Formal Modeling and Verification in the Software Engineering Framework of IEC61499: a Way to Self-verifying Systems

Valeriy Vyatkin, Hans-Michael Hanisch
Dept. of Engineering Science,
Martin Luther University Halle-Wittenberg,
D-06099 Halle, GERMANY
e-mail: Valeriy.Vyatkin@iw.uni-halle.de
Hans-Michael.Hanisch@iw.uni-halle.de

Abstract - **This paper discusses the issues related to the correctness of agile manufacturing systems with distributed architectures. The corresponding development of software engineering methods targets the goal of easy-re-configurable software self-organized similar to that of the hardware. However, the existing methods of software validation (manual testing or computer-aided simulation) are too slow to keep up with a pace of reconfigurations. In this paper we present an approach and a software tools to incorporate the formal verification to the practice of control engineering. The software package "Verification Environment for Distributed Applications" (VEDA) has been developed for model-based simulation and verification united by a homogeneous graphical user interface. Net Condition Event Systems (NCES) are used for modeling. VEDA deals with distributed controllers as defined in IEC61499 and automatically generates the formal model of the controller given its source code.**

I.INTRODUCTION

The state of the art in design of industrial automation systems is characterized by massive substitution of centralized control systems by distributed network-based ones. Along with other benefits (such as cost-efficiency and reliability), the latter provide means for real agility of the production equipment, such as fast, easy and robust reconfiguration. As it is shown in Figure 1, the distributed control system can be easily upgraded by corresponding components in case that the production line is appended by a new device (measurement station). The bottleneck on the way to real agility is software engineering which lags far behind the abilities of the hardware. Definitely, using currently existing methods of software design and testing, the lion share of time is spent on development or modification of the control software components for the re-configured production equipment.

The corresponding development of software engineering methods targets the goal of easy-re-configurable software, which could be self-organized similar to that of the hardware. The mottoes of those concepts are "Open Architectures", "Integration without Master Control", "Plug-and-Play software modules", etc. One of the latest efforts in this direction is the development of the new IEC standard 61499 [1, 2] which provides a uniform vendor-independent solution for programming of distributed control systems by integration and extension of current standards IEC 61131-3 (Programming languages for Programmable Logic Controllers) and IEC 61804 (Function Blocks for Distributed Control Systems). Representation of the systems according to the standard helps to focus on the essential issues, regardless of the variety of existing hardware and software solutions and network protocols. In the framework of IEC61499 the controller can be understood as its software code that can be tested (at least partially) without knowledge of implementation details.

The basic programming structure of IEC 61499 is a function block with event and data inputs and outputs. The block is conditionally divided onto "head" responsible for the execution logic, and "body" which contains algorithms of data processing. The result of this change of paradigm is that there is no longer a strictly sequential execution of the control program. Each function block has its own event-driven execution control, and the complete execution control of a system of composed function blocks is physically as well as functionally distributed.

An application in the IEC61499 is a net of function blocks interconnected via data and events. The blocks of the same application may be distributed physically over different devices. Thus, the control logic can be designed without knowledge about particular architecture of the control system where it is to be executed. The architecture can be described later, or modified independently of the core description of the application.

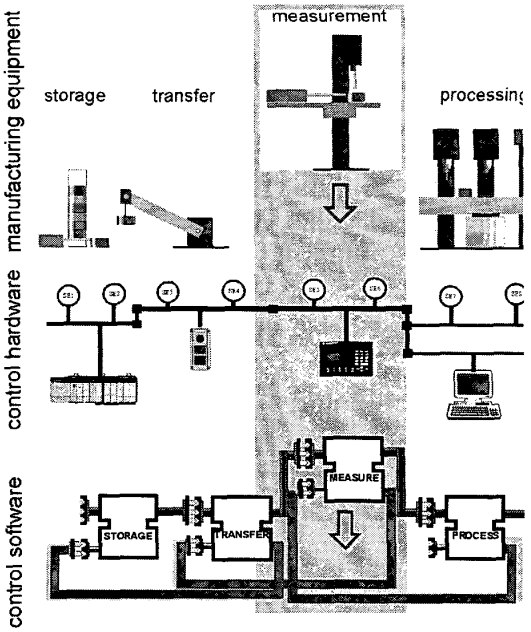

Figure 1. Reconfiguration of the production equipment is easier with distributed control architectures and software engineering concept of "Plug-and-Play function blocks".

0-7803-7241-7/01/$10.00 (c)2001 IEEE

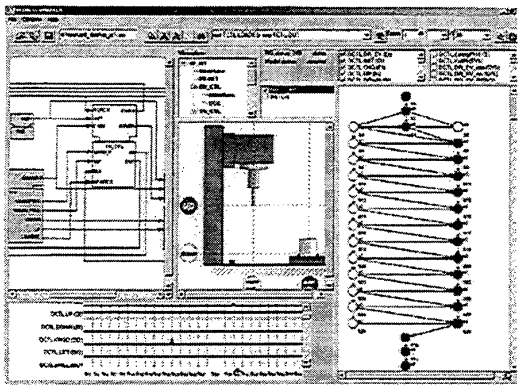

Figure 2. VEDA provides the source code - based verification and simulation of IEC61499 control applications.

II. FORMAL METHODS – SUCCESSES AND SHORTAGES

Requirements to the software engineering concepts, appropriate for application in agile manufacturing control systems include as an important part adjustment and improvement of the methods of testing. The existing methods of testing (manual or even based on computer-aided simulation) are too slow to keep up with the pace of reconfigurations. Taking in account the complexity of the systems under control, the provided level of assurance in error-detection is also far from being satisfactory.The successes in development of formal methods create new opportunities for improvement of the testing routines. Thus, formal verification of the control systems, proposed and actively studied in academia, is getting closer to the reality of control engineering.

Since more than two decades back, when the verification of control systems was proposed in works [3, 4], its common pattern is as follows. Both uncontrolled plant and controller are modeled using a finite state or hybrid formalism. The models are interconnected the same way as a controller is interconnected with the plant, forming thus a model of the closed-loop systems. It is important to keep separated the models of plant and controller in order to provide means of independent controller or plant modification. Specifications of desired or forbidden behavior are formalized, converted to the terms of the model and finally checked with respect to the model using a program tool, called model-checker. Despite of the theoretical elegance and clarity of this framework, there are some reasons preventing wide penetration of the verification to the practice of control engineering. The most essential are as follows:

- Formal modeling is not easy for engineers and is not integrated in the current software engineering practice. E.g. testing and simulation require models different from the models used for verification.
- Modeling of interconnected plant/controller systems requires formalisms equally good for both plant and controller.
- Formulation and formalization of specifications are tricky.
- The model-checking is computationally complex, and is not always supported by a proper Human - Machine Interface.

Overcoming these difficulties was the aim of developing the Verification Environment for Distributed Applications (VEDA) [5,6], that is a software package for model-based simulation and verification united by a homogeneous graphical user interface (Fig.2). VEDA deals with distributed controllers as defined in IEC61499 and automatically generates the formal model of the controller given its source code. The formal modeling is performed using Signal-Net Systems (SNS) [7,8]. VEDA includes facilities to develop models of plants such as an editor for SNS models. VEDA allows to build the closed-loop models of plant and controller, and to prove formally whether the overall behavior of the system satisfies the properties of desired/undesired behavior. The analysis of the verification results in VEDA is supported by simulation and visualization of the process along selected trajectories that helps to accelerate understanding the reasons of

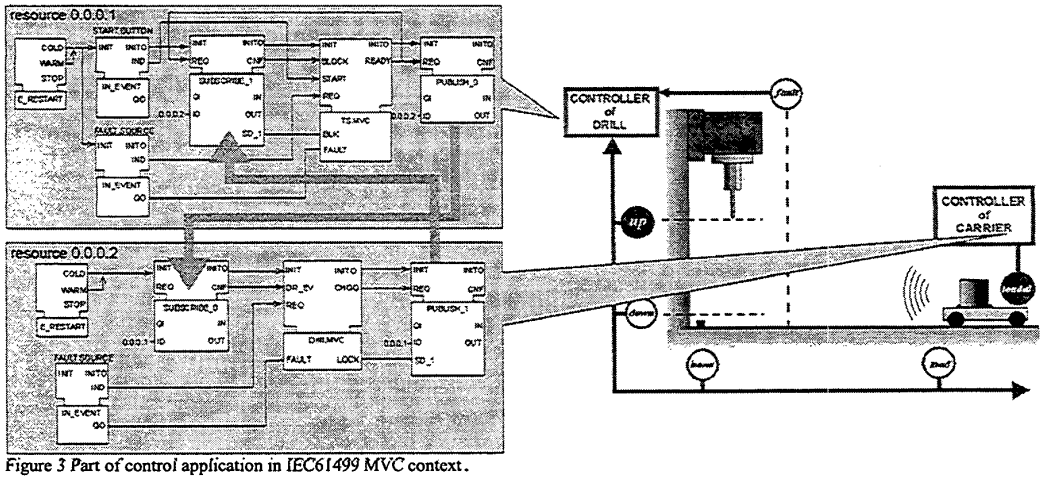

Figure 3 Part of control application in IEC61499 MVC context.

failures and fix them.
The verification trials, conducted so far with VEDA revealed, however, that despite of a lot of provided benefits the verification still involves a big deal of overheads to the normal routines of control engineering. These mostly refer to the modeling of plant and especially to the visualization of the process.

III. INTEGRATION INTO SOFTWARE ENGINEERING FRAMEWORKS

One of the reasons preventing the formal approaches from becoming a part of engineering practice is that modeling of controlled and verified systems is either not a part of the engineering process at all, or that models used in practice for simulation and testing cannot be used for formal procedures of verification or synthesis.
The situation is, however, changing. The necessity of modeling is getting recognized in the industry, and the modeling is better supported by means of IEC61499. The IEC61499 provides abstractions for description of the whole application, rather than merely a programming language for PLC. It gives an opportunity to take into account many implementation issues, such as components distribution, communication, properties of the sensors, actors, etc.
According to the software engineering concept of IEC61499, as described in [9], a control application is developed using Model, View, Control and Diagnostic (MVCD) components. This framework originates in object-oriented design, being adapted for use in the modeling, simulation and testing of industrial process measurement and control systems (IPMCSs) in the IEC 61499 context by modifying some definitions as follows:

1. **Model:** A *function block* that represents the time-dependent logical behavior of the system or device being controlled.
2. **View:** A function block that represents the graphical display associated with one or more **Model** types.
3. **Controller:** A function block that encapsulates the control functions to be performed on one or more instances of associated **Model** types, and presents appropriate *event* and *data interfaces* for integration of its functions with those of other **Controller** blocks.

This basic MVC framework may be also equipped with Human-Machine-Interface (HMI), Diagnostic, and Adapter components. Figure 3 shows an example representing control system of a typical manufacturing component – a processing (drilling) station with a carrier bringing workpieces. The control system is built of two independent components, one of which is responsible for control of the drill, and the other controls the carrier. The controllers are connected to the sensors and actors, as well as to each other by a network. In terms of IEC61499 the controllers are represented as two resources, each of which contains a sub-application. The latter are nets of function blocks, implementing Control, Model, Human-Machine Interface and Communication functions.
Figure 4 shows the internal hierarchical structure of the component related to the control of the carrier. The TS.MVC block implements the closed-loop connection between the controller and the Model/View (MV) part, which, in turn, consists of the MV components of the carrier itself and of the sensors. Finally, the figure shows on the last level of the hierarchy the open-loop structure of the MV of the carrier: it consists of the simulation model of the carrier, which supplies the data to the corresponding View block. Thus, the visualization of the whole process is

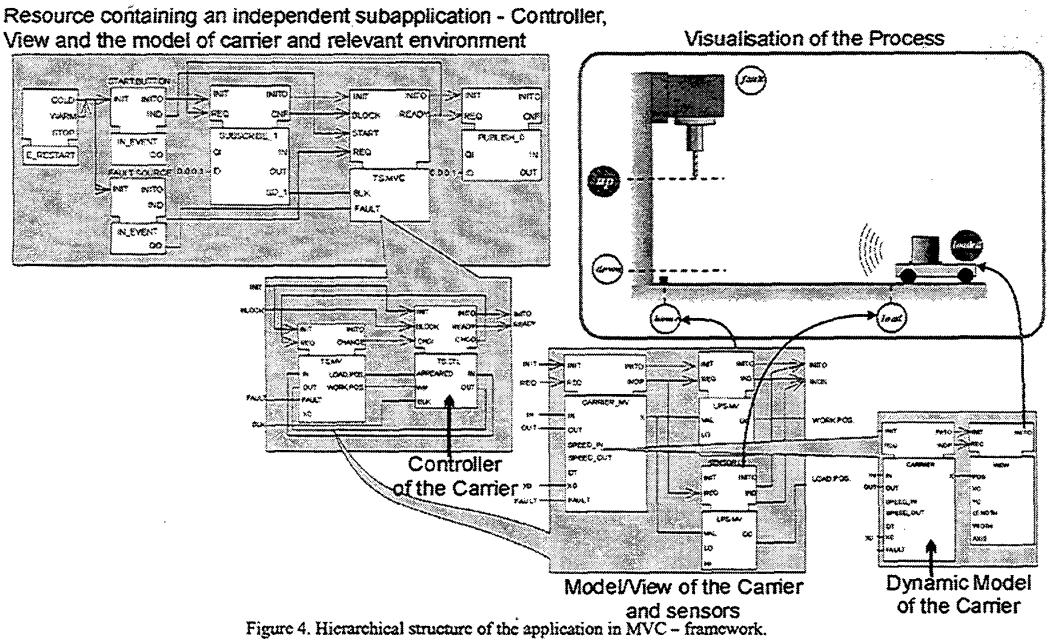

Figure 4. Hierarchical structure of the application in MVC – framework.

built by multitude of the components' View blocks.
The goal of our work is not only to cope with the problems caused by the versatile structure of the control applications in the MVC framework (containing not only pure control logic), but even benefit from it.
The following steps constitute the integration process:

1. Substitute the "Model" components by the equivalent abstractions in SNS. For this purpose a description of the model in a hybrid modeling formalism is required. We assume that it is given as a hybrid automata following the syntax of Execution Control Charts of IEC61499 Function Blocks. The hybrid model is discretized and substituted by the corresponding discrete model in SNS. In future, when the abilities of hybrid model-checkers would allow immediate model-checking of hybrid models of realistic designs, this step could be omitted.
2. Substitute the "View" components by "dummy" SNS primitives (under assumption that the View FBs generate no outputs affecting the logic of execution). The SNS models must have interfaces equivalent to the original function blocks;
3. Convert automatically the "Control" component to the corresponding SNS modules by means of VEDA's translator.
4. Interconnect the obtained SNS modules using the layout of interconnections from the original application (performed by VEDA automatically).
5. Perform the model-checking using the interconnected SNS model. In case that the model-checking reveals some states in the reachability space, where the specification does not hold, generate trajectories to these states and visualize the behavior of the model along them.
6. Visualization in a particular state of the trajectory can be performed given the values of parameters generated during the model-checking and using the existing Model/View components.

IV. TRANSFORMATION OF MODELS

In this section we illustrate the first step of the algorithm given above. First, some essentials on the modeling formalism of Signal/Net Systems are necessary. Net Condition/Event Systems (NCES) [6,9] – a Place/Transition net with modularity support - a finite state formalism which preserves the graphical notation and the

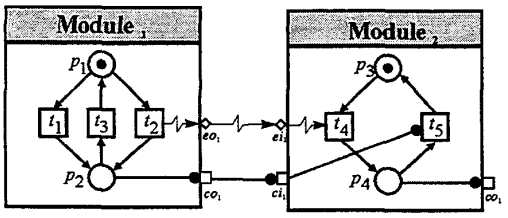

Figure 5. Two NCES modules interconnected by condition and event arcs.

non-interleaving semantics of Petri nets, and extends them with a clear and concise notion of signal inputs and outputs.

Given a Place/Transition net $N = (P,T,F,m_0)$ with set of places P, set of transitions T, flow relation F, and initial marking of places m_0, the Net Condition/Event system is defined as a tuple $N = (N, \theta_N, \Psi_N, Gr)$, where θ_N is an internal structure of signal arcs, Ψ_N is an input/output structure, and $Gr \subseteq T$ is a set of so called "obliged" transitions. The structure Ψ_N consists of condition and event inputs and outputs *ci,ei,eo,co*. The structure θ_N is formed from two types of *signal* arcs. Condition arcs lead from places and condition inputs to transitions and condition outputs, providing additional enableness conditions of the recipient transitions. Event arcs from transitions and event inputs to transitions and event outputs provide one-sided synchronization of the recipient transitions: firing of the source transition forces firing of the recipient, if the latter is enabled by the marking and conditions.

The NCES modules can be interconnected by condition and event arcs, forming thus distributed and hierarchical models. Figure 1 shows an example of NCES which consists of two interconnected modules. NCES having no inputs are called Signal/Net Systems (SNS) [8,12]. The model in Figure 1 is a SNS. The SNS can be analyzed without any additional information about its external environment. Semantics of SNS covers both asynchronous and synchronous behavior that is necessary when modeling of interconnected plants/controller system is concerned.
A state of the NCES module is completely determined by the current marking $m: P \rightarrow N_0$ of places and values of inputs. A state transition is determined by the subset $\tau \subseteq T$ of simultaneously fired transitions, called *step*. The transitions having no incoming event arcs are called *spontaneous*, otherwise *forced*. A step fully determines the values of event outputs of the module. A step is formed by selection of some of the enabled spontaneous transitions, and all the enabled transitions forced by the transitions already included in the step. Thus, both states and transitions of NCES models are distributed, that allows more efficient modeling of distributed systems.
In timed NCES arcs from places to transitions are marked with time intervals, defining low and high limits of the permeability of the corresponding arc. All places bear clocks indicating the age of the marking. Then, a transition is enabled not only when its pre-places are sufficiently marked, but also when values of their clocks are within the time intervals of the incoming arcs of the transition. Clocks of marked places increase their values by discrete units when there are no enabled transitions in the state, but some of the transitions might become enabled after such an increment of clocks. As a direct consequence, the states have attribute "delay", that specifies the time increment of the clocks of this state with respect to the clocks of its predecessors.
The NCES model of the carriage, built in hierarchic modular way is given in Figure 6. The model represents the

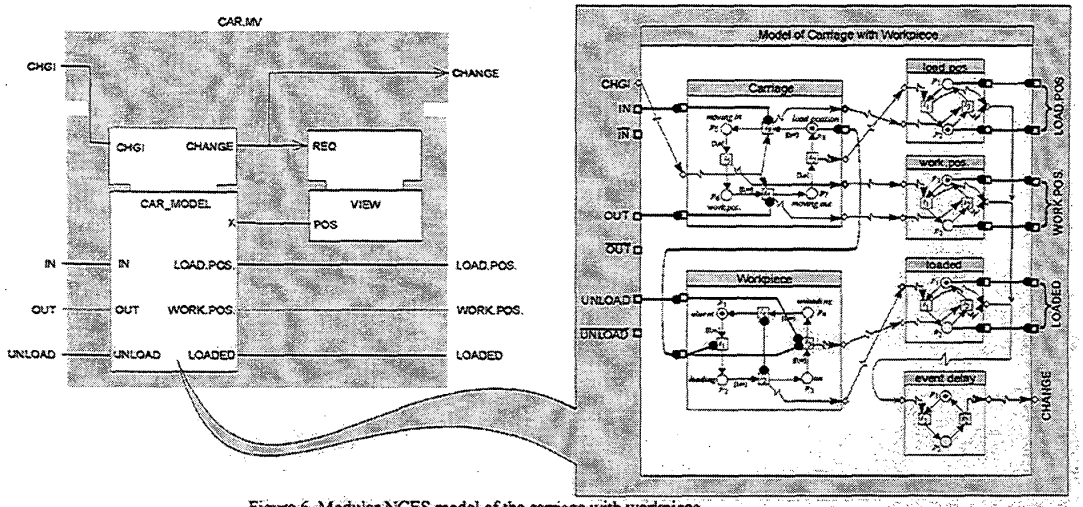

Figure 6. Modular NCES model of the carriage with workpiece.

autonomous carriage and the workpiece including the logic of its placing/removing. It substitutes the function block "CAR.MV" in the original application since it has an identical input/output interface (except for the output X – the numerical value of the coordinate). Interconnections between component blocks of the system are mapped to condition and event arcs connecting modules of the model.

The model consists of modules modeling: a) locations and states of the carriage; b) setting/removing of the workpiece; c) sensors load.pos, work.pos, loaded, and the event generator, which issues an event in every condition change, that is required for communication with the IEC61499 function blocks. The model explicitly defines conditional and time dependencies related to co-existence of several components of the plant's unit. For example, the condition arc from the module "Carriage" to the module "Workpiece" corresponds to the condition "Workpiece can be set/removed to/from the carrier only at the load position of the latter." Typically each elementary unit is modeled as a state-machine, where the states are modeled by the marked places. The power of place/transition nets allows, however, not only to model concurrent interconnected state machines, but also to represent various quantities, such as material flows within the same consistent model. Time properties of the object "Carriage" are expressed in the model by means of time intervals associated with place-transition arcs of the model.

We apply this approach using VEDA and the prototype implementation of the IEC61499 known as Function Block Development Kit (FBDK), which implements function blocks translating them to **Java**. The Model and View function blocks either should be written by user in Java, or composed from library blocks also written in Java. The whole application is translated eventually to the Java code and executed using the Java virtual machine.

When VEDA points out the states in the reachability space, where behavior of the system is of interest to analyze (say, it is erroneous), the usual scenario is to follow the trajectory leading to this state visualizing the process and trends of all relevant data along the trajectory. VEDA uses the data available in the description of states in the computed state space, and re-computes the other missing data directly by means of the original model components. Then the data are passed to the View components to build the visualization screen in the desired state. Thus, the animation can be provided without essential overheads.

V. INTEGRATION OF VERIFICATION TO HOLONIC MANUFACTURING SYSTEMS

One of the applications potentially beneficial for IEC61499 are so called Holonic Manufacturing Systems (HMS) [10]. The HMS concept is based on the idea of *holons*, that are independent self-configuring machines negotiating with other surrounding holons in order to fulfill the production plan. HMS promise to meet the increasing demands for robustness to disturbances, adaptability and flexibility to rapid change on the factory floor of manufacturing enterprises. Besides, the holonic organization helps to cope effectively with failures of production equipment, increasing its workload and output.

On the other hand, it is intuitively clear, that a system built of self-configuring components cannot be completely tested using common testing methods due to the large number of possible combinations of different architectures. Hence, testing of control software for HMS is a challenge for control engineering. The qualitative improvement of validation can be achieved by formal verification of the control logic, along with justification of controller's robustness with respect to possible malfunctions of some system components, such as sensors, actuators, or other equipment units.

Verification that requires high computation power can be used as an external web-service for control engineers developing and testing control systems. A description of the latter in IEC61499 could be sent to the "Verification agent" along with specifications of correct/incorrect

behavior. The server would perform model generation, model-checking and return of results to the client.
Moreover, the verification can be seen as a means of self-testing of holonic manufacturing systems. In a particular configuration of the system (or even before it has been formed), the supervising controller checks (by means of verification), if co-existence of the current components

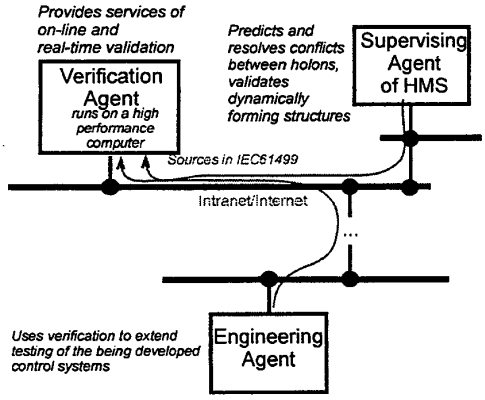

Figure 7. Possible web-integration of the verification agent.

(with their scheduled tasks) may lead the system to a number of known (or inferred) erroneous situations.
The supervising agent might validate them upon appearance before allowing them to function. The verification service can be requested from one of available appropriate verification agents, provided the source codes of control applications of the components (given in IEC61499 format), and descriptions of possible failures or rules, how the latter can be inferred.

CONCLUSION

Application of VEDA allowed to reveal some potentially incorrect internal definitions in the IEC61499. In particular, it was shown that the normative algorithm of input event processing in the basic function block may lead to missing some events when they occur and arrive rapidly.
More progress could be expected after VEDA would be applied to a real control application following IEC61499. This work is currently under way.

ACKNOWLEDGEMENT

This work is supported by the Deutsche Forschungsgemeinschaft under reference Ha 1886/10-1.

REFERENCES:

[1] IEC61499- Function Blocks for Industrial Process Measurement and Control Systems, International Electric Commission, Draft, Tech. Comm. 65, Working group 6, Geneva

[2] J.H.Christensen, *Basic concepts of IEC 61499,* Conference "Verteile Automatisieriung" (Distributed Automation), pages 55--62, Magdeburg, Germany, 2000

[3] E.Clarke, E.A. Emerson, and A.P. Sistla: Automatic verification of finite state concurrent systems using temporal logic., ACM Trans. on Programming Languages and Systems, (8):pp.244--263,4986.

[4] J.S. Ostroff. Temporal Logic for real-time systems, Wiley, London, 1909.

[5] V. Vyatkin, H.-M. Hanisch: Bringing the model-based verification of distributed control systems into the engineering practice, in Proc. of IFAC IMS Workshop, Poznan, 2001.

[6] V.Vyatkin, H.-M.Hanisch. *VEDA - a prototype tool for deep debugging of distributed applications.* Information leaflet, June 2000, <http://at.iw.uni-halle.de/~valeriy/refs/veda.pdf>

[7] V. Vyatkin, H.-M. Hanisch: A Modeling Approach for Verification of IEC1499 Function Blocks using Net Condition/Event Systems, Proc. ETFA'99, pp. 261--270

[8] V. Vyatkin, H.-M. Hanisch: Practice of modeling and verification of distributed controllers using Signal-Net Systems, in Proceedings of the International Workshop on Concurrency, Specification and Programming' 2000, pp.335—349, Humboldt University, Berlin, 2000.

[9] J.H.Christensen, *Design patterns for system engineering with IEC 61499*, Conference "Verteile Automatisieriung" (Distributed Automation), pages 55--62, Magdeburg, Germany, 2000

[10] IMS (Intelligent Manufacturing Systems) project on holonics, <http://hms.ifw.uni-hannover.de /public/overview.html>