
Valeriy Vyatkin, Hans-Michael Hanisch

Automation Technology Group,
Department of Engineering Science,
Martin Luther University Halle-Wittenberg,
Kurt-Mothes Str. 1, 06099 Halle
GERMANY
Tel.: +49/345/55 25 972
Fax: +49/345/55 27 304

e-mail: Valeriy.Vyatkin@iw.uni-halle.de
Hans-Michael.Hanisch@iw.uni-halle.de

Extended abstract

Introduction

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 if 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-reconfigurable 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 development of 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 to be executed. The architecture can be described later, or modified independently of the core description of the application.

**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 complexity of the systems under control, provided level of assurance in error-detection is also far from satisfactory.

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 a 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 separate out the models of plant and controller in order to provide means of independent controller or plant modification. Then specifications of the correct or incorrect 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 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:

1. Formal modeling is not easy for the 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.
2. Modeling of interconnected plant/controller systems requires formalisms equally good for both plant and controller.
3. Formulation and formalization of specifications are tricky.
4. The model-checking is computationally complex, and is not always supported by a proper Human - Machine Interface. Overcoming of these difficulties was the aim of developing of the Verification Environment for Distributed Applications (VEDA) [5,6], that is a software package for model-based simulation and verification.

Figure 2. VEDA provides source code-based verification and simulation of IEC61499 control applications.
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 a graphic editor for SNS models. It 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 failures and fix them.

The verification trials, conducted so far with VEDA revealed, however, that despite 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.

**Next Step: Integration into Software Engineering Frameworks**

One of the reasons preventing the formal approaches from becoming a part of the 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 the 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 programming language for PLC. It gives an opportunity to take in 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

![Figure 3 Part of control application in IEC61499 MVC context.](image)
typical component of control systems – a processing (drilling) station with a carrier bringing work-pieces. 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 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 implement 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 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 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 a description of the model in a hybrid modeling formalism is required. We assume that it is given as a hybrid automata following 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 original application (performed by VEDA automatically).
5. Perform the model-checking using the interconnected SNS model. In case if the model-checking revealed some states in the reachability space, where the specification does not hold, generate trajectories to these states and visualize 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.

We apply this approach using VEDA and the prototype implementation of the IEC61499 known as FBDK, which implements function blocks translating them to Java. The Model and View function blocks are 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 of behavior is provided without essential overheads.

Conclusion

In general, the verification can be seen as the means of self-testing of agile manufacturing systems. In a particular configuration of the system (or even before it has formed), the supervising controller checks (by means of verification), if co-existence of the current components (with their scheduled tasks) may lead system to a number of known (or inferred) erroneous situations.

This task can be accomplished using the source codes of control-applications of the components (say given in IEC61499 format), and descriptions of possible failures or rules, how the latter can be inferred.

We hope that the presented work partly helps to bridge the gap on the way to this distant goal.

References:


