EPiC Series in Computing

Volume 48, 2017, Pages 92–99

ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems

Testbed for Model-based Verification of Cyber-Physical Production Systems

Christof J. Budnik[1], Sebastian Eckl[2*], and Marco Gario[1]
[1] Siemens Corporate Technology, Princeton, NJ, USA
[2] Technical University of Munich (TUM), Munich, Germany
{christof.budnik, marco.gario}@siemens.com, sebastian.eckl@tum.de

Abstract

Cyber-physical production systems (CPPS) build a network of industrial automation components and systems to enable individualized products at mass production costs. Failures or vulnerabilities in CPPS can be life threatening and can cause physical damage while hiding the effects from monitors. Thus, software verification and validation methods need to analyze the dynamics and behavior of CPPS. In this work, we present a hybrid testbed used in Siemens Corporate Technology. The testbed combines a physical CPPS together with its virtual simulated counterpart, allowing us to verify the system using runtime monitoring, model-based testing, simulation and formal techniques.

1 Introduction

Future industrial automation is enabling new production processes where products will drive their production by cyber-physical production systems (CPPS) without the need for human intervention (Kephart & Chess, 2003). A CPPS needs to be able to fulfill its specified goals effectively and efficiently under changing conditions. CPPS intrinsically combine hardware, software, networking, and physical systems. Today, the modeling and design abstractions used for hardware and physical systems are entirely different from those used for software, and few modeling or design languages support mixtures of the two. This makes it harder to model, harder to design, and harder to analyze CPPS. Nevertheless, CPPS are safety critical systems that need to provide a high-level of confidence in their safety and functionalities under multiple operating conditions.

Field tests for CCPS provide high-fidelity information on the behavior of the system by enforcing realistic conditions for testing, and by considering the integration of all the components in the entire production unit. Field tests are expensive, and require a significant effort in both time and resources. For this reason, only a limited set of potential scenarios can be tested, and hazardous situations are

[*] Worked as intern at Siemens US when the majority of this work has been achieved.

often ignored to avoid damaging costly equipment. Existing Model-, Simulation-, and Hardware-in-the-Loop approaches rely on a virtual environment where dangerous scenarios can be safely executed. Unfortunately, these techniques are usually limited to a specific subset of a CPPS entire functionality, and therefore are not good candidates for testing the big picture.

In this work, we present a hybrid testbed currently in use at Siemens Corporate Technology. Our testbed combines the advantages of both virtual and physical testing, by placing an entire CPPS together with its virtual counterparts into the same simulation.

Our proposed testbed solution enables test engineers to:

- Test the deployed PLC software against a simulated environment, thus safely detecting catastrophic programming issues.
- Model the test environment and link it with the behavior model. Define key test points in the environmental model, and use these points to generate variations of the test cases stimulating boundary conditions.
- Compare the real behavior and the expected behavior of the system under test in order to detect discrepancies, and refine the simulation model.
- Apply formal verification techniques on more abstract models by relying on the simulation model in order to analyze spurious counter-examples that would be otherwise hard to validate using only the physical plant.

2 Motivation and Background

The main objective for verification is to ensure the reliable operation of the Programmable Logic Controller (PLC) software within the CPPS. Modern PLC programming languages (such as SCL (Berger, 2012) and Structured Text (International Electrotechnical Commission, 2013)) can be used to provide complex control logics (in fact these are Turing complete languages). This makes PLC software verification as difficult as more traditional (e.g., C) software verification problems. Perhaps the most significant difference between verification of traditional software and verification of CPPS is that the correctness of traditional software is defined with respect to a fixed and known machine model, whereas CPPSs operate in environments that are at best partially known by the system designer, and in many cases very difficult to capture precisely. This is particularly true for systems that have a limited visibility of their surrounding environment like robots. Therefore, verifying a CPPS means verifying both the PLC software, but also verifying that the final solution is operating correctly and effectively in its targeted dynamic production environment. Moreover, the verification needs to ensure that the behavior of such a system in response to disruptions (or failures) and changes in the environment meets its stated objectives.

In these cases, it is common to verify that the system acts correctly given the knowledge that it has, avoiding the problem of modelling the real environment (Chan, Chen, Mak, & Yu, 1996). Researchers concerned with the verification of software by simulation often acknowledge the need for software testing, but typically do not present techniques beyond what is common for all types of software (Sargent, 2005), such as test-driven development or using code reviews. Others have focused on using formal specifications (W. T. Tsai, 2005), as have researchers investigating the verification of optimization software, as in compiler optimizations. However, the use of formal languages to act as an oracle can be challenging from a practical point of view, given that the specification often needs to be complete in order to be useful which cannot be assumed for CPPS. Additionally, the creation of a formal specification can be fairly complex after the software has already been developed, and requires intimate knowledge of the algorithm being implemented. Nevertheless, in the context of formal

verification, we find examples of these approaches in works that study the controller by considering a very abstract plant in which any behavior of the input/output is possible (Lange, Neuhäußer, & Noll, 2013), (Fernández Adiego, 2015). These approaches are sufficient to find bugs in the code, but cannot identify bugs in the design, since this requires reasoning by considering both the controller and the plant. From a formal methods perspective the use of hybrid modeling languages does not directly address the issue, since it is unreasonable to require the control engineer to develop a formal model describing the plant. In our opinion, this is the biggest road block hindering the adoption of formal methods within these production systems.

Model-based testing approaches (or MBT) are used to support test generation. Tests are produced based on abstract test cases from high-level system specifications written in standardized specification languages such as UML. Model-based testing in this context can be seen as a black-box test approach. The specification and execution of test cases on a model level verification and problem analysis is much easier and more efficient than it is for traditional, code-centric test cases. Model-based testing approaches automate many of the testing activities, including creation of test architectures, generation and execution of test cases. For example, Tedeso™ (former TDE/UML (Hartmann, Vieira, & Foster, 2005)) is an extensible model-based testing tool that supports different testing stages: from system specification, model analysis, test generation, and code and report generation. A distinctive feature of Tedeso™ is its extensibility and configurability by means of plug-ins. This makes it easy to integrate and extend Tedeso™ with existing tools and approaches. While MBT approaches focus on test generation strategies and coverage algorithms, they have mostly been focused on software or system testing, and need to be extended to support the challenges arising from CPPS testing. In particular, we need to describe and reason about high-dimensional environment and their dynamic behavior, define metrics for coverage criteria, and define intelligent search strategies.

3 Cyber-Physical Simulation-Based Testbed

Siemens Corporate Technology has developed a novel system testbed for cyber-physical production systems. The testbed simplifies the co-testing of PLC software for CPPS by combining simulation and physical monitoring. The simulation relies on a physics engine and serves two purposes. First, it detects disastrous failures such as collisions between production units and their environment. Second, it provides realistic data as external inputs for the system under test.

An abstracted environment is also modeled in order to capture the structure and behavior of the real-world objects under consideration. This allows us to use model-based testing approaches to generate different environmental conditions under which the CPPS is verified using simulation.

The goal of the testbed is to enable a staged approach to the verification of the PLC control software. We start by verifying the CPPS in a purely simulated environment. This allows us to flush out major issues in the control logic. Later we move to a co-verification by running the virtual and real CPPS in parallel. Detecting different behavior enables the simulation to stop the real system and prevent it from major damage. After this process, the CPPS is ready to be deployed, since most major issues have been flushed out during the preceding phases, thus reducing the possibility of disastrous failures during production.

Having an accurate simulation model of the physical plant allows us to explore the use of formal methods techniques to the verification of higher-level properties of the CPPS. We can use the simulation model to identify spurious counter-example traces generated, e.g. by model-checkers. Section 4 provides a more detailed discussion on how the testbed will allow us to use formal methods.

3.1 Testbed Architecture and Setup

The testbed architecture is based on the concept of hybrid simulation, combining physical and virtual components within the context of a distributed, embedded, real-time system to be tested. Conceptually, the setup consists of the following parts (Figure 1):

- A PLC controller running the software application under test
- A physical test setup (plant components and physical I/O interfaces)
- A virtual test setup (simulated plant components and virtual I/O interfaces)
- A plugin for a test generation framework

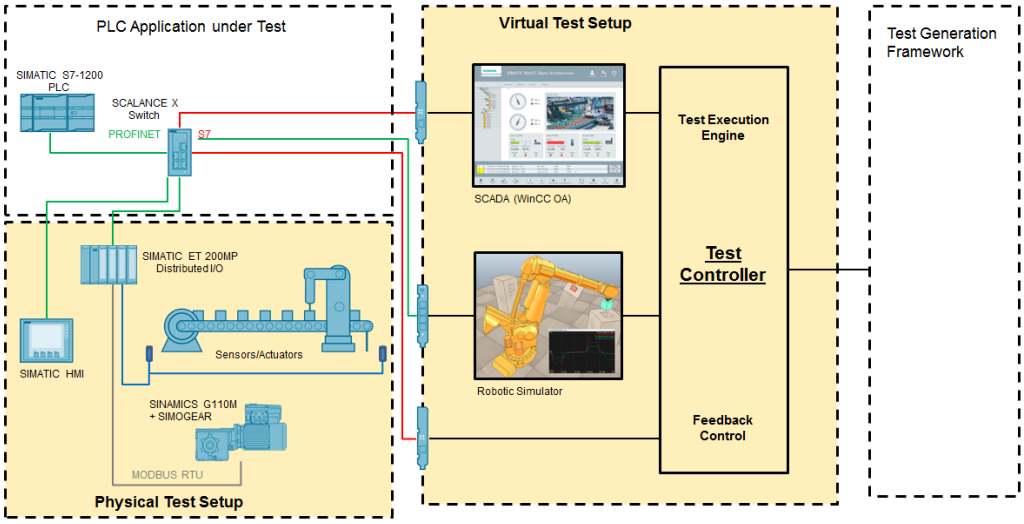

Figure 1: Testbed Architecture

It is important to note that software application under test is executed on a physical PLC controller. In principle, it would be possible to simulate this block too. However, we decided to use a physical controller for two reasons. First, different models of the controllers have slightly different capabilities and performances, making it extremely difficult to faithfully capture their behavior in a simulated environment. Second, this approach allows us to seamlessly alternate between physical and simulated environment. This reduces potential configuration errors during deployment, and makes it possible to apply the simulation after the controller has been deployed in the field. This allows us to leverage the co-verification approach in order to validate changes in existing plants.

The physical controlling element is interconnected with both the physical and the virtual testbed via network. The physical test setup includes all physical actuators, sensors and I/O interfaces. The virtual test setup implements the same plant model inside a simulator. The virtual test setup contains a test controller component, which is responsible to send (test) instructions to the physical controlling element via a virtual operating control (test execution engine). The test controller communicates with a test generation framework, in order to obtain setup data and test-cases. The test controller receives data from the physical controlling element for continuous test (setting) adaptation and comparison of results and original expectations (feedback control).

The testbed setup hereby reflects an industrial automation system, consisting of a production line and utilizing typical industrial components like actuators (e.g. conveyor belts, robots), sensors (e.g.

ultrasonic, induction) and associated controlling elements. Both physical and virtual components are interconnected with controlling elements via Industrial Ethernet, a specific Ethernet standard developed for (harsh) industrial requirements and providing protocols that allow for determinism, low latency and real-time control to deliver data under tight time constraints. Within the industrial environment, controlling elements are depicted in terms of PLCs, particular embedded control units that can act as a hard real-time system, consuming sensor input and calculating actuator output within a fixed period of time. The physical representation of the industrial automation system to be tested is based upon selected industry training models, creating a small-scale model of a physical plant. Its virtual counterparts are implemented in form of three-dimensional models inside a physics-based simulator environment. Both physical and virtual sensor data can be sent to the controlling elements and are assimilated similarly. The resulting actuator output can be visualized in real-time, either on the simulator or on the physical plant model. Physical and virtual operating control units are both based on SCADA (Supervisory Control and Data Acquisition) control architecture. Both, the physical and virtual test setup act in form of I/O devices. The reference implementation of the testbed (Figure 2) contains:

- A management environment (test engineering machine)
- A physical industry training model (e.g. fischertechnik™ industry training models, I/O device, SCADA-based Human Machine Interface (HMI))
- A virtual test environment (virtual test machine)
- An (Industrial) Ethernet switch
- A software application under test (e.g. PLC)

The management environment hosts the Windows-based Siemens IDE (Totally Integrated Automation (TIA) Portal/STEP 7) on a separate PC workstation. The IDE is required for programming and transferring the software applications under test to the physical controlling elements (PLC and I/O devices). Furthermore, an abstract network definition containing controlling and data collection elements is generated and exported using a specific interface (TIA Openness); this data is then used as input to the test generation framework.

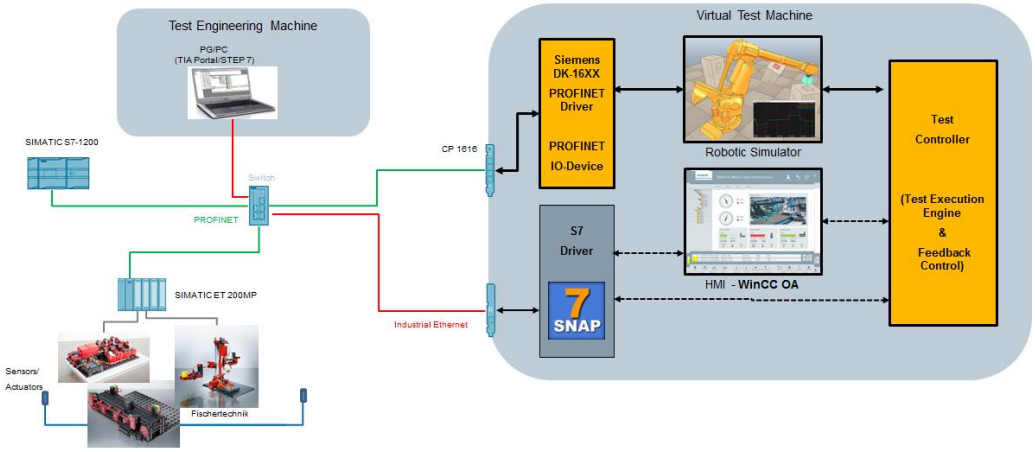

Figure 2. Testbed Realization

The choice of technologies used in the testbed is driven by a desire of being as close as possible to representative of the challenges faced by control engineers using PLCs. For this reason, we chose to use fischertechnik™ training material, real PLC hardware, and the TIA Portal IDE.

The physical industry training model consists of sensing and actuating elements typical for the field of industrial automation, but manufactured in a much smaller scale than usually deployed on traditional sites. As physical sensors and actuators in real industrial automation setups are mostly implemented in a distributed manner, within the physical test setup they are also not directly connected to the PLC itself, but rather to a distributed I/O device, which allows for remote access from a PLC via Industrial Ethernet. Both PLC and distributed I/O device belong to the Siemens SIMATIC product line (SIMATIC S7-1200, SIMATIC ET 200MP) and include several digital input/output, motor control and timer modules to connect physical sensors and actuators. The physical test setup can thus be abstracted to a form of a physical I/O device.
For connecting real-time components like PLC, distributed I/O device (physical test environment) and virtual test environment, we use the PROFINET industrial automation protocol. The virtual test environment is hosted on a Linux-based virtual test machine on a separate PC workstation, which in contrast acts as a virtual I/O device. It consists of a robotic simulator (e.g. V-Rep) (Frese, 2013), a SCADA-based HMI suite (e.g. WinCC Open Architecture), a test execution and feedback control unit and the test controller software which supports a plugin for test generation framework.

3.2 Lessons Learned

The main challenge of the testbed arises from the combination of physical and virtual components within the context of a distributed, embedded, real-time system. Thereby, (hard) real-time capable hardware components (e.g. PLC) have to be connected to a PC-based test setup, which is running software-based components that are only supporting non-real-time behavior by design. Thus, the interaction of both worlds requires a precise adjustment of the simulation step regarding the pace of individual physical control components and simulator elements.

Industrial automation test environments usually focus on the creation of a PC-based device that mimics or enhances the behavior of a PLC in software. The soft PLC therefore has to be implemented as a PROFINET IO-controller, controlling a physical system under test instead, which is connected to an I/O device. The testbed architecture depicted in Section 3.1 swaps the test objective by switching this premise and placing the PLC application itself under test. Therefore, it is important to implement the virtual test environment in form of a PROFINET IO-device, which is backed by a scenario specific simulator (e.g. robotic simulator). According to the above mentioned setup, only a simulator's API has to be adapted to the PROFINET software interface.

Despite the initial investment regarding analysis and evaluation of possible components and their interactions as well as the implementation of the testbed, the possibilities of safely and automatically testing myriad (and even hazardous) situations outweigh the effort. Combining the novel approach of feedback-based automatic creation of simulator-specific model descriptions with the proven concept of autonomous test case generation allows for efficient testing of CPPS at an unprecedented scale.

3.3 Model-based Test Generation Framework

Model-based testing (MBT) using Tedeso™ has been applied within Siemens business domains to effectively automate testing (Silva Filho & Budnik, 2012). We plug in Tedeso™ as MBT solution for test case generation. In this case the test cases are given as environmental conditions for the simulation under which the PLC code is verified (Figure 3). With Tedeso™ as our model-based testing framework, we first create an abstract model of the industrial automation system. Afterwards, an ad-hoc environment generator is used to transform the abstract model into a simulator-specific model description, which is sent to the simulator component. A code generator also translates abstract test cases into concrete test instructions, which are performed upon individual operating control units inside an execution engine.

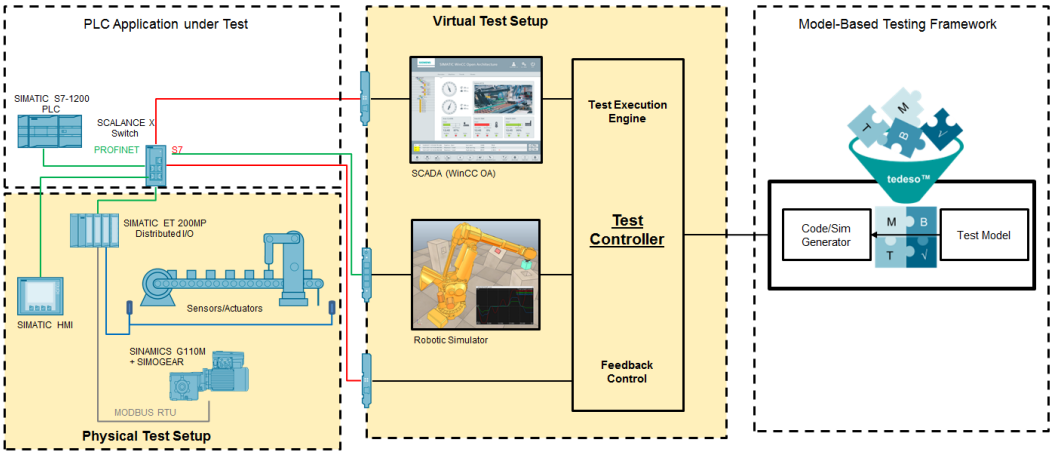

Figure 3. Supported MBT Verification by the Testbed

The environmental conditions are modeled as classes where each entity of the simulation environment can have properties such as size or max number of entities allowed and their methods of dynamic behavior implementation. Stereo-typed notes are used to define environmental constraints of and between simulation entities. For automated test environment generation the MBT model is executed with a pre-generation step that solves the environmental constraints.

The task of the test generator is to resolve the constraints leading to a set of environmental conditions which are the test cases. During test execution the physics engine is checking on the achieved goals of the system. Such feedback can be given by the physics engine for instance for collision detection of the system with its environment. The approach allows testing of the system under various conditional environments in a simulated run. Current research work is focusing on optimizing the generated test cases covering critical and exceptional scenarios from real world.

4 Integrating Formal Verification

Formal models for CPPS are difficult to express. This limits the applicability of formal verification for CPPS. Our goal is to use the simulated environment in order to simplify the formal description of the plant (to some extent of abstraction) by reusing information used to build the simulation. As a second step, we plan to use the simulation in order to increase our confidence on the correctness of the formal model.

Modern simulation frameworks (e.g., Modelica (Fritzson, 2010)) rely on libraries of components in order to speed-up the construction of the simulation. These systems feature massive libraries of components coming from multiple domains. By equipping these components with a simplified formal description, it could be possible to derive a high-level formal model of the plant automatically from the simulated model. The formal model of the plant, together with the formal model of the controller (derived automatically from the PLC code), can then be verified using theorem proving and model-checking techniques (Fulton, 2015), (Cimatti, Griggio, Mover, & Tonetta, 2015), (Tiwari, 2012). To address the environmental generation issue described in Section 3.3, we already need to provide constraints describing the formal relation between objects in the simulation. This suggests that there is a significant amount of information coming from the simulation description that can be leveraged to obtain a formal model. Even a very abstract model can be useful to prove high-level properties of the system. The typical issue in this case, however, is how to validate counter-examples and identify

spurious ones, i.e., counter-examples that do not exist in the real system but only in the abstract model. In our approach, we can rely on the simulation model to validate spurious counter-examples and refine the formal representation accordingly. Without a simulated model, it would be much more difficult (if not impossible), expensive and time-consuming to validate the counter-example on the physical plant. Our second step is to use the simulation to validate the formal model. The idea is to use the formal model to generate runtime monitors (Mitsch & Platzer, 2014). These monitors are attached to the simulated environment and run for a massive amount of scenarios. This process will allow us to validate the formal model, and this would not be possible if we had to execute these monitors directly on the physical plant. Since the simulation model is aligned to the physical system, and the formal model is aligned with the simulation model, we have a strong confidence on our formal verification results. This allows us to gradually apply formal verification techniques, without requiring a significant up-front investment in modeling. In particular, we can dedicate resources to formally verify systems or components that are about to change, thus providing a strong business argument for the application of more rigorous techniques.

Finally, our approach to model-based testing relies on a black-box view of the system. Once a formal model of the CPPS is defined, it becomes easier to generate test-cases that lead to particular configurations, by using model-checking techniques. Many questions still need to be addressed in order to effectively combine formal verification and simulation techniques. Nevertheless, having a realistic testbed will allow us to better characterize the problems, and create interesting benchmarks to drive the development of tools.

References

Berger, H. (2012). *Automating with STEP 7 in STL and SCL: SIMATIC S7-300/400 Programmable Controllers.* Wiley.

Chan, F., Chen, T., Mak, I., & Yu, Y. (1996). Proportional sampling strategy: guidelines for software testing practitioners. *Inf. Softw. Technol. 38 (12)*, 775–782.

Cimatti, A., Griggio, A., Mover, S., & Tonetta, S. (2015). HyComp: An SMT-based model checker for hybrid systems. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Heidelberg: Springer Berlin.

Fernández Adiego, B. e. (2015). Applying model checking to industrial-sized PLC programs. *IEEE Transactions on Industrial Informatics*, 1400-1410.

Frese, E. R. (2013). V-REP: a Versatile and Scalable Robot Simulation Framework. *International Conference on Intelligent Robots and Systems (IROS).*

Fritzson, P. (2010). *Principles of object-oriented modeling and simulation with Modelica 2.1.* John Wiley & Sons.

Fulton, N. e. (2015). KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. *International Conference on Automated Deduction.* Springer International Publishing.

Hartmann, J., Vieira, M., & Foster, H. (2005). A UML-based approach to system testing. *Innovations in Systems and Software Engineering*, 12-24.

International Electrotechnical Commission. (2013). *IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages.*

Kephart, J., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 41-50.

Lange, T., Neuhäußer, M., & Noll, T. (2013). Speeding Up the Safety Verification of Programmable Logic Controller Code. *Haifa Verification Conference*, (pp. 44-60).

Mitsch, S., & Platzer, A. (2014). *Patent No. U.S. Patent Application No. 15/026,690.* U.S.

Tiwari, A. (2012). HybridSAL relational abstracter. *International Conference on Computer Aided Verification.* Heidelberg: Springer Berlin.