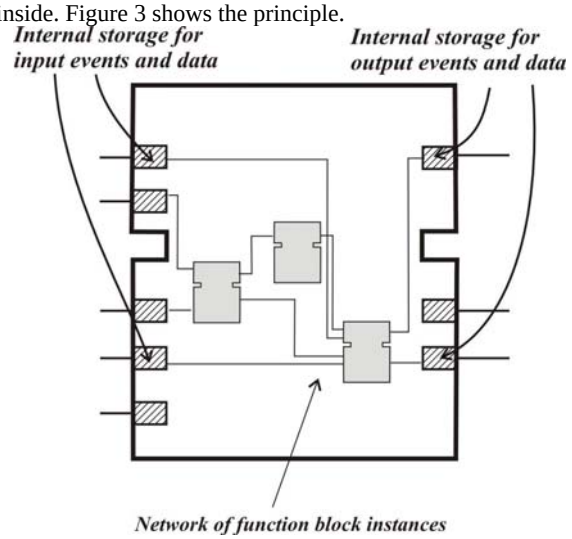

Figure 3. Composite Function Block

More precisely, members of the network are instances of function block types. These can be either Basic Function Blocks or other Composite Function Blocks. Therefore, hierarchical applications can be built. The functionality of Composite Function Blocks completely depends on the behavior of the constituent function blocks and their interconnections by events and data.

An *application* following the IEC 61499 is a network of function block instances whose data inputs and outputs and event inputs and outputs are interconnected (see Figure 4).

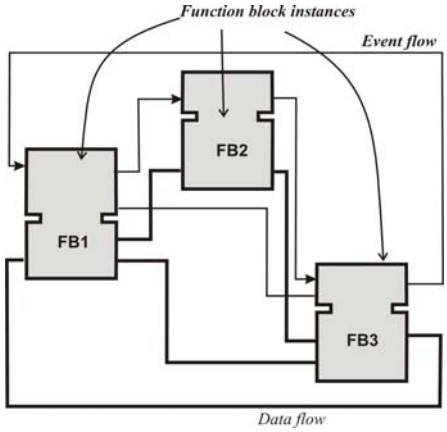

Figure 4. An application.

An application can be considered as an intermediate step in the system development. It already defines the desired functionality of the system completely, but it does not specify the system's structure in terms of computational devices where the function blocks can be executed. The following step in the engineering process is to define a particular set of devices and to "cut" the application, assigning the blocks to the devices as illustrated in Figure 5.

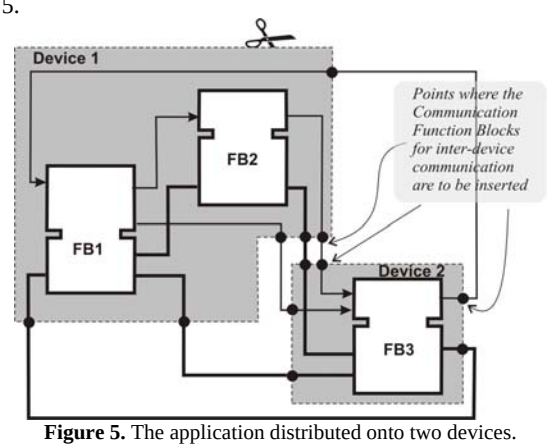

Figure 5. The application distributed onto two devices.

The way how the separated parts of the distributed applications communicate to each other has to be explicitly defined. This can be done by adding Communication Function Blocks in the places where the "cut" took place. A brief introduction to the concept together with an example can be found in [HaVy04].

Up to now, very few attempts have been made to formally model Function Blocks as they are defined in the standard [VyHa03, BoFa03, StaGu03,Wurmus00].

We will focus here only on the function blocks that form the application since we are interested in formally verifying its logical correctness. Verifying the correct runtime behavior of an application after mapping it to a particular system architecture will be also required, but this goes beyond the scope of this contribution.

4 MODELING AND VERIFICATION METHODOLOGY

4.1 Modular Models

There are some specifics in the control area regarding the models that are used.

Models should be graphical and executable. This is especially needed for validation and for gaining acceptance of the engineering staff.

Models must be modular to clearly identify the parts of the original system they describe.
Models must be compositional to support a way of engineering that constructs bigger systems from smaller subsystems.
We use a modeling formalism that were called NCES or SNS in the past [HaLu99, HaLu00, Thi02].
They provide means for modular, graphical modeling in a way that is intuitively understandable by any engineer. The basic patterns are modules or blocks that encapsulate some kind of dynamic behavior model. Modules can be interconnected by signals. This reflects the standard paradigm in automatic control.
In contrast to modeling of continuous systems where the variables take their values from a subset of the real numbers, our models describe systems with discrete states and discrete state transitions. This means that any executable modeling formalism for such systems may be used for modeling the internal dynamic behavior of a module. We use some graphical notation that was borrowed from the graphical appearance of Petri nets since they provide nice graphical means for clearly distinguishing between states and state transitions and for expressing conflicts as well as concurrency.
The more interesting question is how the signals are defined that are used for interconnecting the modules. We use two types of signals, namely condition signals carrying state information and event signals carrying state transition information. Therefore, the signal concept is complete in that sense that anything what can happen inside a module (states as well as state transitions) can be expressed via appropriate signals.
The interconnection of modules by means of signals forms the structure of the system. The structure is static, i.e. it is not changed during the dynamic execution of the system.
It is obvious that the dynamic behavior of the system is determined by its modules as well as by its structure. The dynamic behavior can be studied by simulation (execution of the model) or even by analysis (mostly based on complete state space enumeration). Formalisms and tools for both purposes are available.

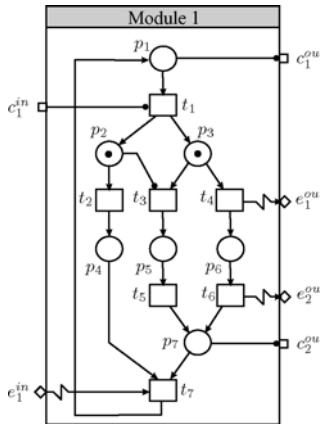

Figure 6. A module

Consider, as an example, the module shown in Figure 6.
Input and output signals are attached to the module boundary. Condition inputs and outputs are represented by little square symbols, and event inputs and outputs are represented by diamond symbols. Condition signals themselves are represented by dotted arcs whereas event signals are represented by zigzag arcs.
The internal dynamic behavioral model is drawn as usual in Petri nets with some extensions (see for example test arc (p_2, t_3) in Figure 6).
The internal behavior of the module is (in general partially) controlled by input signals from the environment of the module and is observed (in general only partially as well) by the environment via the output signals of the module.
In our example, transition t_1 can be enabled/disabled by condition input c_1^{in} if t_1 is enabled by marking, and transition t_7 can be forced to occur via event input e_1^{in} if t_7 is enabled by marking. On the other hand, the marking of places p_1 and p_7 can be observed via condition output signals c_1^{out} and c_2^{out}, and the occurrence of transitions t_4 and t_6 can be observed via the corresponding event output signals e_1^{out} and e_2^{out}. The rest of the behavior of the module cannot be observed.
The behavior of the module is as follows:
Transitions t_2, t_3 and t_4 are enabled at the marking shown in the figure. Transitions t_3 and t_4 are mutually in conflict. Transitions t_2 and t_3, however, are neither in a classical conflict nor can they fire concurrently due to the condition arc (p_2, t_3). This means that t_3 would be disabled if t_2 fires, but firing of t_3 would not disable t_2. Anyway, the system would end up in the state where p_4 and p_7 are marked and condition output e_2^{out} is true, either issuing event output signal e_1^{out} and afterwards event output signal e_2^{out} or being "silent". Although transition t_7 is enabled by marking, it can and must occur only in the case when event input e_1^{in} is true. This would lead to the state where the only token in the system is on place p_1 and therefore condition output c_1^{out} is true. Transition t_1 may fire in this case if c_1^{in} is true, but it is not forced to fire.
The module in Figure 7 is completely controllable. Hence, it shows no autonomous behavior without condition and event signals from its environment.
The computation of the state space for non-autonomous models requires an assignment of values to the input signals. This is called input status. Any combination of values of input signals needs to be taken into account since the behavior of the environment is not known. If we compute the state space for closed-loop models, as it is the case in our verification environment, the models are in almost all cases autonomous. Exceptions could be a small number of input signals that come from a human operator. A method how to replace them by small models is provided in [HaLu00]. The fact that the models are autonomous makes verification much easier since the number of states that are reachable in closed-loop behavior is by magnitudes smaller than in the open-loop behavior of the components. This makes verification of realistic closed-loop systems feasible.

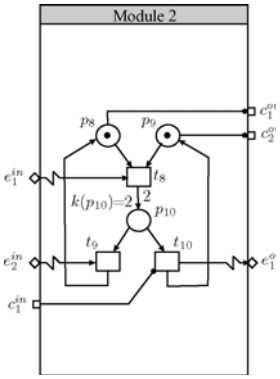

Figure 7. Another module

The model modules are interconnected in the same way as the Function Blocks in the original system, namely by means of event and condition signals. Figure 8 provides an example of the interconnection of module 1 and module 2. This is done by composition arcs that interconnect inputs with outputs of the same type.
The composition establishes a behavior of the composed system that differs from the behavior of the isolated modules. The interconnection of transitions in the composed system via event signal results in a one-sided synchronization among transitions. This means the following. If a transition fires that emits an event signal to another transition, the receiving transition is forced to fire together with the emitting transition if the receiving transition is enabled by marking. If this is not the case, the emitting transition fires alone. Hence, enabled transitions that are interconnected via event signals form steps.

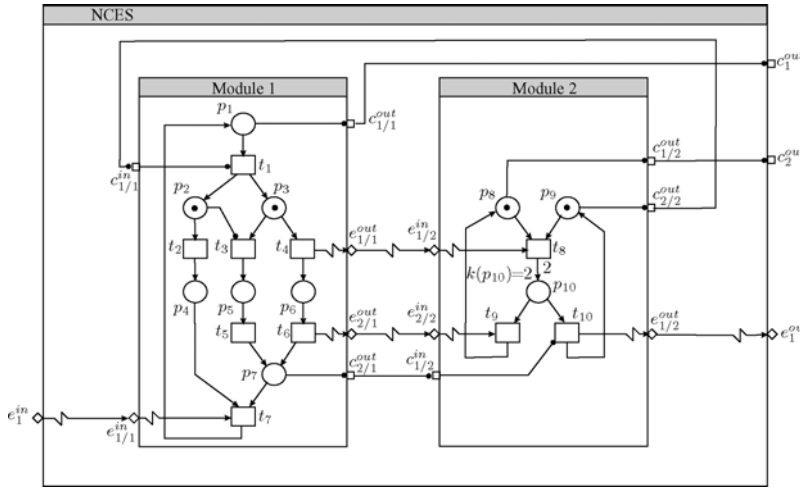

Figure 8. Composition of modules.

The modeling formalisms, composition and firing semantics are formally defined in [Thi02]. This would go beyond the scope of this contribution. We can see, however, how the composed model in Figure would be executed. In our example steps $\{t_2\}$,$\{t_3\}$ and $\{t_4,t_8\}$ are enabled. Firing $\{t_3\}$ would disable $\{t_4,t_8\}$ whereas $\{t_2\}$ and $\{t_4,t_8\}$ can fire concurrently (in any arbitrary order). Note that transition t_8 cannot fire without transition t_4, but it *must* fire if it is enabled and t_4 fires. Firing $\{t_4,t_8\}$ would remove the tokens from places p_3 , p_8 and p_9 and would put one token at p_6 and two tokens at p_{10}. This would enable step $\{t_6,t_9\}$. If it would fire, it would remove a token from p_6 and p_{10} and put one token to p_7 and to p_8. Eventually, after firing $\{t_2\}$, the system would end up in a state where places p_4, p_7, p_8 and p_{10} carry one token. Hence, steps $\{t_7\}$ and $\{t_{10}\}$ are enabled. Note that $\{t_{10}\}$ is not forced to fire but $\{t_7\}$ must fire at the moment the input event $e_{1.}^{in}$ occurs. If input event $e_{1.}^{in}$ occurs before transition t_{10} fires, the system would deadlock. In the other case, the system could eventually come back to its initial state.

This and other properties can be checked by generating the state space of the system. Additionally, we are able to define the input/output language of the composed system as well. This provides means for state reduction, but this goes beyond the scope of this paper.

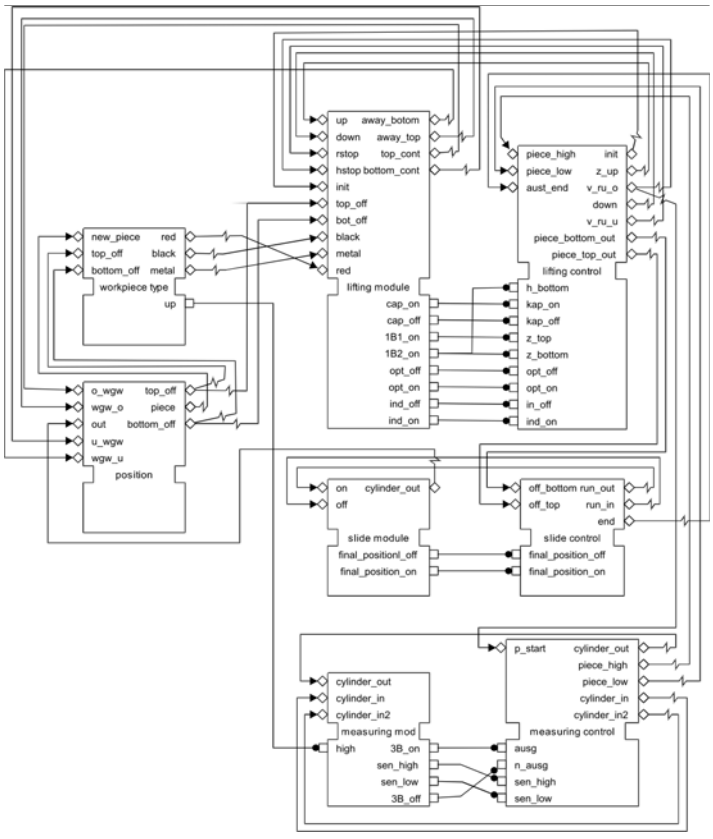

Figure 9. An example of interconnected model.

4.2 Modeling of the Controller

If we compare the modeling formalism with the IEC 61499 specification of Function Blocks, we see some strong similarities:

- dynamic behavior is encapsulated in the blocks and hidden to their environment
- blocks resp. modules have a signal interface that clearly distinguishes between data (resp. conditions) and events.

The structure of the system is given by the interconnection of blocks by means of signals.

The dynamic behavior of the system is given by the behavior of the modules and the structure, i.e. the interconnection of blocks or modules.

It is therefore very natural to map Function Blocks to formal modules and to interconnect them exactly in the same way as the Function Blocks are interconnected in the application. Due to this strong similarities, a mapping of a Function Block application to a formal model with identical structure could be defined and implemented. That has been done and integrated in the tool (see Subsection 4.5).

4.3 Modeling of the Plant

Such a model is, however, only half of the truth since it reflects only the behavior of the controller. As we have seen in Section 2, we need a model of the plant behavior as the counterpart of the controller and a basis to define specifications of the desired or forbidden behavior of the plant.

When modeling continuous plants, the basis for any model are conservation laws of physics, chemistry etc. Unfortunately, this is not the case when modeling discrete behavior of plants. Since we cannot solve the problem how to develop models for systems with discrete states and state transitions in such a concise and clean way as models for continuous systems are developed, we try at least to propagate a methodology that is not completely based on exact sciences (which is impossible at the moment), but reflects at least some of the common principles that are used in the engineering disciplines when models are designed.

The model of the plant is engineered by predefined components that are stored in a repository of models. The modularity of the modeling approach is helpful for this. Therefore, the modeling process does not need to be started from scratch. Predefined and validated component models are encapsulated in modules and can be used over and over again without dealing with their internal details.

4.4 Modeling the Closed-Loop Behavior

Having both models, the model of the controller and the model of the plant, the model of the closed-loop system is formally constructed by interconnecting the input/output signals of the controller model with the input/output signals of the plant model in the same way as the real controller would interact with the real plant.

As a result, we end up with a model of the closed-loop behavior that is as close as possible to reality. This is especially useful for finding and correcting design errors that are detected by formal verification.

Figure 10 shows the closed-loop model for the plant described in Section 2. The two left colums of blocks represent the characteristics of the workpiece as well as the behavior of the plant equipment. The blocks at the right side represent the controllers. One clearly sees the feedback loops from the controllers to the plant equipment and from the equipment to the workpiece position. The internals of the blocks are built from submodules in a hierarchical way.

Our experience gained within preceding applications [Hanisch et al 01, Lobov et al. 03, Lobov et al. 04] shows that it is highly desirable to use models that may constitute a one-to-one mapping of the structure of the manufacturing process onto the structure of the corresponding model. Coming up with large monolithic models is almost infeasible due to the size and complexity of the original system.

4.5 Software Tools

The validation of automation systems modeled 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 behavior. 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.

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 by referencing 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.

It is worth mentioning that the closed-loop approach to the modeling enables expression of the specifications directly in terms of the machine behavior (not only input and output signals of the controller).

In particular, the following properties of automation systems could be scrutinized applying the formal validation techniques:

- Robustness of the system in case of malfunctions of some sensors;
- The control programs in some programming languages (e.g. Structured Text and Sequential Function Charts) have branching structure. Formal verification may help to prove that the response time is never exceeded in any feasible IO combination.
- Quality assurance: sometimes it is important to ensure that the plant never occurs in undesirable situations, when, for example, inexact synchronization of processes in the plant occurs as a result of wrong synchronization of control programs;

A framework of tools and methodologies has been developed in our lab to facilitate the NCES-based modeling and verification of automation systems by control engineers. The framework is presented in the following Figure 10.

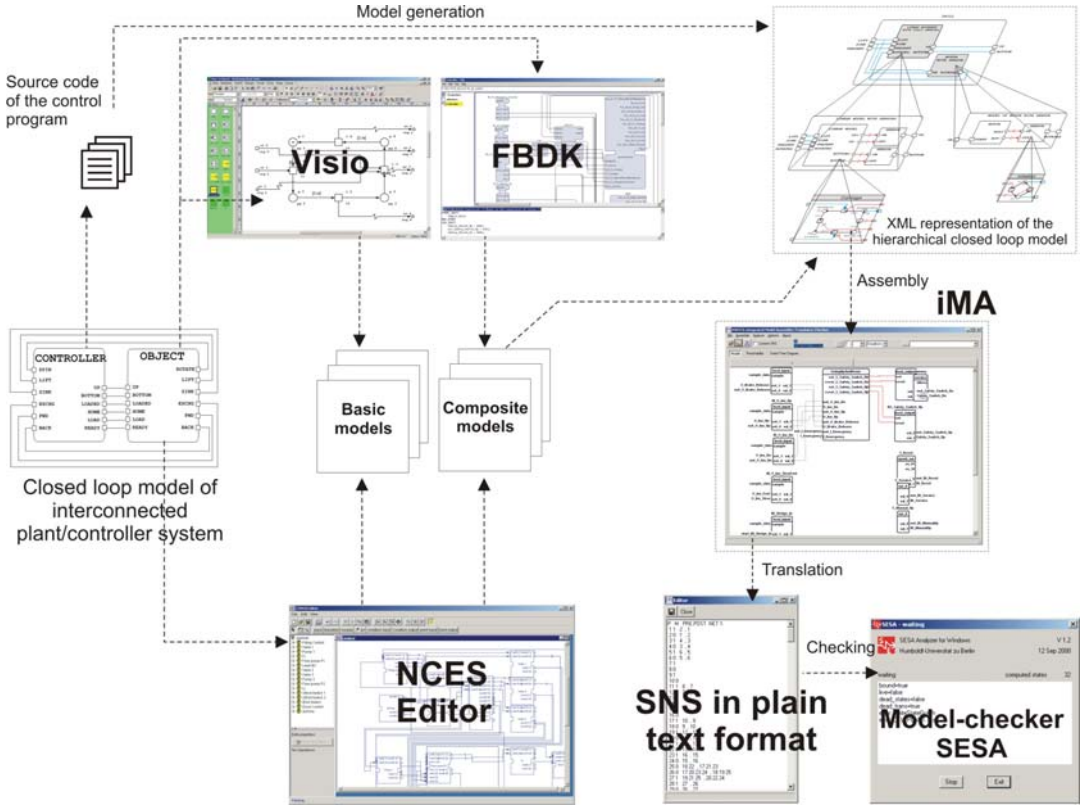

Figure 10. Framework of tools and methods supporting the NCES-based formal modeling and verification of automation systems.

Graphical editors providing 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 the tool *FBDK* [Hlbl].

iMA – an integrated tool that contains a model builder (assembler), a translator to the flat format for subsequent model-checking, interfaces to several model-checkers, and the means for analysis of scenarios (e.g. their visualization in form of state/time diagrams), or even system simulation along the selected scenarios. (iMA) inputs the model type files given in XML and is capable of:

Assembling a composite, hierarchically organized model from modules contained in different libraries. The component model types are instantiated into NCES modules.

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.

The model checker **SESA** allows for efficient model-checking of fairly complex systems (millions of discrete states);

The application methodologies are represented as libraries of standard model elements and by the web-based documentation;

The formalism is open – an XML based data format allows the development of add-ons to the existing tools, for example model-generators for particular programming languages representing control algorithms.

More details on this framework can be found in [VHP03, VHB04]

5 CONCLUSION AND FUTURE WORK

What we have so far are a methodology and a prototype software tool for demonstration by means of first application examples. Future applications of the standard are in sight and give some hope that at least parts of the methodology may be applied in practice and not only in academia.

Another major future issue that is closely related to the development of the standard as well as of the methodology presented in this contribution is the development of reconfigurable systems in manufacturing. The need to have such highly flexible systems with dynamic structure will surely give rise to broader application of modern methods of system design and validation.

6 ACKNOWLEDGEMENTS

This work was supported in part by the cooperative project VAIAS funded by the German Ministry for Education and Research (BMBF).

7 REFERENCES

[BoFa03] M. Bonfe, C. Fantuzzi: Design and verification of mechatronic object-oriented models for industrial control systems, IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03), Proceedings, vol. II, pp.253--260, Lisbon, September, 2003

[Hanisch et al 01] H.-M. Hanisch, T. Pannier, D. Peter, S. Roch and P. Starke: Modeling and verification of a modular lever crossing controller design. Automatisierungstechnik, 2000

[Ha04] H.-M. Hanisch: Closed-Loop Modeling and Related Problems of Embedded Control Systems in Engineering, 2004, in Abstract State Machines 2004, LNCS 3052

[HaLu99] 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

[HaLu00] H.-M. Hanisch und A. Lüder: A Signal Extension for Petri Nets and its Use in Controller Design, Fundamenta Informaticae. Nr.4, March 2000, pp. 415 – 431.

[HaVy04] H.-M. Hanisch and V. Vyatkin: Achieving Reconfigurability of Automation Systems by Using the New International Standard IEC 61499: A Developer's View, The Industrial Information Technology Handbook, CRC Press, October 2004

[Hlbl] http:// www.holobloc.com – web site devoted to IEC61499.

[IEC1131] International Standard IEC 1131-3, Programmable Controllers - Part 3, International Electrotechnical Commission, 1993, Geneva, Switzerland

[IEC1499] Function Blocks for Industrial Process Measurement and Control Systems, Publicly Available Specification, International Electrotechnical Commission, Part 1: Architecture, Tech. Comm. 65, Working group 6, Geneva, 2002

[Lew01] R. Lewis: Modeling Distributed Control Systems using IEC 61499, Institution of Electrical Engineers; London, 2001, ISBN: 0852967969

[Lobov et al. 04] A. Lobov, J. LM Lastra, R. Tuokko, V. Vyatkin: *Modelling and Verification of PLC-based Systems Programmed with Ladder Diagrams*, INCOM'2004, Proceedings, Salvador, Brazil, April, 2004

[Lobov et al. 03] A. Lobov, J. L.M. Lastra, R. Tuokko, V.Vyatkin, "Methodology for Modeling Visual Flowchart Control Programs using Net Condition/Event Systems Formalism in Distributed Environments", IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03), Proceedings, Lisbon, Portugal, September 2003, ISBN 0-7803-7937-3

[StaGu03] M. Stanica, H. Guéguen, "A Timed Automata Model of IEC 61499 Basic Function Blocks Semantic", ECRTS'03 Euromicro European Conference on Real-Time Systems, Porto, Portugal, July 2003

[Thi02] J. Thieme: *Symbolische Erreichbarskeitanalyse und automatische Implementierung struktuirter, zeitbewerter Steuerungsmodelle*, Dissertation zur Erlagung des Grades Dr.-Ing., Berlin: Logos Verl., 2002

[VHP03] V. Vyatkin, H.-M., Hanisch, T.Pfeiffer, "Modular typed formalism for systematic modeling of automation systems", 1st IEEE Conference on Industrial Informatics (INDIN'03), Proceedings, Banff, Canada, August 2003, ISBN 0780382005

[VHB04] V. Vyatkin, H.-M. Hanisch, G. Bouzon: *Open Object-Oriented Validation Framework For Modular Industrial Automation Systems*, INCOM'2004, Proceedings, Salvador, Brazil, April, 2004

[VyHa03] V. Vyatkin, H.-M. Hanisch: Verification of Distributed Control Systems in Intelligent Manufacturing, Journal of Intelligent Manufacturing, vol.14, N.1, 2003, pp.123-136

[Wurmus00] H. Wurmus, B. Wagner,: «IEC 61499 konforme Beschreibung verteilter Steuerungen mit Petri-Netzen»; Conference Distributed Automation 2000 (Verteilte Automatisierung), Institut für Automation und Kommunikation e.V., Magdeburg 2000