Vyatkin, Valeriy; Rumiantsev, Roman

Framework for Faster-Than-Real-Time Testing of IEC 61499 Applications with Embedded Process Simulation

Published in:
Proceedings - 2024 IEEE 22nd International Conference on Industrial Informatics, INDIN 2024

DOI:
10.1109/INDIN58382.2024.10774354

Published: 01/01/2024

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Framework for Faster-Than-Real-Time Testing of IEC 61499 Applications with Embedded Process Simulation

Valeriy Vyatkin*†, Roman Rumiantcev*
*Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
†Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden

Email: vyatkin@ieee.org, roman.rumiantcev@aalto.fi

***Abstract*—This paper presents a novel framework for testing automation applications compliant with the international standard IEC 61499 and including process simulation. The framework enables automation programs to be run in testing mode faster than real time, validated by an external tester program. Application of the approach is illustrated on example of a simple component-based system.**

***Index Terms*—Faster-than-real-time simulation, Simulation in the loop, Automatic testing, IEC 61499**

I. Introduction

Automation systems are evolving to be more decentralised and flexible, frequently integrating process control, discrete control, and motion control, which poses challenges to the design, development, and testing phases. IEC 61499 [1] is an international standard for distributed control and automation systems that provides a framework for distributed automation that aids in the development of advanced automation systems incorporating numerous, sometimes even hundreds of devices. The deployment of these systems poses substantial challenges for developers and integrators. The IEC 61499 standard offers a general model for describing distributed systems, with functional blocks as the fundamental units. The standard sets the guidelines for the description of industrial systems based on functional blocks and establishes the rules for their execution, configuration, and management. It also specifies a set of requirements for instruments and devices that operate with compliant software. The standard was published in 2005 and since then many compatible tools and devices have been developed [2], [3].

Testing of automation applications is often done with a simulation model in the loop. However, this may be a time consuming process as the developers need to observe system's behaviour for the duration of simulation. Applying automatic testing approaches still will be limited by the duration of the simulated process. To run such closed loop systems in faster than real-time mode could affect the timing behaviour of the controller and requires manual modifications of the code.

In this paper, an approach is proposed to avoid the mentioned complications and enable automation programs to be run in testing mode faster than real time, validated by an external tester program. In regular operation mode the same programs would run as normal.

The rest of this paper is structured as follows. In Section II a short survey of related works is provided. Section III presents the architecture of the testing framework. Section IV elaborates on the modelling framework of the applications under test which is necessary for understanding of the method. Section V describes the testing algorithm, followed by use case in Section VI, and summary in Section VII.

II. Related works

The closed-loop application development pattern, introduced in [4] and based on the Cyber-Physical Component (CPC) concept, aims at facilitation of testing by simulation on account of using embedded simulation models in each component and simultaneously integrating the control and simulation functionalities. The CPC, in turn, bases on the adaptation of MVC design pattern to IEC61499 by Christensen [5].

The pre-commissioning testing of the applications is a common practice in industry. Several approaches were proposed to introduce the testing techniques utilising the IEC 61499 standard [6], [7]. However, in all of the related papers the communication between the testing environment and the running program was established by either the introduction of special, dedicated for this purpose, libraries installed in the solution or plugins for the runtime. In this paper we cover a novel approach of monitoring and testing the behaviour of the system solely by utilising the functionality of the built-in instrumentation provided by the Programmable Logic Controller (PLC) manufacturer, IDE and the executable Python program.

III. Testing architecture

The proposed testing framework is based on the generic monitoring pattern [8] as depicted in Figure 1.

The controller software is connected in closed loop with the plant model and in another closed loop with human-machine interface. The Monitor&Tester (MT) component receives information on the current plant state from the plant model. The

plant model needs to be designed in a way that allows MT to switch between normal operation and testing mode. This can be done by exposing the corresponding mode flag via OPC UA or using the symbolic link mechanism of IEC 61499.

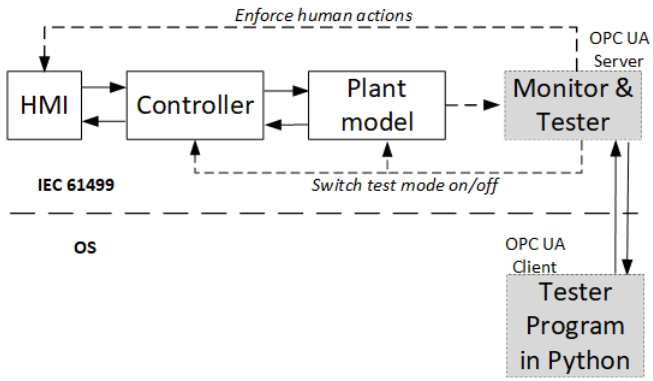

Fig. 1: Generic monitoring pattern.

The MT function block communicates with an external program (in our case, written in Python) which contains implementation of the testing algorithm. This architecture enables to perform the testing of function block systems in the runtime without involvement of human, in automatic mode.

IV. Modelling Framework

The HMI-Controller-Plant application can be implemented straightforwardly using IEC 61499 function blocks as depicted in Figure 2, top layer. Here it is exemplified for the case of a dual-acting pneumatic cylinder. The plant model, however, is implemented in several layers, resulting in function block nesting. This complexity is the the downside of the simplicity of one-line adapter interfaces, but also reflection of the physical complexity of the object being modelled. Here, the CylinderHAdpOL layer represents implementation of the model FB type which is equipped with adapter interfaces for closed loop connection to controller and open-loop connection to the (optionally connected) tester. The following CylinderH_SA layer represents a ”structural” function block model of the cylinder that includes its mechatronic building parts: two end-position sensors and two pneumatic valve actuators, connected with the model of the cylinder itself. The linear motion process of the cylinder's stock is modelled in the CylinderH function block, which contains a model of cylinder's dynamics and its visualisation. The dynamics is modelled in the CylinderHC function block as a combination of a hybrid state-machine (CylinderH) with Integrator and Clock generator. Hybrid in this context means continuous-discrete, following the definition of Hybrid State Machines in [9].

The core part of the simulation engine is depicted in more detail in Figure 3. Here the hybrid state machine function block, which models the behaviour of the physical process, is connected with the integrator function block which is calculating the change of continuous parameters of the process

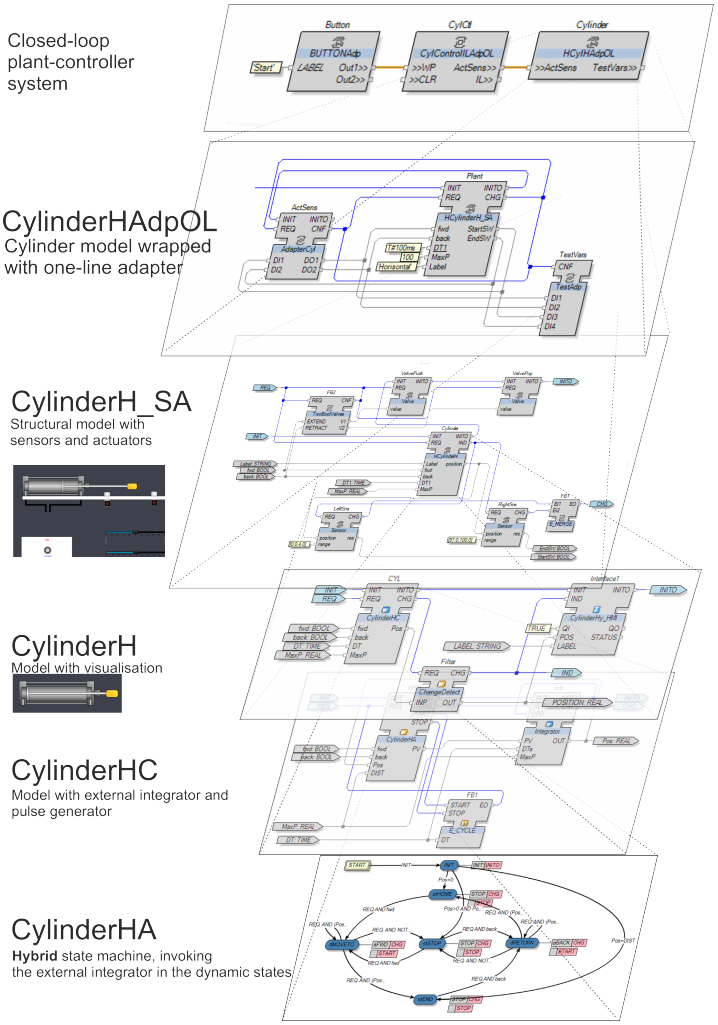

Fig. 2: Layered architecture of the closed-loop modelling framework.

every clock interval as driven by the clock generator function block CLK. The latter is started and stopped in certain states of the hybrid state machine.

V. Testing Algorithm

Unlike previous research discussed in Section II, our system broadcasts data from a PLC with an enabled OPC UA server to a separate, external device. This device processes and stores the data to conduct tests and enhance monitoring using a Python algorithm.

Testing and monitoring were transferred to the external device in order to preserve and optimise the computational capabilities of the PLC. This approach utilises settled Python-based techniques, enabling a broader range of functionalities and simplifying the integration of diverse testing methodologies.

Scenario-based testing, a well-established software testing technique [10], was chosen to prove the feasibility of the proposed testing framework. The core concept around which the scenario-based tests are executed is “scenario”. A scenario

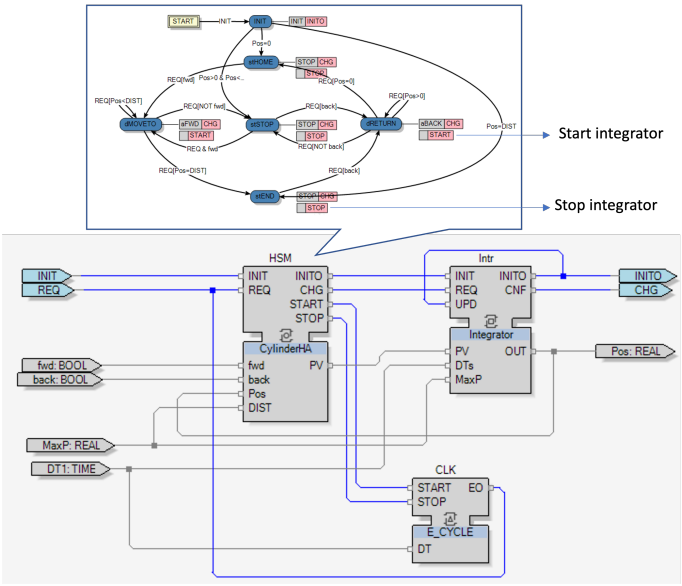

Fig. 3: Hybrid state machine implementation of the simulation.

defines the desired behaviour of the system by specifying expected changes in variable values over time. While future research will explore the potential of automated Python code generation from UML diagrams, this study employed a manual, hardcoded implementation of the scenario.

This scenario serves as a benchmark against which the actual system performance is compared.

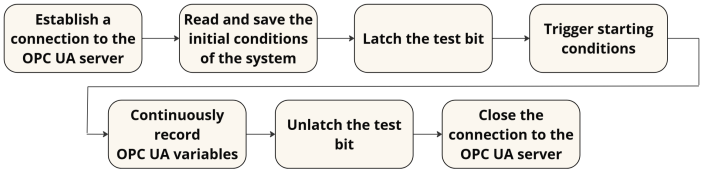

Fig. 4: Algorithm of the tester implemented in Python.

The overall Python algorithm depicted in Figure 4 could be described as follows:

- Establish the connection to the OPC UA server and start the OPC UA client. For that purpose *opcua* library was used.
- Read the initial conditions of the systems which would be later used to identify the end of the run cycle of the PLC program running on the PLC.
- Latch the test variable. It is used for the model to switch into the test state.
- Trigger the start conditions. For the use case described later in the paper the start condition was pressing a start button.
- Continuously read the and record the OPC UA variables to later compare the behaviour of the simulation with the desired one.
- Once the initial conditions were reached the test variable is unlatched.
- The last step is terminate the OPC UA communication by turning off the OPC UA client.

A. *Operating states*

To modify the simulated closed-loop solution, two distinct operating states were defined. The first, an automatic state, represents the solution's normal operating condition where no testing is conducted. The second state, designated as the testing state, enables the substitution of input signals from the I/O and HMI with variables retrieved from the OPC UA client.

A dedicated test variable serves as a trigger for transitioning between these states. This variable is set by a Python program and transmitted to the Programmable Logic Controller (PLC). The same test variable is also used to reduce the simulation model's clock cycles.

B. *Modifying application timing for faster than real-time execution*

The closed loop model includes simulation model with that is driven by clock pulse generating timers of quite small time scale (20-100 ms) to ensure smoothness of continuous process modelling. The testing algorithm would reduce this pulse interval to the shortest modellable time in the runtime, that is usually 1 ms.

It is assumed that timers in the controller part of the application are on a longer time scale, e.g., seconds or minutes.

VI. Use case

TwoCylinders is a simple system where two pneumatic cylinders positioned as seen in Figure 5 are activated simultaneously by one button. The activation, if applied to a single cylinder, leads to cylinder's complete extension followed by retraction to the initial position.

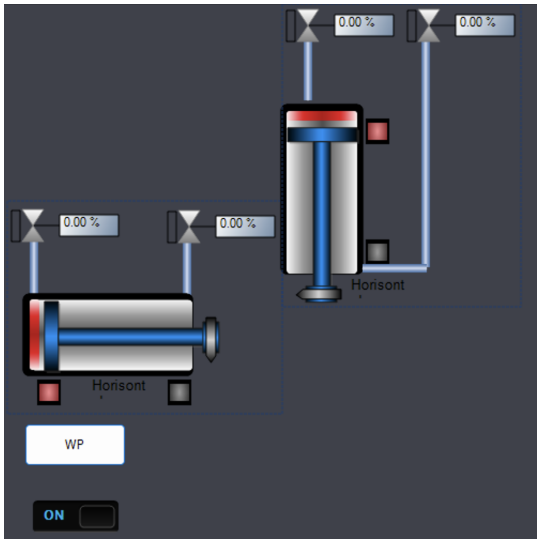

Fig. 5: System of two cylinders.

The application implementing control and simulation of the two cylinders example is depicted in Figure 6. Here one can see that each cylinder is independently controlled

by its controller function block. For example, the horizontal cylinder is represented by function block HCylPlant, which is connected by an adapter link to the controller function block CtlH. The adapter link includes bi-directional flow of event and data signals. The vertical cylinder is represented by VCylPlant, connected to the controller CtlV.

The function block TestFB (surrounded by the red dashed line) is added to the application to enable and implement the testing regime, by connecting it to the testing program in Python.

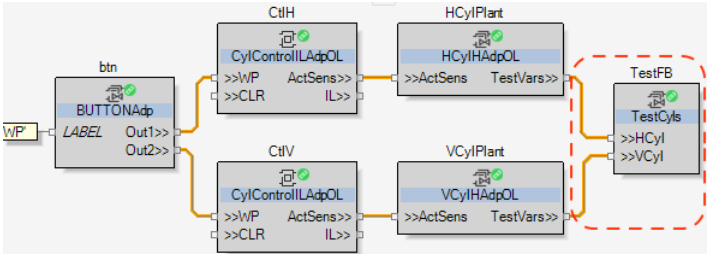

Fig. 6: Comparison of the initial and modified solutions.

The TestFB function block serves as a primary interface for transmitting and receiving data through the OPC UA protocol. Data destined for transmission is acquired within the application using dedicated adapters. Conversely, data received from the OPC UA client is distributed throughout the application by means of symbolic links.

The inside of TestFB is presented in Figure 7. The connection to the Python program is implemented via OPC UA variables associated with inputs and outputs of the FB OPC_UA_con. For testing purposes the RF is manually set to value 20.

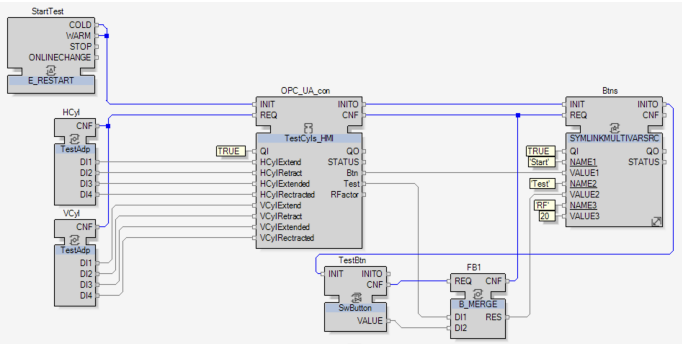

Fig. 7: Composition of the TestFB function block.

A. Testing and monitoring

In order to describe the behaviour of the system 8 variables were identified. These variables include: horizontal cylinder extract ($HCylExtract$), retract ($HCylRetract$), extracted ($HCylExtracted$), retracted ($HCylRetracted$) and the same variables for the vertical cylinder ($VCylExtract$, $VCylRetract$, $VCylExtracted$, $VCylRetracted$).

To prove the feasibility of the concept, minor modifications were implemented in the initial design. These changes resulted in a deviation from the expected behaviour depicted in Figure 8. Specifically, the extraction of the pistons was terminated before reaching their fully extended state, directly influencing the retraction time of the cylinders.

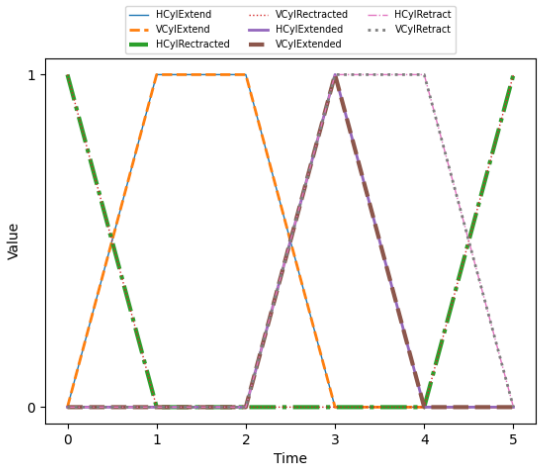

Fig. 8: The desired system behaviour.

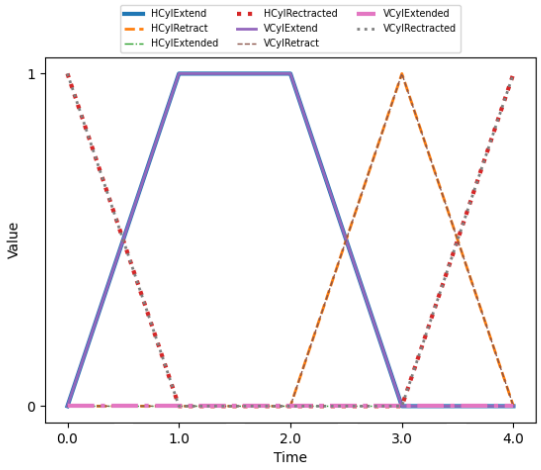

Fig. 9: The actual system behaviour.

By visual investigation of the plot in Figure 9 it is seen that the variables $HCylExtracted$ and $VCylExtracted$ never change their values as well as the time required for cylinders to retract decreased by 1 ms as a result of the implemented changes to the application.

VII. Conclusions and Discussion

Increasing complexity of the applications based on IEC61499 standard requires implementation of the various testing approaches. Introduction of the additional functionality to the applications is usually time consuming, complex and rarely can be performed on the go. The proposed testing framework enables easy modification of closed-loop component applications with embedded simulation to run in the faster than real time mode, which enables their automatic testing.

The tester extension interface is added to the application by the developer to enable connection of the application to the external tester program during the run-time.

This paper focuses exclusively on full system scenario-based testing, excluding other techniques such as unit testing, continuous integration, and regression testing. While Python and OPC UA offer significant potential for integrating multiple testing techniques to the IEC61499 applications, their full implementation remains an area for future research.

Currently, testing is conducted partially visually. Future work will incorporate algorithms to compare time series data of the desired and actual system behaviour.

Another opportunity for future research is the automation of the application preparation for testing. This involves automatic introduction of a dedicated TestFB into the application. One potential solution for this aspect could involve swapping XML files containing the application description during performance of tests.

Overall, this testing framework offers a foundation for further research into automated testing methodologies using the external computing resources, while enhancing the efficiency and effectiveness, reducing time and complexity of the system validation.

ACKNOWLEDGMENT

This work was supported, in part, by the Aalto-R2B-CloViC project (Dnro 7221/31/2023) funded by Business Finland.

REFERENCES

[1] "IEC 61499-1: Function blocks part 1: Architecture," 2012.

[2] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[3] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[4] G. Zhabelova, C.-W. Yang, S. Patil, C. Pang, J. Yan, A. Shalyto, and V. Vyatkin, "Cyber-physical components for heterogeneous modelling, validation and implementation of smart grid intelligence," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2014, pp. 411–417.

[5] J. H. Christensen, "Design patterns for systems engineering with IEC 61499," in *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung (VA2000)*, 2000, pp. 63–71.

[6] R. Hametner, I. Hegny, and A. Zoitl, "A unit-test framework for event-driven control components modeled in iec 61499," in *2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[7] R. Hametner, D. Winkler, and A. Zoitl, "Agile testing concepts based on keyword-driven testing for industrial automation systems," in *2012 38th IEEE Annual Conference on Industrial Electronics Society (IECON)*, 2012, pp. 3727–3732.

[8] P. Jhunjhunwala, J. O. Blech, A. Zoitl, U. D. Atmojo, and V. Vyatkin, "A design pattern for monitoring adapter connections in IEC 61499," in *2021 22nd IEEE Int. Conf. Ind. Technology (ICIT)*, vol. 1. IEEE, 2021, pp. 967–972.

[9] T. Henzinger, "Theory of hybrid automata," in *Verification of Digital and Hybrid Systems (M.K. Inan, R.P. Kurshan, eds.)*, ser. NATO ASI Series F: Computer and Systems Sciences. Springer-Verlag, 2000, vol. 170, pp. 265–292.

[10] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Wiley Publishing, 2011.