IEC61499, distributed control systems,
Discrete-state modelling,
formal verification

Valeriy VYATKIN
Hans-Michael HANISCH

SOFTWARE ENVIRONMENT FOR AUTOMATED VERIFICATION OF DISTRIBUTED INDUSTRIAL CONTROLLERS FOLLOWING IEC61499

Abstract. This paper presents the Verification Environment for Distributed Applications (VEDA), which is a software package for Deep Debugging of distributed controllers. Deep Debugging is a combination of model-based simulation and verification united by a homogeneous graphical user interface. VEDA deals with controllers defined in IEC61131 and IEC61499 and automatically generates the formal model of the controller given the controller's source code.
VEDA includes some facilities to develop models of plants: a graphic editor for Signal-Net Systems models and means to specify visualization of the model. It allows also to simulate the controller/plant closed-loop system, and to prove formally whether the overall behavior of the system satisfies some desired/undesired properties. For better analysis, results of the verification can be visualized by tracing trajectories with timing diagrams of required parameters. Formulation of specifications is facilitated using the visual Signal Diagram Specification Language (SDSL).

INTRODUCTION

The theory of formal modeling and finite-state verification of discrete control systems has been actively developed in the last decade, starting from the works of E.Clarke et al. [Clarke86], and J. Ostroff [Ostroff89]. The general pattern of the verification is as follows: the closed-loop plant/controller system is modeled using a discrete state formalism (for instance, state machines, Petri nets, or their derivatives). As a result the closed-loop model is obtained. The model can be used to check the validity of certain properties for its reachable states. It can be done automatically if the properties are described formally as Boolean conditions, predicates, or temporal logic expressions.

Despite of its numerous advantages, the formal verification remains exotic in the practice of control engineering. One of the reasons of that is the high computational complexity of the formal model-checking. But progress in hardware development and successes in the theory of verification constantly extend the borders of its practical applicability. The existing formalisms do not completely meet the requirements of modeling of the control systems. These are either synchronous (Esterel, Grafcet) that fits to the modeling of controllers, or asynchronous (Petri nets, State machines) that better fits to the modeling of plants. And, last but not the least, is the lack of appropriate user-interface solutions. To bring verification into engineering practice, it has to be better integrated with the currently existing simulation functions.

Martin-Luther University of Halle-Wittenberg
Department of Engineering Science

Current ways of controller design are usually based on a controller's architecture, which is a Programmable Logic Controller (PLC). The execution system of such a PLC executes the control program - written in special languages, which are standardized in the International Standard IEC 61131-3 – [IEC1131] in a purely sequential, cyclic order. First, the current status of the plant, which is indicated by sensors, is stored in an input buffer, then the whole control program is executed, while the values of inputs in the buffer remain unchanged, and in the last step the calculated outputs are transmitted to the actuators of the plant.
Although the problems of verifying correctness of a control program in such a classical architecture is hard enough, practical needs for more flexibility, distribution, and fast and easy reconfiguration bring up new controller architectures.
The uniform vendor-independent solution for programming of distributed control systems provides the currently being developed IEC standard 61499 [IEC61499, Christensen2000]. 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 which can be tested without knowledge about particular implementation details. In IEC 61499 the basic programming structure is function blocks with event and data inputs and outputs. The block is conditionally divided onto the "head" responsible for the execution logic, and "body" which contains the 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. It is obvious that new challenges for testing or verification result from this design approach.

In this paper, which is a sequel to the paper [Vyatkin2000] presented on the previous Workshop, we present an overview of the verification process supported by a number of methods and software tools, destined to make it really integrated with the routine work of control engineers. The presented approach bases on the modelling of control systems with Signal Net Systems (SNS) [Starke2000]. An integrated software environment VEDA (Verification Environment for Distributed Applications) has been developed to support all steps of controller design and testing. The paper is structured according to these steps.

STEP 1. CONTROLLER DESIGN

The process of verification applies to the controllers following the IEC61499. For example, consider a simple plant "BORING STATION" in Figure 1. It consists of a boring machine (drill) and a transfer stage, which delivers workpieces to the working position of the drill. The loading/unloading of the drill is performed in the *loading* position (detected by sensor *load.pos.*), opposite to the working position. The drill starts drilling automatically, when the workpiece comes to the working position.

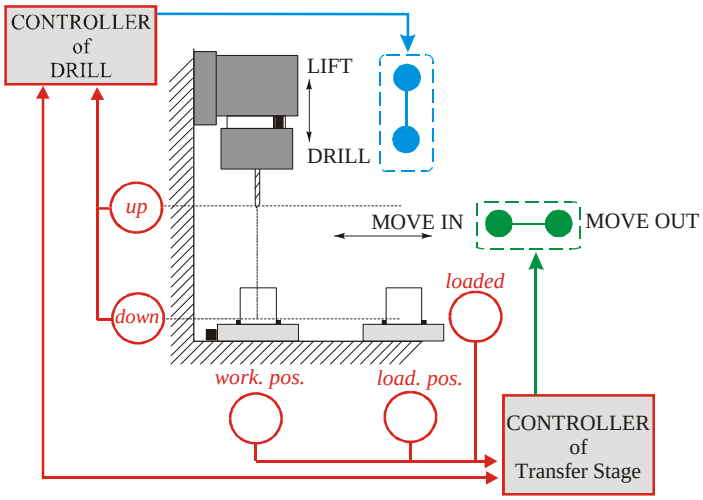


Figure 1. Structure of the distributed control system.

When drilling is over, the workpiece is moved away. Presence of the workpiece on the tray is reported by means of the sensor *loaded*. Control of the plant is performed by means of two independent controllers (one for the drill, and the other for the transfer stage). Sensors and actuators are connected to their respective controllers by segments of networks. The controllers also communicate via network. Access to the data is limited for each controller by the data available in its own network segments, plus the data explicitly provided by the other controller. A block diagram (following IEC61499) of the distributed control system of the plant is presented in Figure 2. Function block TS_CTL implements the control logic of the transfer stage, block DR_CTL controls the drill, the block MOD represents the plant or its model, and the block INIT models initialization of the controllers (for example in case of a start-up of the control device). According to IEC61499, a function block consists of head (the upper part) and body (the lower part). The head is connected to the event inputs and outputs and is responsible for the execution logic. The body is connected to the data input/outputs and contains the data processing algorithms, which are called by the execution control.

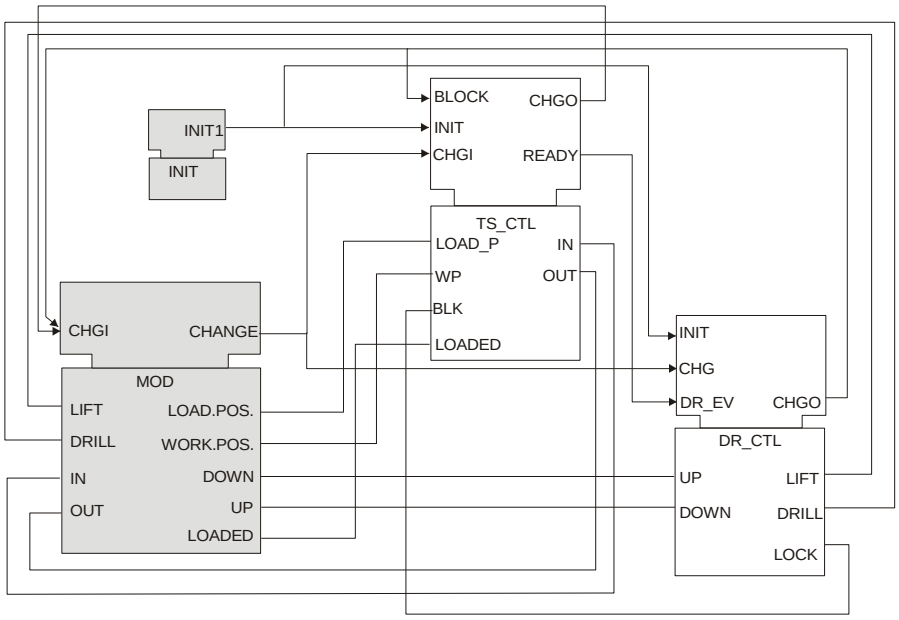


Figure 2. Block diagram of the control system following IEC61499.

The internal control logic of the blocks TS_CTL and DR_CTL is implemented by means of Execution Control Charts (ECC) presented in Figure 3. The ECC represents state-machines, and its syntax is a simplified version of the Sequential Function Chart (SFC), which is a

programming language of IEC61131-3. For simplicity we extended the syntax of the ECC, allowing simple algorithms, such as an assignment of a variable to be presented directly at the places, reserved for the algorithm calls.

The control logic of drill and transfer stage is quite simple: once loaded with a workpiece the transfer stage delivers it to the working position, and then reports to the controller of the drill about that. The latter blocks moving of the transfer stage during the drilling, by issuing the corresponding signal LOCK.

As a result of the design step, the source code of the application (network of interconnected function blocks) is obtained. In general it includes the component algorithms, and is presented in Extended Structured Text format (the Structured Text (ST) is also a programming language of IEC61131). Now the application needs to be validated, i.e. correctness of its outputs has to be proved for all input combinations, which are possible in the real life. To generate the inputs, a finite state model of the plant is used. It is connected to the controllers in the closed loop, as it can be seen from Figure 2.

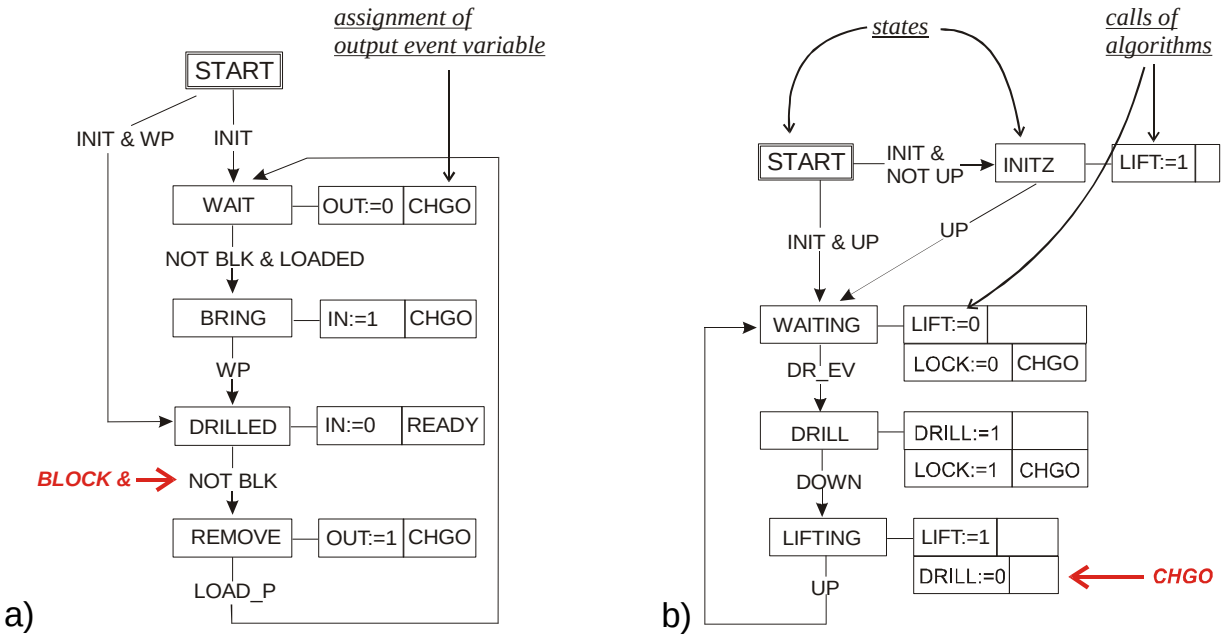


Figure. 3 Execution Control Charts of the function blocks TS_CTL (a), and DR_CTL (b).

In fact, the whole interconnected system has to be substituted by its finite-state model. We use for modeling the formalism of Signal-Net systems [ASEN, Starke2000]. The model has the same modular structure as the original application. The models of the control function blocks are generated automatically by VEDA, given their source code. The model of plant has to be prepared manually. This will be briefly explained in the following section.

STEP 2. MODELING PLANT

The model-based testing of control algorithms is widely used in practice and suggested as a basic component of system engineering in IEC1499, which supports the MVC framework (Model, View, Controller). To add the verification to this framework, a formal model of plant is required.

It is also useful to combine the formal model with the models used for simulation and visualization, and further incorporate them into an IEC61499 function block to provide a homogeneous representation. The model of plant can be also built in the modular way. The modeling starts with the design of a finite state model of each subsystem, further connecting these components by event and condition signals. The SNS formalism allows designing models according to this top-down approach. The model of the drilling station is presented in Figure 4. The model is a module of SNS, composed as a network of modules, where modules "Drill" and "Transfer Stage" model the units of the same name, and the other five modules model the logic sensors.

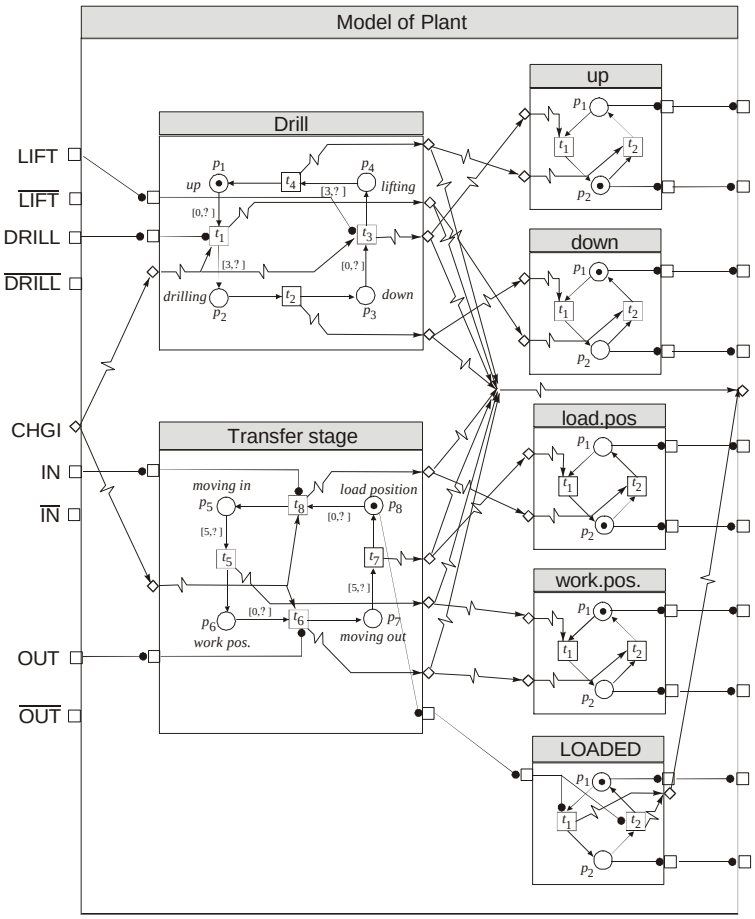


Figure 4. SNS model of plant and sensors.

Outputs of the logic sensors are represented by two event outputs in SNS: one for value "false", and the other for the value "true". Hence, a connection line from MOD to either TS_CTL, or DR_CTL in the block diagram in Figure 2 corresponds to two condition arcs in the SNS model. The same applies to the connections from the controller to the model of plant: the negative value of each signal is always available.

The model is built using timed SNS, where each arc from a place to transition is marked by the permeability time interval [*lo*, *hi*]. The default value assigned implicitly to all arcs is [0,?]. In terms of the model, the interval sets the time limits of the marking age in the source place when the corresponding arc may affect firing of its target transition. Its usage can be illustrated as follows. If the *moving in* takes minimum 5 time units, then we mark the corresponding arc from the place p_5 to the transition t_5 in the module "Transfer Stage" by interval [5,?]. In this paper we are not able (due to limited space) to present all the details of the model building using SNS. Obviously, features of the SNS do not limit the developers only by state-machine modeling. The general framework of the model-development, as it is supported by VEDA, is given in Figure 5. Based on the technological documentation of the plant, the SNS model of each unit's behavior is developed. This process is assisted by the interactive editor of SNS. Along with that, each state of the unit's behavior can be associated with the corresponding visualization display, and even with animation. For this purpose a Simple Modeling Language (SML) is developed. Statements of SML describe every object of the visualization in connection with the SNS model.

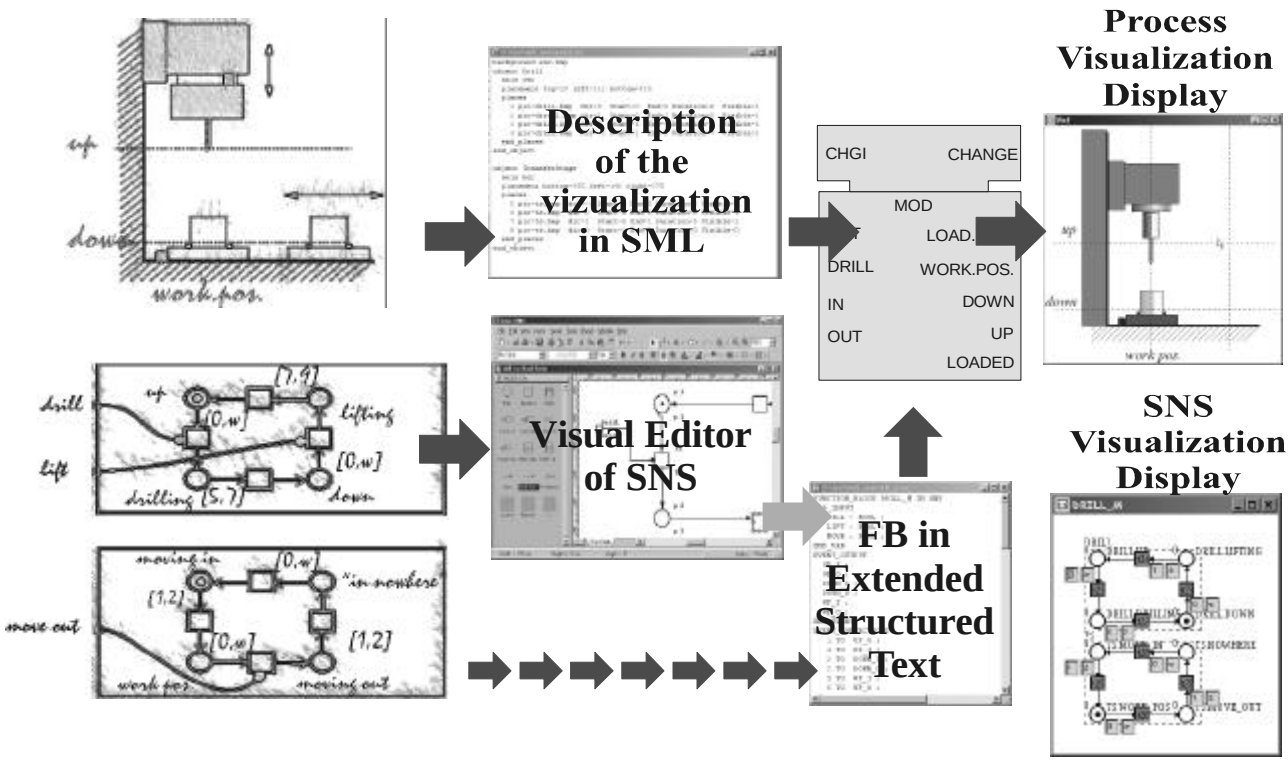


Figure 5. Modeling of plant supported by means of VEDA.

The visual editor of SNS provides editing of SNS models and their transformation in Function Block format, which is correspondingly extended to be able to store the information about SNS on one hand, and to be compatible with the usual IEC61499 FBs on the other. As a result of this step we obtain the ASCII file of the Function Block, containing the model of plant, and the corresponding SML file, containing the information about visualization.

STEP 3: SELECT AN APPROPRIATE DEBUGGING PATTERN

VEDA offers a number of verification opportunities. Sometimes the easiest way to test the controller is to enter the control program to a control device (say a PLC) directly connected to the plant, and then run it. For this purpose the model of the controller can be run in the step-by-step mode in connection with the real plant, provided by a softPLC software being run together with VEDA. Since no implementation of IEC61499 is available so far, the combination of VEDA and softPLC can be regarded as one. The other options are model-based.

If the full model of plant is developed as previously explained, the block with the model has to be linked to the block of the controller forming thus the model of the closed-loop system. The model can be run step-by-step (simulation) or can be exposed to formal verification. The properties to be verified, or conditions of correct behavior are to be expressed as predicates, or temporal logic formulae.

If the model of the plant is not available, a partial model can be generated from specifications given in the Signal Diagram Specification Language (SDSL).

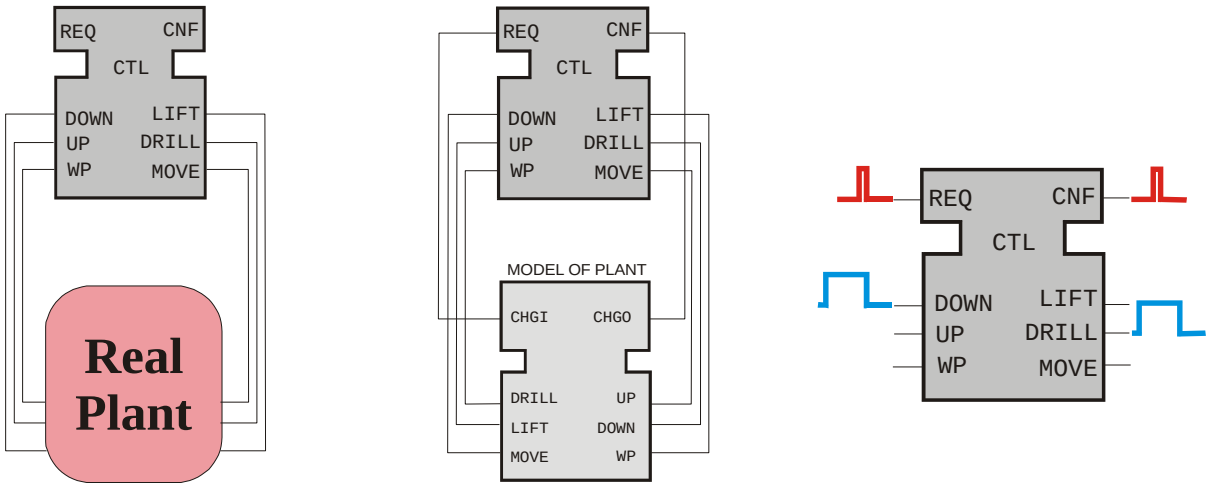


Figure 6. a.) Testing in connection with real plant; b) Model-based testing in the closed-loop; c) Open-loop testing;

It is possible to define values of some event and data inputs (which are of interest) and their relationship to each other, leaving the others non-specified explicitly (i.e. assigned by a default constant). The outputs can be specified in a similar manner. The specification of inputs is translated then to an equivalent SNS model, which can be connected to the model of the controller. The result is the open-loop model, which can be used in a way similar to that of the closed-loop modeling. The specifications of outputs can be automatically translated to temporal logic expressions, i.e. to statements about output behavior of the controller under specified inputs. The statements can be further proved using a model checker.

STEP 4. DEEP DEBUGGING

Deep Debugging is a combination of model-based simulation and verification united by a homogeneous graphical user interface. This process assisted by VEDA is illustrated in Figure 7. VEDA integrates several functions of simulation and verification. Starting from a source code of the closed-loop system, which consists of the controller and a model of the plant, VEDA generates automatically the SNS model of the system.

Several options then can be chosen: The model can be simulated in step-by-step way with visualization of the model's state during this process. Simulation, however, does not allow checking of all possible scenarios of system's behavior. However if the required property can be formulated as a Boolean statement or expression in CTL (Computation Tree Logic) then it can be checked using either the internal model-checker of VEDA, or SESA – a more powerful model-checker and analyzer. As a result, the property is either proved, or it fails in a counterexample state. Then, it is important to explain the reason of the failure. For this purpose the trajectories from the initial state to the counterexample state have to be studied. They can be visualized using timing-state diagrams of selected parameters (such as inputs, outputs, and internal variables of the controller). Each state of the diagram can be further visualized by SCADA-style process display, and model state display. As a result failures in the controller can be analyzed and corrected efficiently.

After several iterations of model-based simulation and verification the controller can be tested in closed-loop with the real plant using the soft-PLC component of VEDA (not implemented yet).

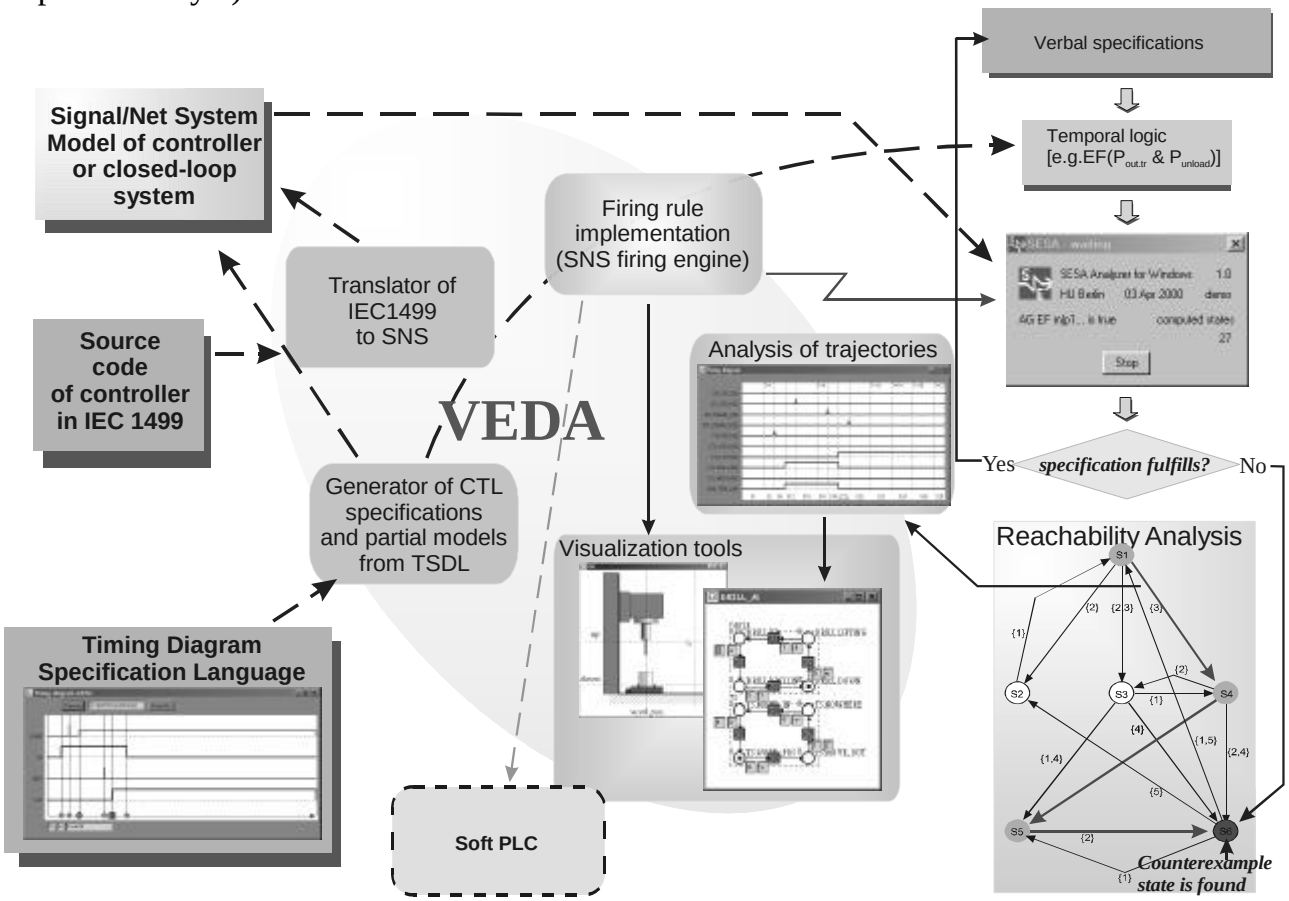


Figure 7. Process of the Deep Debugging assisted by VEDA.

To facilitate formulation of specifications the Timing Diagram Specification Language can be used. To check how the controller reacts on a particular input sequence, it is possible to specify the sequence in SDSL (even partially, for some inputs it is possible to set the "undefined" value), and automatically get the SNS model with equivalent output behavior. The model can be connected to the model of the controller, and all together examined by the model-checker. As a result we can get the timing diagrams of outputs. Moreover, the desired outputs can be specified in SDSL, then automatically converted to equivalent CTL expressions, which can be verified/falsified using the model-checker as it was explained above.

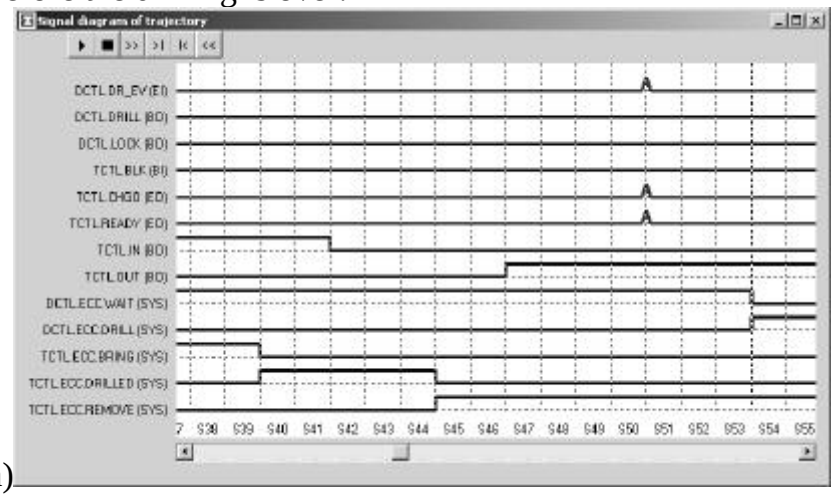
EXAMPLE OF APPLICATION

We expose the drilling station controller to VEDA, checking it on presence of dangerous states in its reachability graph. One of the dangerous conditions is an attempt to move the transfer station during the drilling. The corresponding predicate is: “DCTL.LIFT and TCTL.OUT”. VEDA checked the reachability space of the closed–loop model (4861 states) on the validity of the predicate and pointed out the dangerous states. Also VEDA reported

presence of a dead state in the reachability graph, which signals about a deadlock in the system behavior.

To analyze reasons of the incorrect behavior, it is possible to visualize the trajectories leading to this state with signal diagrams of the variables, and provide the view of the animated process visualization display along it. The reason becomes clear at the look at the signal diagram in Figure 8,a and corresponding visualization in Figure 8,b.

Once the transfer stage arrives to the working position, its controller has to come to the state DRILLED and send a corresponding message READY to the controller of drill. The latter issues blocking condition BLK, which does not allow the transfer stage to move away before the drilling is over.

a) 

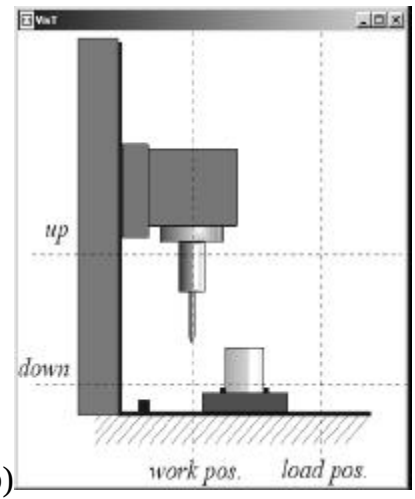
b) 

Figure 8. Timing diagram of signals along a trajectory leading to a dangerous state and its animated visualization display.

In fact, the state DRILLED in the controller of the transfer stage is unstable: the condition on the transition to the next state REMOVE is immediately TRUE, due to the logic of ECC execution: the outgoing transition conditions are evaluated even before the output events, related to a particular state are issued. This implies that the state of TS_CTL is changed to the REMOVE even before the signal READY to DR_CTL has been issued. To fix this error the transition condition NOT BLK has to be fortified as: BLOCK & NOT BLK, where BLOCK is an event issued by DR_CTL after setting the BLK to 1.

After the modification, the deadlock state disappears and no more states satisfying the danger condition is present. Besides the number of reachable states is reduced to 330 that clearly simplifies further verification of the controller. The repetitive application of such a procedure with various specifications helps to improve the controller's correctness.

CONCLUSION

The paper presents main features of the verification software VEDA and shows a basic idea of its application. The further work will include theoretical and practical steps aimed at the extension of the functions of VEDA, as well as attempts of its application in real control engineering.

ACKNOWLEDGEMENT

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

REFERENCES

[ASEN] P.Starke, S.Roch, K.Schmidt, H.-M. Hanisch, A.Luder: Analysing signal-event systems, Technical report**,** *Humboldt Universitat zu Berlin*, Institut fur Informatik, July, 1999

[Clarke86] E.Clarke, E.A. Emerson, and A.P. Sista: Automatic verification of finite state concurrent systems using temporal logic, ACM Trans. on Programming Languages and Systems, (8):244-263, 1986.

[Christensen2000] J.H.Christensen: Basic concepts of IEC 61499. in Proc. of Conference "Verteile Automatisieriung" (Distributed Automation), pages 55--62, Magdeburg, Germany, 2000

[Halbwachs93] N. Halbwachs: Synchronous Programming of Reactive Systems, Kluwer, 1993

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

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

[Ostroff89] J.S. Ostroff: Temporal Logic for real-time systems, Wiley, London, 1989

[Sreenivas91] R.S.Sreenivas, B.H.Krogh: On condition/event systems with discrete state realisations *Discrete Event Dynamic Systems: Theory and Applications, 2(1):209-236,1991*

[Starke2000] P. Starke: Symmetries of signal-net systems. Workshop on Concurrency, Specification and Programming, pages 285--297, October 2000.

[Vyatkin2000] V.Vyatkin, H.-M. Hanisch: Modeling of IEC61499 function blocks as a clue to their verification. In Proc. of XI Workshop on Supervising and Diagnostics of Machining, Karpacz, Poland, March 2000.

[Vyatkin2000-1] V. Vyatkin, H.-M. Hanisch: Practice of modeling and verification of distributed controllers using Signal-Net Systems, in Proceedings of the Workshop on Concurrency, Specification and Programming' 2000, pages 335—349, Humboldt University, Berlin, 2000.