Journal of Intelligent Manufacturing, 14, 123–136, 2003

Verification of distributed control systems in intelligent manufacturing

VALERIY VYATKIN and HANS-MICHAEL HANISCH

Department 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

Received ???? and accepted ????

This paper presents an application of formal methods for validation of flexible manufacturing systems controlled by distributed controllers. A software tool verification environment for distributed applications (VEDA) is developed for modeling and verification of distributed control systems. The tool provides an integrated environment for formal, model-based verification of the execution control of function blocks following the new international standard IEC61499. The modeling is performed in a closed-loop way using manually developed models of plants and automatically generated models of controllers.

Keywords: Verification, distributed computer control systems, modeling

1. Introduction

The ultimate goals of the ongoing research and development works, united by the intelligent manufacturing systems (IMS) initiative lie in a reduction of the production cycle along with extended number of available options in the product, and on-demand production and delivery. One of the solutions to fulfill these goals is a concept of holonic manufacturing systems (HMS) (IMS, $$$). The HMS concept is based on the idea of *holons* that is independent self-configuring machines, capable of 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, holonic organization helps to cope effectively with failures of production equipment, along with increasing its workload and output.

The development of HMS implies numerous impacts on every part of manufacturing technologies, including control and software engineering. The state-of-the-art in hardware of control systems is characterized by wide application of distributed architectures. If compared with centralized control systems, the distributed ones consist of components connected to the environment and to each other by means of networks, such as PROFIBUS, DeviceNet, CAN, etc. This architecture provides easy physical integration of programmable controllers, intelligent field devices, and engineering tools into the control system. Thus, the hardware part of the distributed control system (DCS) can be assembled almost following the ‘‘Plug-and-Play’’ principle that fits well to the requirements of HMS. The more evident becomes then the importance of the corresponding software integration. The methods and tools currently used for software design of industrial controllers do not provide the desired level of software portability, inter-operability and configurability.

The draft of the international standard IEC61499 (Function Blocks for Industrial Process Measurement and Control Systems, 1998; Lewis, 2001) is an attempt to bridge this gap and to provide a comprehensive software engineering concept for DCSs, logically consistent with the currently existing practice and standards of software development. The

IEC61499 provides the component architecture for development and modeling of distributed measurement and control applications built from the units complying with IEC61131-3 (1993)—programming concept for centralized programmable logic controllers (PLCs), and IEC 61804 for DCS.

However, along with numerous benefits that the HMS concept and IEC61499 promise to bring, there are serious doubts related to the testing of control software. With the growing complexity of the industrial automation systems, software becomes the most complicated and error prone part. 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. The current practice of testing is not sufficient even for complex centralized control systems. Validation, which relies on a finite number of tests, guarantees that only the developed software works correctly for this finite number of input tests. In reality, the software may encounter input combinations not covered by the tests, thus generating unpredictable outputs. This may lead to erroneous behavior of the production equipment with the corresponding dismal consequences.

A 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. The following distinctive features of the IEC61499 seem to be potentially beneficial for the achievement of this goal.

- The IEC61499 provides a formal semantic model of distributed control systems. The basic programming structure of the new standard—a function block, is more autonomous than in IEC61131-3. It may contain a number of algorithms united by the event-driven execution control Chart (ECC)—a state machine that determines the basic execution logic of the block. Thus, the IEC61499 model provides a clear separation of concerns: event flow is separated from data flow, and execution control is separated from data processing in application algorithms.
- An application in IEC61499 is built as a net of function blocks (possibly hierarchial) interconnected via event and data signals, and distributed over resources and devices. Cooperation of the latter with the environment can be also specified using so called service and communication interfaces. Thus, a distributed application can be developed on the abstract logic level and then applied to a particular architecture of hardware.
- The MVCDA component architecture using IEC61499 as suggested in Christensen (2000) relies upon integration of such components as control, modeling, view, diagnostics, and adapters within the same project. It allows to define the formal model of the whole control system reflecting properties of controllers, other software and hardware components, service and communication interfaces, as well as properties of the controlled plant.

The above mentioned features provide the means for analysis of the execution semantics of control applications on various abstraction levels. Clearly, more the details known better is the behavior predicted. The first level of abstraction is only achieved by the given structure of the system as an interconnection of the component blocks. A more detailed semantics is described when the execution control of the blocks and/or the algorithms are available. The next level can be achieved when the structure of the container resources/devices is given along with mapping of component function blocks over the containers. A more precise semantics corresponds to the level when communication interfaces are defined, and so forth.

In this paper we present an attempt to develop methods and tools of formal verification applicable to IEC61499, and correspondingly to HMS. Our approach is destined to support verification as a natural part of control engineering. A prototype software tool VEDA has been developed to show the feasibility of the approach.

The paper is structured as follows. Section 2 gives a brief overview of classical methodologies of model-based formal verification. The software design concept of IEC61499 is illustrated by means of a simple example in Section 3. Section 4 is devoted to the issues of formal modeling of plant-controller systems, which provide the basis of formal verification. We outline some features of the formalism of Net Condition/Event systems, though in a rather informal way. More rigorous definitions can be found in Starke *et al*. (2000). Finally, the verification tools are applied in Section 5 to the example introduced in Section 3. The paper concluded with a discussion of verification perspectives in intelligent industrial automation systems.

2. Classic framework of verification

Methods of formal model-based verification of control systems have been developed during the last decade. Representative descriptions can be found in Clarke *et al*. (1986), Ostroff (1989), Kowalewski *et al*. (2000). An outline of those is as follows:

(1) Verification is applied to a piece of software code, describing the controller of a certain plant. Plant and controller form the interconnected closed-loop control system. The controller code is given in one of general-purpose or specialized programming languages, for example, following IEC61131.

(2) The closed-loop system is modeled using an appropriate finite-state or hybrid formalism, for example, state machines, Petri nets, etc. The model of the plant has to be designed manually by control engineers, while the model of the controller can be built automatically given the code.

(3) The experience of the control engineers as well as technical documentation usually provide a lot of specifications of desired or forbidden behavior of the plant, i.e. the properties to hold or to avoid. The specifications have to be formalized using a formal language compatible with the description of the model.

(4) Given the model and a number of formal specifications, it can be formally checked whether the specifications hold with respect to the model. This process is called model-checking.

(5) The results of the model-checking have to be interpreted in terms understandable by the engineers. For this purpose, a bi-directional mapping from the original system to its model and back has to be provided.

This way of verification, however, is still far away from being everyday practice due to a number of reasons such as follows:

(1) Despite the number of theoretical and practical approaches towards practical application of verification, there are very few software packages available for smooth integration of verification into the control engineering practice. In particular, the ways of formal modeling of the controlled plants are different from the modeling used for simulation and testing. The variety of formalisms developed for formal modeling makes standardization of this process difficult.

(2) The verification trials conducted in academia deal with very simplified examples of control systems. Comprehensive verification of control applications, taking into account not only control logic, but also system issues, is computationally complex and therefore unfeasible.

(3) Last, but not the least, is that the control design companies do not want to cast doubt on their practice of software development, insisting on that the settled routine (based on certain norms, rules, and software engineering concepts) ensures the quality of the final product.

3. An example of a distributed control application following IEC61499

Let us illustrate the impact of the distributed design with the help of a simple plant ''BORING STATION'' as presented in Fig. 1. It consists of a boring machine (drill) and a carriage, which delivers workpieces to the home position of the drill. The loading/unloading of the carriage is performed in the position detected by sensor *load.pos.*, opposite to the home position. The drill has to start drilling when the workpiece comes to the home position. When drilling is finished, the

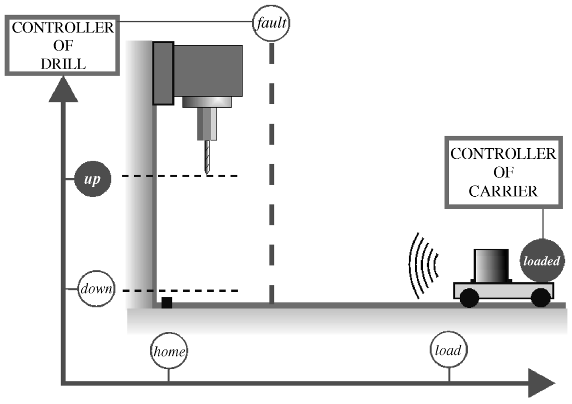

Fig. 1. Structure of the distributed control system.

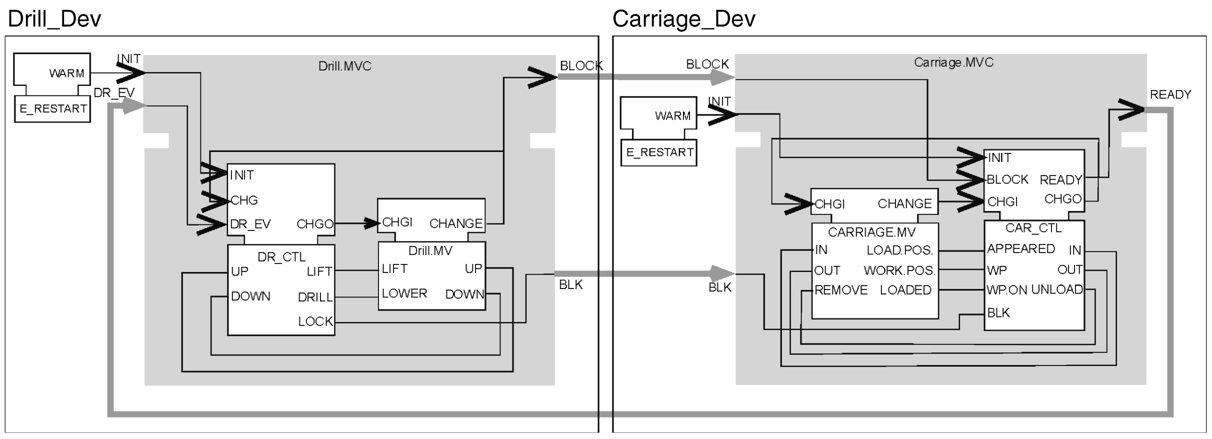

Fig. 2. Block diagram of the control system following IEC61499.

workpiece is moved away. The presence of the workpiece on the tray is reported by the sensor *loaded*.

The plant is controlled by two independent controllers, one for the drill, and the other for the carriage. Sensors and actuators are connected to their respective controllers by segments of networks. The controllers also communicate to each other via a 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.

The block diagram in Fig. 2 shows the (simplified) structure of the DCS of the plant following the IEC61499. The application consists of two interconnected subapplications ''Drill.MVC'' and ''Carriage.MVC'' (the MVC abbreviation stands for model-view-controller). In this particular case, the subapplications are mapped onto two devices: ''Drill_Dev'' and ''Carriage_Dev''. According to IEC61499, a basic function block consists of a head (the upper part) and a 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 invoked by the execution control.

Function block CAR_CTL implements the control logic of the carriage, block DR_CTL controls the drill. The blocks ''Drill.MV'' and ''Carriage.MV'' in our diagram represent models of their respective objects. The block ''E_RESTART'' models a service function of the resource which issues the corresponding event on the warm start-up to initialize the software. Further elaboration of the block ''Carriage.MV'' is given in Fig. 3. The ''VIEW'' component provides visualization of the object, given its coordinate, generated by the model in block ''Car_Model''. The model can be substituted by a process interface function block implementing interfaces to the corresponding objects. This way of software design provides a consistent incorporation of modeling into the development of control applications.

The internal control logic of the blocks CAR_CTL and DR_CTL is implemented by means of execution control charts (ECC). The ECCs are state-machines whose syntax is a simplified version of the sequential function chart (SFC) of IEC61131-3. Fig. 4 shows direct information connections between the two controllers. For simplicity we allow in our examples some deviations from the textual and graphical syntax as defined in IEC61499. Thus, simple algorithms,

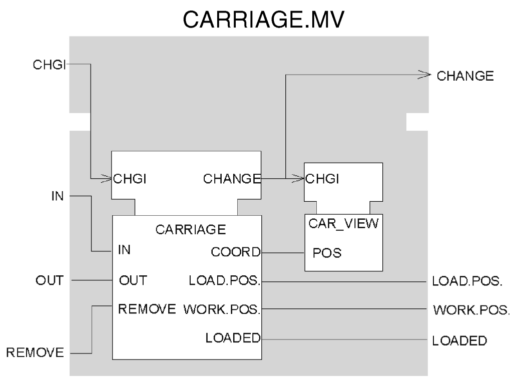

Fig. 3. Internal structure of the block CARRIAGE_MV.

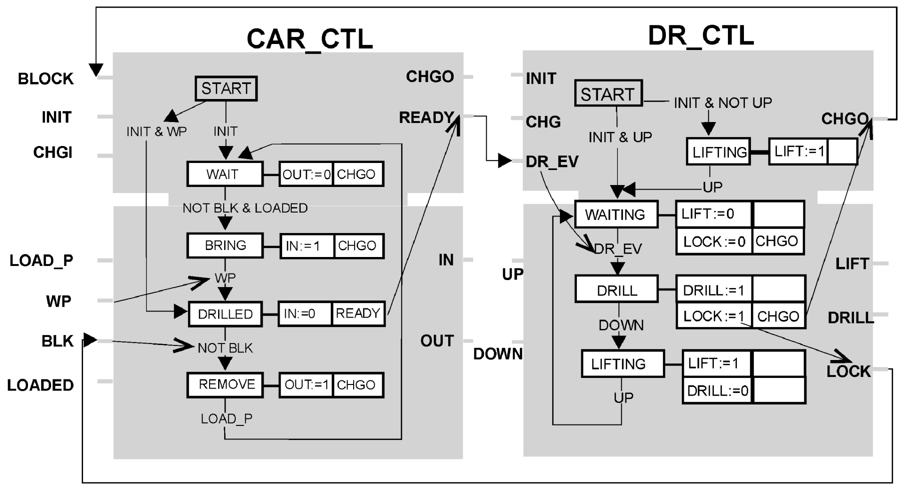

Fig. 4. Execution control charts of function blocks CAR_CTL and DR_CTL.

such as an assignment of a variable, are presented directly at the places, reserved for the algorithm calls.

The applications following IEC61499 can be presented either graphically, as it is partly shown in Figs. 3 and 4, or textually, in extended structured text format. The source code contains the description of the system's structure, including internals of component function blocks (ECCs, algorithms), their interconnection to each other, as well as the mapping of the function blocks to resources and devices. Thus, the IEC61499 provides a comprehensive model for description of distributed control applications, which can also be used for the validation purposes.

4. Modeling formalism of signal-net systems

The interconnected system has to be substituted by its finite-state model to perform formal model-checking. We use a modular way of modeling by means of so called net condition/event systems (NCES) or signal/net systems (SNS) which exist in several dialects and have been applied for modelling, verification and synthesis of controllers and control systems of various types (Hanisch and Lüder, 1999; Hanisch *et al.*, 1997; Vyatkin and Hanisch, 1999; Hanisch *et al.*, 2000).

From today's point of view, the general idea which is common to each particular ''dialect'' is quite simple, namely, the way of thinking and modeling a system as a set of modules with a particular dynamic behavior and their interconnection via signals. This way of modeling is a very intuitive one, and the modules can be pre-tailored and used over and over again. Each module is equipped with inputs and outputs which are of two types:

(1) Condition inputs/outputs carrying state information, and

(2) Event inputs/outputs carrying state transition information.

This way of extension of the system with inputs and outputs clearly reflects the duality of Petri nets, namely the clear distinction between states and states transitions with their own graphical representation, semantics, and formal properties. An illustrative example of the graphical notation of a module is provided in Fig. 5.

Condition input signals as well as event input signals are connected with transitions inside the module. Whether a transition of a module fires does

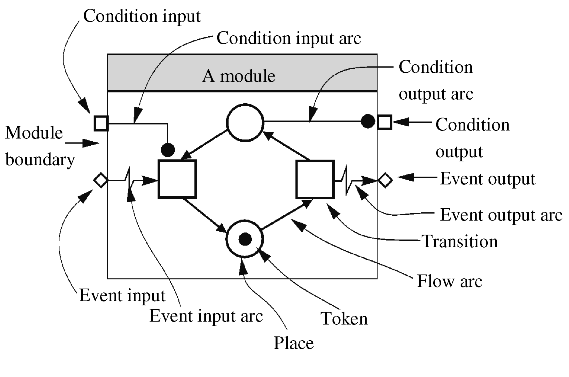

Fig. 5. Graphical notation of a module.

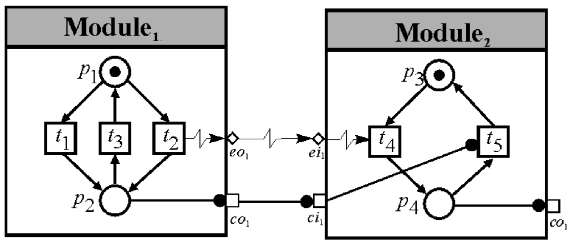

Fig. 6. Example of modular composition.

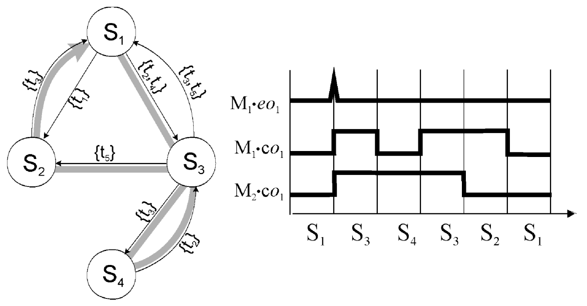

Fig. 7. Reachability graph of the interconnected SNS model with outlined path and sequential state diagram of input/output signals of the modules.

not only depend on the current marking (as it is the case in classical Petri nets) but also on the incoming condition and event signals. Incoming condition signals enable/disable a transition by their values in addition to the current marking. Incoming event signals force transitions to fire if they are enabled by marking and by condition signals. Hence, we get a modeling concept that can represent enabling/disabling of transitions by signals as well as enforcing transitions by signals. More than this, the concept provides a basis for a compositional approach to build larger models from smaller components. ''Composition'' is performed by ''glueing'' inputs of one module with outputs of another module as depicted in Fig. 6.

Result of the composition of two NCES N_1 and N_2 is an NCES N_c obtained as a union of the components and which can be represented as a new module. Inputs and outputs of the ''composition'' are unions of the components' inputs and outputs, except for those which are interconnected to each other, and hereby ''glued'', i.e. substituted by the corresponding condition and event arcs.

NCES having no inputs are called signal/net systems (SNS) (Hanisch and Lüder 1999; Starke *et al.*, 2000). The model in Fig. 6 is a SNS. The SNS can be analyzed without any additional information about their external environment. The semantics of SNS is defined by the firing rules of transitions. There are several conditions to be fulfilled to enable a transition to fire. First, as it is in ordinary Petri nets, an enabled transition has to have a token concession. That means that all pre-places have to be marked with at least one token.[1]

In addition to the flow arcs from places, a transition in SNS may have incoming condition arcs from places and event arcs from other transitions. A transition is enabled by condition signals if all source places of the condition signals are marked by at least one token. The other type of influence on the firing can be described by event signals which come to the transition from some other transitions in the net. Transitions having no incoming event arcs are called spontaneous, otherwise forced. By default presence of at least one non-zero event signal is required to enable transitions by event signals. A forced transition is enabled if it has token concession and it is enabled by condition and event signals.

SNS are executed in steps, which are the sets of simultaneously firing net transitions. An executable step is formed by first picking up a nonempty subset of enabled spontaneous transitions, and then adding as many as possible of enabled transitions which are forced to fire by event signals produced by other transitions included in the step. Such a step is called maximal with respect to its forced transitions.

A state of SNS is defined by marking of places. The tuple $M = 0 >$ denotes the reachability structure of a SNS, where Z is a finite set of reachable states, R a finite set of state transitions, and s_0 an initial state. A state trajectory is a sequence of states $(s_i) = s_0, s_1, \ldots, s_i, \ldots,$ such that $\forall s_j, s_{j+1} \in Z \exists \tau \in R : s_j[\tau > s_{j+1}$. Figure 7 presents the reachability graph for the SNS from Fig. 6. Nodes of the graph correspond to the states, arcs correspond to the state transitions. The arcs are marked with their respective steps of net transitions. One of the trajectories is unfolded in the figure to the linear sequence of states, to illustrate changes of the the model's parameters by means of a sequential diagram.

There are special means provided for description of both asynchronous and synchronous behavior in the same net. This is necessary when modeling of interconnected plant/controller systems is concerned. This is achieved either by introduction of obliged transitions firing whenever they are enabled, or by timing.

The concept of discrete timing is applied to the SNS as follows: to every pre-arc $[p, t]$ of the transition t, we attach an interval $[l, h]$ of natural numbers with $0 < l < h < \infty$. The interval is also referred to as permeability interval. If a pre-arc has no explicitly designated permeability interval, it is assumed to be $[0, \infty]$. The interpretation is as follows. Every place p bears a clock $u(p)$ which is running iff the place is marked $(m(p) > 0)$ and switched off otherwise. All running clocks run at the same speed measuring the time the token status of its place has not been changed. If a firing transition t removes a token from the place p or adds a token to p, the clock of p is turned back to 0. A (marking-enabled) transition t is time-enabled only if for any pre-place p of t the clock at place p shows a time $u(p)$ such that $l(p, t) < u(p) < h(p, t)$.

A state is characterized by the marking of system plus the positions of the local clocks at the places. A state is called dead if no transition is time-enabled and no transition would become able to fire after any increments of the clocks. If in state S_i there is such a minimum increment Δ that some of the transitions become enabled after elapsing it, then it is said that the state transition $\tau : \mathsf{S}_i \rightarrow \mathsf{S}_j$ has a ''delay'' Δ. Conversely, it can be interpreted as the state S_i has a ''duration'' Δ, that specifies the time increment of the clocks of this state required to make the transition τ enabled.

At a given state all (time-)enabled steps have to be computed and placed into the list of enabled steps. Firing of each step brings one more state successor to the current state. Repetitive application of this procedure to every subsequent state forms the reachability space of the model. Time-enableness is a required but not sufficient condition to include transition to the firing step. The interpretation of the timing intervals is defined by the timing firing rule. Several such rules have been studied:

(1) *Strong vs. weak firing*: with the strong rule all marking enabled (spontaneous) transitions, which have pre-places with clock position equal to either low or high time limit, are obligatorily inserted into the step (can be specified to make for example, either strong earliest firing rule, or strong latest firing rule). If the weak rule is chosen then at least one of the enabled spontaneous transitions has to be included in step.

(2) *Earliest vs. interval firing*: in case of the interval firing a transition is time-enabled at every clock position within the interval [*l*,*h*]. In the earliest firing rule a transition is time-enabled if it has a pre-place with the clock value equal to the low bound *l* of the time interval.

(3) *Ultimo firing*: is a certain combination of the interval and strong rules: a transition is time-enabled every time tick within the interval and must fire at the latest at clock position equal to *h*.

It is necessary to mention that in case a transition has several incoming arcs with permeability intervals $[l_1, h_1], [l_2, h_2], \ldots [l_n, h_n]$ then all arcs have to be permeable for the firing, that means $l = \max(l_i)$, $h = \min(h_\mathrm{i})$. Among all possible combinations of time constants and time-firing rules, some were found of interest in some industrial applications. These combinations are presented in Table 1.

The lower or higher time limits may or may not (depending on the corresponding rule) force transition to fire. The ''interval'' firing rule accepts presence of empty transition steps, when time elapses even in the absence of any enabled transitions. This option may be useful if aimed at finding of all possible combinations of overlapping processes and, correspondingly, simultaneous events. On the other hand, it

Table 1. Combinations of time-firing rule and time intervals commonly used for modeling

	Time onstants	*Firing rule*	*Interpretation*
1.	$l > 0, h \geq l$	Interval, weak	Event is expected with minimum delay *l*, maximum delay *h*, or may not occur at all.
2.	$l > 0, h \geq l$	Ultimo	Process must get terminated within the interval [*l*,*h*].
3.	$l > 0, h = \infty$	Earliest, strong	Process has duration *l*, and all simultaneously started processes with the same duration finish simultaneously.
4.	$l > 0, h = \infty$	Earliest, weak	Process has duration *l*, but termination of all processes with the same duration may be not synchronized.

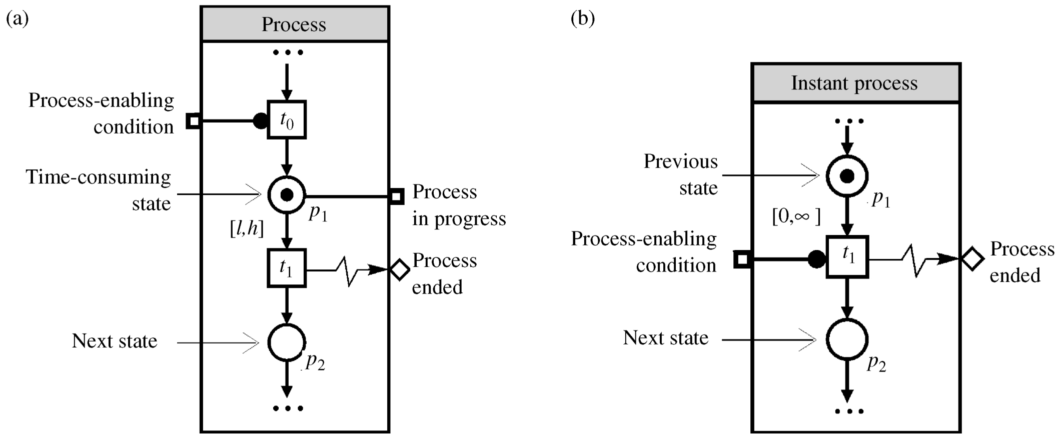

Fig. 8. (a) Model of plain time consuming process; (b) Model of instant process.

obviously explodes size of the reachability space. The (may be confusing) variety of choices extends the modeling horizons and allows to describe models more concisely.

5. Modeling of processes

Underlying (sub-)processes which represent on abstract levels (sub-) states of the model can be modeled in SNS as places with safe (0/1) marking. Each stable state can be also explicitly associated with a duration. Figure 8(a) shows an example of such a model. The time consuming state, represented by p_1, models an action in progress. Transition t_0 has an incoming condition arc which starts the process. The process is in progress while p_1 is marked. Permeability interval of the arc (p_1, t_1) defines the duration of the process in one of the ways described in the previous section.

There are some processes which have no duration, or it can be neglected in comparison to other ones. Such processes are called ''instant''. An example of a model of such a process is given in Fig. 8(b). The process is started by ''enabling condition'', but there is no place marking corresponding to the state ''process in progress''.

More realistic modeling of processes (as the one shown in Fig. 9) may include exception states in which the system comes in case of abnormal process termination. The place p_1 (modeling the time-consuming state as in the previous example) is connected with two transitions: t_1 stands for the normal operation mode with a duration as described in the previous case, while t_2 models the exception state, which may occur anytime within the normal operation time. This model is most adequate with the ''ultimo'' firing rule and $h < \infty$. The reached upper time limit forces only one of the transitions: either ending the action normally, or exceptionally.

The described models of atomic processes can be combined to more complicated ones. Consider the carriage from the example presented above. The corresponding model is given in Fig. 10. The model represents the autonomous carriage and the workpiece including the logic of its placing/removing. Every component of the system is modeled by a module. 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:

- locations and states of the carriage;
- setting/removing of the workpiece;

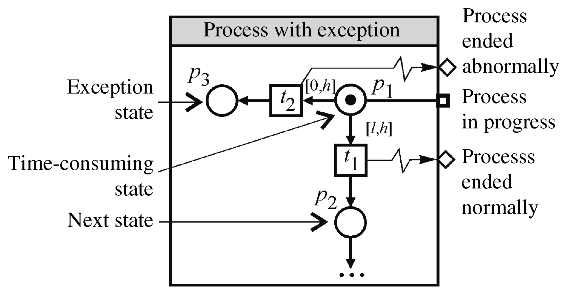

Fig. 9. Model of process with a possible exception.

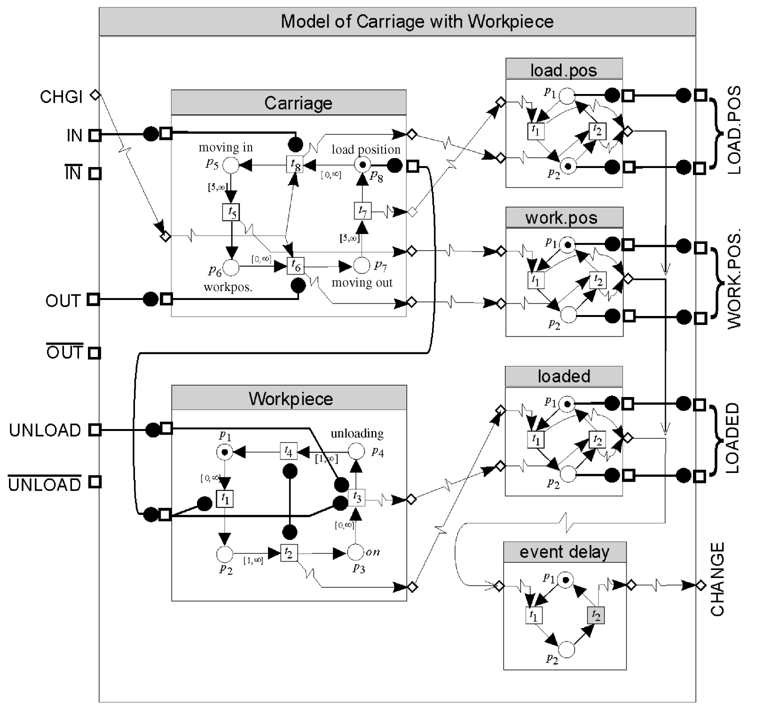

Fig. 10. Modular NCES model of the carriage with workpiece.

- 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 the 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 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. The model can substitute the function block ''CARRIAGE'' in Fig. 3 since it has an identical input/output interface (except for the output COORD—the numerical value of the coordinate.).

Some aspects are worth mentioning before we proceed to the application of the presented modeling technique for verification. These aspects are:

(1) We have an intuitive way to model a system which closely corresponds to the design and engineering practice and trends.

(2) The modeling technique supports a bottom-up modeling as well as top-down strategy. One could start with a set of modules and create a larger model by composing them. On the other hand, one could start with a larger system and could decompose it to a set of subsystems (expressed by modules) and a set of interconnections (expressed by composition arcs).

(3) Even after composition, the state-of-the-system is distributed, and the original structure of the modules is preserved. Even removing a module and replacing it by another module with the same input/output interface would be a local operation over the structure of the model. Hence, composition is far less complicated as building the cross product of automata or the interleaving language. This allows us to build models of realistic scale efficiently.

(4) Discrete state formalisms obviously have more limited expressive power than the hybrid ones, such as

discussed in Kowalewski *et al*. (2001) and Hanisch *et al*. 2001. This is true also for NCES. However, currently existing implementations of SNS model-checking, such as SESA (1993), allow the model-checking of realistic-scale applications, having millions of reachable states, that is hardly to expect from the hybrid formalisms.

The models of the controller function blocks are generated automatically by VEDA given their source code. The methodology of the modeling is discussed in our prior works (Vyatkin and Hanisch, 2000a; 2000b). Our verification approach essentially requires the closed-loop interconnection of the plant and controller models by data and event signals. Thanks to the distributed nature of the systems being studied, the resulting models may contain several of such closed-loop connected components.

6. Verification tools

In this part, we illustrate the application of the developed integrated software tool VEDA. Overall correctness of distributed control applications depends not only on the control logic and data processing within the algorithms, but also on the scheduling of the algorithms, communication between devices, particular architecture of the system, and mapping of components of the application to a particular architecture. The description of the control system following IEC61499 allows to model all these issues and take them into account in the process of verification. Though, at the current stage of the work VEDA supports modeling of rather primitive function blocks, it allows to illustrate various qualitative benefits of IEC61499.

Given a control system, designed and coded according to IEC61499, the following steps are assumed as the preceding to VEDA application:

(1) Develop the NCES models of plant components and integrate them into the IEC61499 control applications as discussed above. The design of the models is supported by means of the graphical editor as presented in Fig. 11. The formal model can be appended by a visualization model, which assigns visualization actions to certain markings of places in the formal model. The resulting composite model can be presented as a function block with an interface following IEC61499. The arcs linking the block in the original application to other function blocks are substituted in the model by condition and event arcs connecting the model of the block with the models of other respective blocks. We substitute this way the block CARRIAGE in Fig. 3 by the model from Fig. 10.

(2) Formalize the specifications of desired or forbidden behavior of the control system. This can be done using second order predicates or temporal logic formulae over the variables declared in the controllers or over parameters of the model of plant.

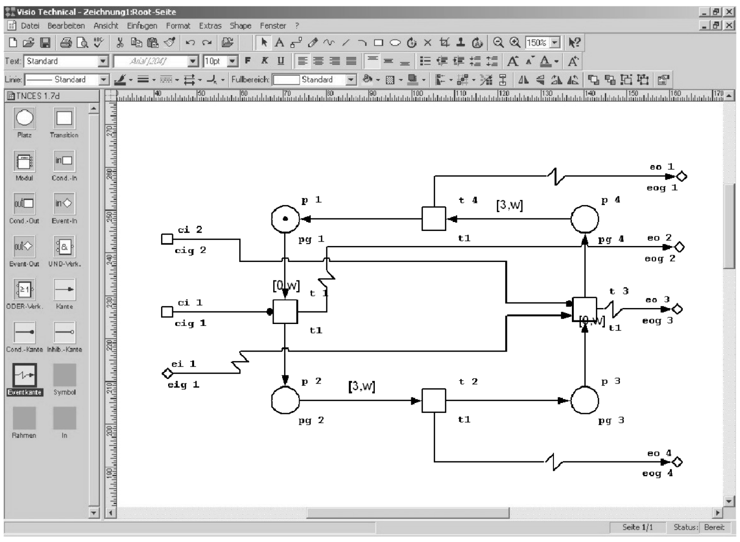

Fig. 11. Graphical editor of NCES models.

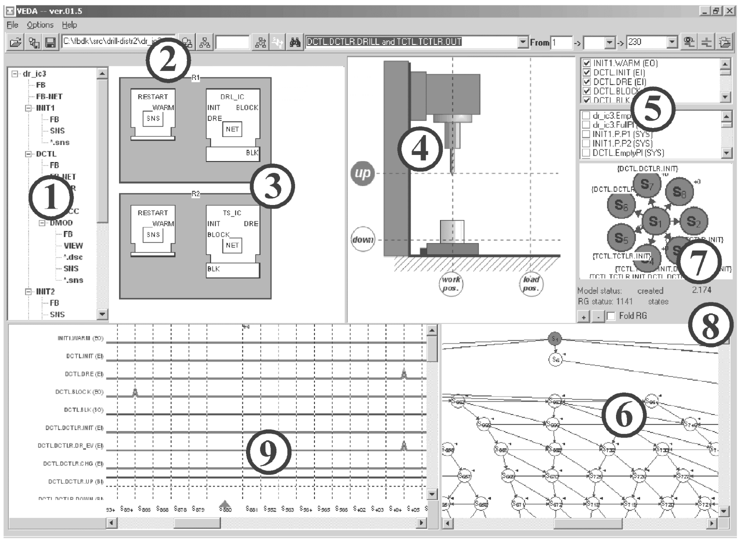

Fig. 12. VEDA's display.

When the above steps are fulfilled, the application can be loaded into VEDA. The main screen of VEDA is shown in Fig. 12. Its structure is as follows:

(1) The tree view shows the hierarchial structure of the application and a variety of presentation options, which can be applied to each component. This means that a block containing the formal model of plant can be seen as its interface (inputs and outputs) as well as the textual or graphical form of the NCES model (in window 1). A ''pure'' IEC61499 basic function block is visible as its interface, as structured text of its content, or as a graphical view of its execution control chart. The graphical views can display information about particular state of the model (marking of places, etc), or about the state of the execution control.

(2) Modeling/model-checking controls implement the following functions: create the reachability space of the model, search for the states satisfying certain logic conditions, or more sophisticated navigation functions in the reachability space, search for a trajectory satisfying a number of conditions, expressed either as a sequence of (second-order) predicates, or as a temporal logic formula.

(3) Application/model source view displays the application being verified in one of the available graphical or textual formats. The IEC61499 overall structure, for example, can be presented as a net of interconnected component function blocks as well as an equivalent ASCII text presentation. The same applies to every component function block.

(4) Process visualization display shows the modeled plant in the state selected in the reachability space. For the states having non-zero time duration, process animation can be displayed.

(5) Lists of input, output, and internal variables of the original application allow to select the desired set of variables to observe in window the state/time diagrams of their values.

(6) The behavior of the interconnected model is represented by means of its reachability graph, where nodes correspond to the states, and arcs represent transitions between the states. Thus, a path (a trajectory) in the reachability graph corresponds to a particular scenario in the behavior of the model. The specified path is highlighted in the graph.

(7) Selected state of the reachability graph is displayed in this window with all outgoing states and with state transitions marked by the variables initiating them.

(8) Statistics of the model and the model-checking status shows the size of the generated model, the size of the reachability space, and results of the search for a particular state.

(9) The system's behavior along the highlighted trajectory can be represented for detailed analysis by means of asynchronous timing diagrams. Values of

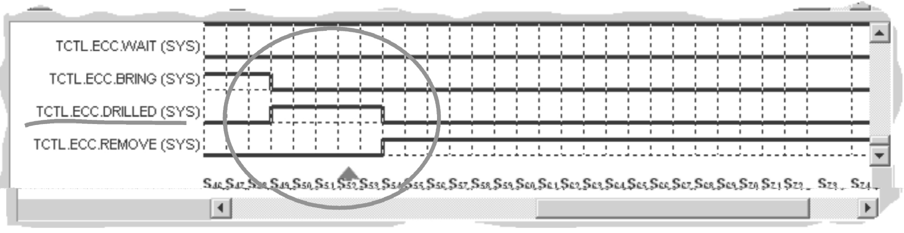

Fig. 13. Timing diagram of the state activity variables: state DRILLED of the transfer stage is transient.

the requested inputs, outputs or internal variables can be shown graphically with respect to the states of the trajectory. In addition, each state can be visualized in the animation window (5), and in the source window (1).

Let us illustrate how VEDA is applied to make sure that our application never forces the system to the dangerous state specified as follows: ''Attempt to move the transfer stage during drilling''. This condition can be represented as a predicate in terms of input/output variables of the blocks as follows: ''DR_CTL.DRILL and CAR_CTL.OUT''. VEDA checked the validity of the predicate in the reachability space of the model (2207 states) and found states where the condition holds. To analyze the reasons of the incorrect behavior, it is possible to visualize the trajectories leading to such states with state/time diagrams of the variables, and provide the view of the animated process visualization display along it. The reason becomes clear after a look at the signal diagram in Fig. 13 and to the corresponding visualization of ECC (as in Fig. 4) and process view.

Once the carriage arrives at 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 carriage to move away before the drilling is completed. In fact, the state DRILLED in the controller of the carriage happens to be unstable: the condition of 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 CAR_CTL is changed to 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 AND NOT BLK, where BLOCK is an event issued by DR_CTL after setting BLK to 1. After the modification, the erroneous states disappear from the reachability space.

The repetitive application of such a procedure with various specifications helps to improve the controller's correctness.

7. Conclusion

The use of industrial standards provides easy network integration of our verification approach, as shown in Fig. 14. Verification that requires high computing power can be used as an external web-service for control engineers developing and testing control systems. Description of the latter in IEC61499 could

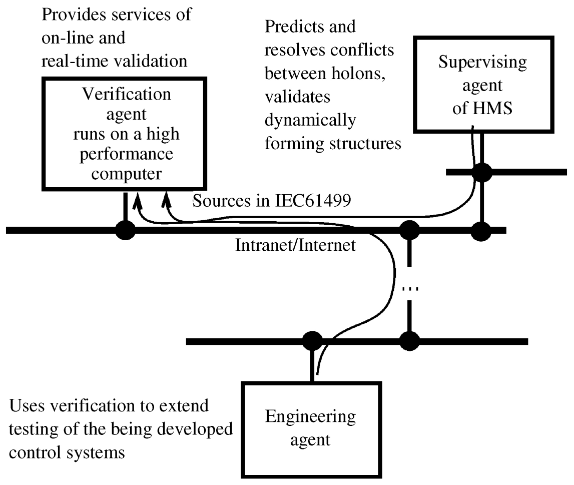
Provides services of on-line and real-time validation
Predicts and resolves conflicts between holons, validates dynamically forming structures
Supervising agent of HMS
Verification agent runs on a high performance computer
Sources in IEC61499
Intranet/Internet
...
Uses verification to extend testing of the being developed control systems
Engineering agent

Fig. 14. Possible web-integration of the verification agent.

be sent to the ''Verification agent'' along with specifications of desired/forbidden behavior. The server would perform model generation, model-checking, and return of results to the client.

Moreover, it can be used in automatic mode by supervising agents of HMS. It is possible to imagine dynamically arising configurations of holons which were not completely tested during the engineering of the system. In such cases the supervising agent might validate them upon appearance before allowing to function. This service can be requested from one of available appropriate verification agents.

There are two major differences of the approach to verification proposed in this paper with other similar works. First, it applies to the domain of distributed systems and takes into account various ''adjacent'' issues in addition to pure control logic. Second, it works with standardized source code of controllers, that simplifies its application in the engineering practice. These arguments give a hope on eventual applicability of this work in industry.

Major doubts on practical applicability of the proposed approach concern two issues: functional constraints necessary to impose on original control systems in order to verify them, and closely related issue of computational complexity of the formal model-checking. Though we intentionally did not address those issues in the paper, certainly any researcher runs into them dealing with similar problems. Currently existing model-checking tools allow handling of systems having up to billions of reachable states. This means that many industrial systems, such as machine tools, or automation cells could be modeled quite precisely for the verification purposes. New challenges of intelligent manufacturing would make the formal methods of validation simply unavoidable, while the progress in the information technology ensures constant extension of the horizons of applicability of such methods.

Practical application of the verification in industry is in a very early stage, and it is quite likely that it is limited by other reasons than pure computational constraints. We hope that our work highlights some of the problems that have to be addressed along with waiving the computational limits. In fact, some of the solutions applied in our work, such as application of SNS rather than automata, and the closed-loop way of model composition already helped to minimize the computational complexity of the model-checking essentially.

Acknowledgments

The work is supported by the Deutsche Forschungsgemeinschaft under reference Ha 1886/10-2.

Notes

1 In case of weighted arcs, with as many tokens as the weight of the corresponding arc from the pre-place to the transition.

References

Christensen, J. H. (2000) Design patterns for system engineering with IEC 61499. *Proceedings of Conference ''Verteile Automatisierung'' (Distributed Automation)*, Magdeburg, Germany, pp. 63–71.

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

Function Blocks for Industrial Process Measurement and Control Systems (1998), Publicly Available Specification, International Electrotechnical Commission, Technical Communication 65, Working group 6, Geneva.

Hanisch, H.-M., Lautenbach, K., Simon, C. and Thieme, J. (2001) Modeling and validation of hybrid systems using extended timestamp nets. *Automatisierungstechnik*, **2**, 60–65.

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

Hanisch, H.-M. and Lüder, A. (1999) Modular modeling of closed-loop system. *Proceedings of Colloquium on Petri Net Technologies for Modeling Communication Based Systems*, Berlin, Germany, October, pp. 103–126.

Hanisch, H.-M., Thieme, J., Lüder, A. and Wienhold, O. (1997) Modeling of PLC behavior by means of timed net condition/event systems. *International conference on Emerging Technologies and Factory Automation (ETFA'97)*, Los Angeles, USA, September.

IMS (Intelligent Manufacturing Systems) programme, project on Holonics, http://hms.ifw.uni-hannover.de/public/overview.html⟩.

International Standard IEC1131-3 Programmable Controllers—Part 3 (1993) International Electrotechnical Commission, Geneva, Suisse.

Kowalewski, S., Herrmann, P., Engell, S., Huuk, R., Krumm, H., Lakhnech, Y., Lukoschus, B. and Treseler, H. (2001) Approaches to the formal verification of hybrid systems. *Automatisierungstechnik*, **2**, 66–73.

Lewis, R. (2001) Modeling Control Systems using IEC 61499, IEE, London.

Ostroff, J. S. (1989) *Temporal Logic for Real-Time Systems*, Wiley, London.

SESA—signal/net system analyzer. Humboldt Universität zu Berlin, Institut für Informatik, http://www.informatik.hu-berlin.de/lehrstuehle/automaten/tools/⟩

Starke, P., Roch, S., Schmidt, K., Hanisch, H.-M. and Lüder, A. (July 2000) Analysing signal-event systems. Technical report, Humboldt Universität zu Berlin, Institut für Informatik, http://www.informatik.hu-berlin.de/lehrstuehle/automaten/tools/asen.ps.gz⟩.

Vyatkin, V. and Hanisch, H.-M. (1999) A modeling approach for verification of IEC1499 function blocks using net condition/event systems. *Proceedings of ETFA'99 Workshop*, Barcelona, Spain, pp. 261–270.

Vyatkin, V. and Hanisch, H.-M. (2000a) Development of adequate formalisms for verification of IEC1499 distributed applications. *39th Conference of Society of Instrument and Control Engineers* (SICE) of Japan, Iizuka, Japan, July.

Vyatkin, V. and Hanisch, H.-M. (2000b) Modeling of IEC 61499 function blocks as a clue to their verification. Workshop on Supervising and Diagnostics of Machining Systems, Karpacz, Poland, March.