A Hybrid Testbed for 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. This work presents a hybrid testbed combining the monitoring of the real physical CPPS components together with its virtual counterparts in simulation.

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 (J.O. Kephart and D. M. Chess, 2003). Thus, the CPPS need to be able to effectively and efficiently fulfill its specified goals under changing conditions.

CPPS intrinsically combine hardware and physical systems with software and networking. 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 than homogeneous systems in addition to their uncertain behavior adapting to their environment.

The operation of field tests for CCPS is usually an expensive, time and resource consuming effort. However, an entire production unit can be tested under realistic conditions and particularly in cooperation with additional production units. But, thereby only a limited set of potential scenarios can be realized, mostly ignoring possibly hazardous situations. Existing Model-, Simulation-, and Hardware-in-the-Loop approaches and virtual test drive setups instead are executed in a rather static (virtual) environment, providing the ability for safely testing of even dangerous scenarios.

[*] Majority of his work has been achieved during the internship with Siemens Corporate Technology.

Unfortunately, they are usually limited to a specific subset of a CPPS entire functionality, and therefore lack in opportunities for testing the big picture. Therefore, this contribution of a testbed, combining the advantages of both virtuality and reality, by placing an entire CPPS together with its virtual counterparts into the same simulation, sounds very promising. Our proposed testbed solution will enable test engineers to:

- Model the test environment and link it with the behavior model. The environmental model will be used to generate variations of environmental changes. Key test points shall be defined in the environmental model which affects the system behavior.
- Feed the environmental changes as inputs to the system by devised control mechanisms in a simulation environment to verify its behavior. Testing can be done completely in the simulation or with the actual system control hardware in the loop.
- Monitor the behavior and perceived environment of the system under test in order to dynamically direct the system under test towards generated environmental conditions to verify its correctness.

2 Motivation and Background

The main objective for verification is to ensure the reliable operation of the Programmable Logic Controller (PLC) software in Structured Control Language (SCL) (Berger, 2012) that controls the CPPS and provide confidence that the compound solution, as a collective result of collaborating production units, is operating correctly and effectively in its targeted dynamic production environment. Thus, the verification is to ensure that the behavior of such a system in response to disruptions (or failures) and changes in the environment meets its stated objectives.

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 especially robots and other CPPS operate in environments that are at best partially known by the system designer. In these cases, it is practical to verify that the system acts correctly given the knowledge that it has, avoiding the problem of modelling the real environment (F.T. Chan, 1996).

Siemens Corporate Technologies has developed a novel system testbed for cyber-physical systems that is utilizing simulation. The simulation is used as an external input to the system under test representing its environment. The environment is modeled by the structure and behavior of its real world objects under consideration and their relationship to each other defined by constraints. A model-based testing approach is generating different environmental conditions under which the CPPS is verified using simulation.

2.1 Model-based Testing

Model-based testing approaches (or MBT) have been used in support of 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 (J. Hartmann, 2005)) is an extensible model-based testing tool that supports different testing stages: from system specification, model checking, test

generation, and code and report generation. A distinctive feature of Tedeso™ is its support for extensibility and configurability of its features by means of plug-ins. Through the use of these mechanisms, Tedeso™ can be integrated with existing tools and approaches.

While MBT approaches focus on test generation strategies and coverage algorithms, they do not address the generation of environmental configurations.

2.2 Simulation based Testing

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.

3 Cyber-physical Simulation-based Testbed

We present a testbed that is built out of components that are also used by Siemens to train PLC experts. The testbed simplifies the co-testing of PLC software for CPPS by combining simulation and physical monitoring. The simulation is using a physics engine which main objective is to detect disastrous failures such as collisions between production units and their environment.

The testbed enables a staged approach to verify the PLC control software first in a virtual environment, having then the real and simulated environment run in parallel in order to compare their behavior. Detecting different behavior enables the simulation to stop the real system and prevent it from major damage. In the last stage, the system would be run in the real world only. However, the previous stages shall have significantly reduced the risk in finding disastrous failures during the last stage.

3.1 Testbed Architecture and Setup

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

- A software application under test
- A physical test setup (plant model and physical operating control)
- A virtual test setup (plant model in simulator and virtual operating control)
- A plugin for a test generation framework

The software application under test is executed on a physical controlling element. The physical controlling element is interconnected with both the physical and the virtual testbed via network. The physical test setup is made up of a physical plant model and can be manipulated by a physical operating control unit. Within its virtual counterpart, the same plant model is implemented inside a simulator. Besides this, the virtual test setup also hosts a test controller component, which is responsible to send (test) instructions to the physical controlling element via a virtual operating

control (test execution engine). It further exchanges required setup data between the test generation framework and the simulator and receives data from the physical controlling element for continuous test (setting) adaptation and comparison of results and original expectations (feedback control).

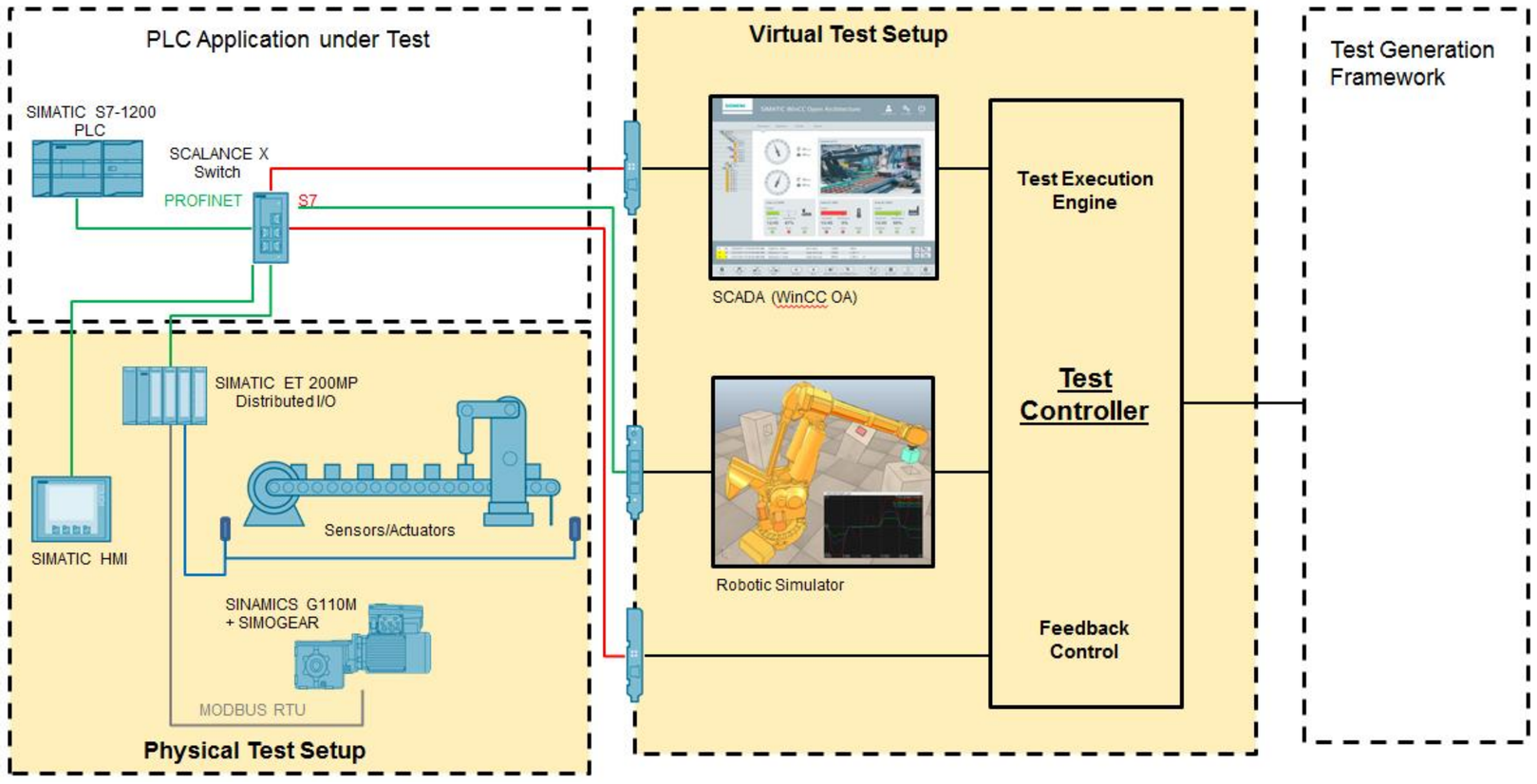

Figure 1. Testbed Architecture

The testbed setup hereby reflects an industrial automation system, exemplary 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 SCADA-based. Both, the physical and virtual test setup act in form of I/O devices.

The according reference implementation realizing the testbed setup (see Figure 2) is deployed by the following components:

- 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 is hosting the Windows-based Siemens IDE (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, e.g. to PLC(s) and I/O device(s). Furthermore, an abstract network definition containing controlling and data collection elements is generated. Via a specific interface (TIA Openness), relevant data concerning the network and device setup can be exported as input to the test generation framework.

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. The physical test setup consists of industrial training models from fischertechnik GmbH. 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 are descended exemplarily from the Siemens SIMATIC product line (SIMATIC S7-1200, SIMATIC ET 200MP) and backed by several digital input/output, motor control and timer modules to connect physical sensors and actuators. The physical test setup as a whole can thus be abstracted in form of a physical I/O device.

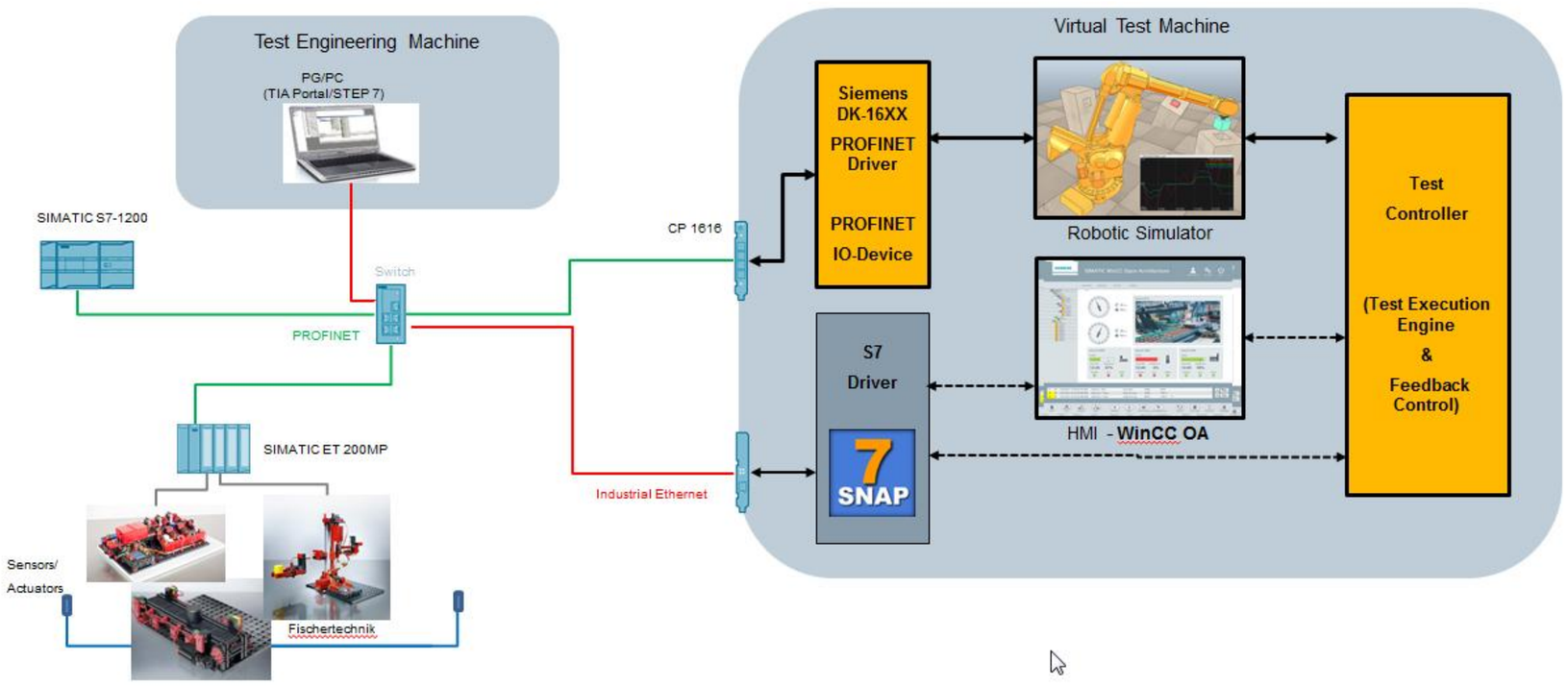

Figure 2. Testbed Realization Setup

As exemplary Industrial Ethernet standard for connecting real-time components like PLC, distributed I/O device (physical test environment) and virtual test environment, the PROFINET automation protocol is utilized.

The virtual test environment is used for hosting 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) (Freese, 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. Section 0 will describe a sample test generation framework that can be used as solution.

3.2 Lessons Learned and Conclusion

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. With Tedeso™ as our model-based testing framework, at first an abstract model of the industrial automation system can be created. A specific environment generator is then used to transform the abstract model to 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.

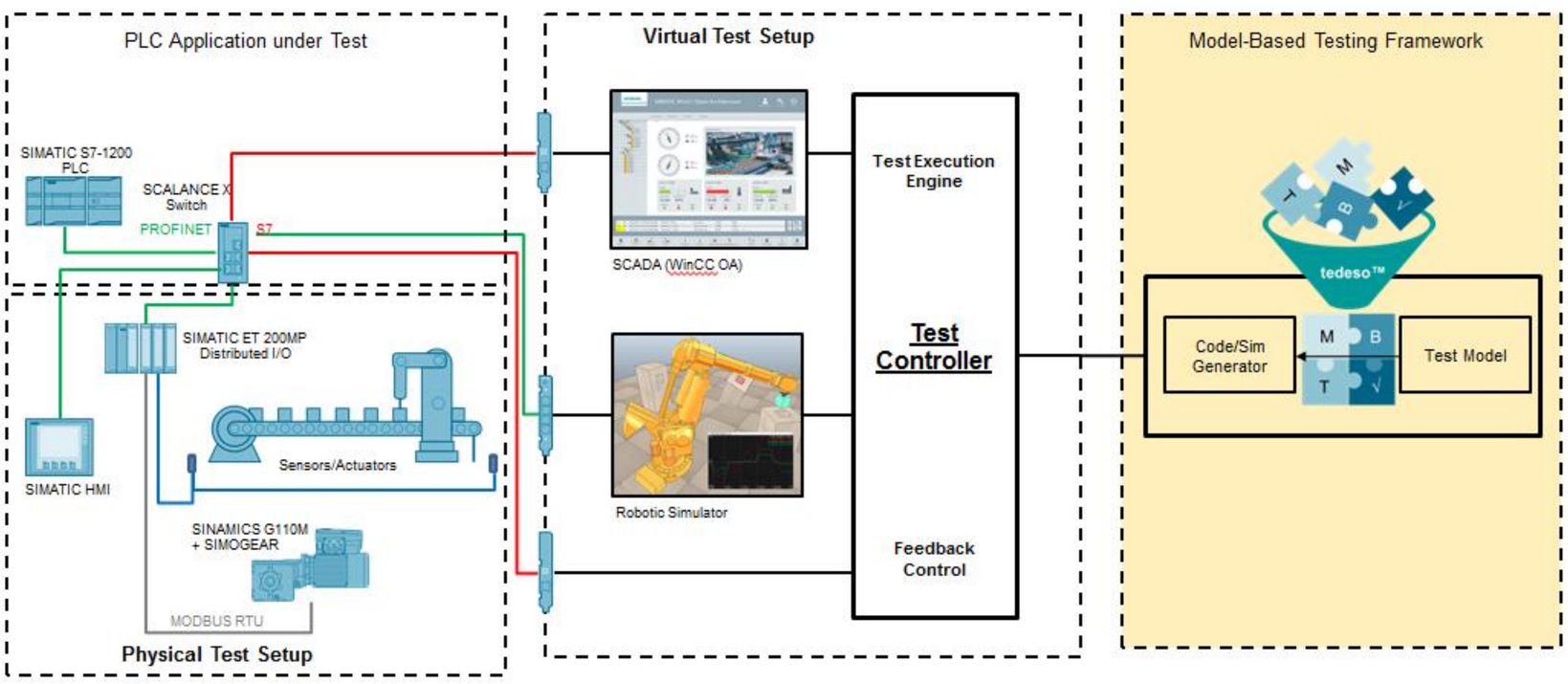

Figure 3. Supported MBT Verification by the Testbed

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 environmental MBT model is run through the test generation process, 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 Outlook on the use of Formal Verification

Understand how to better combine formal verification and simulation is one of the directions that we aim at exploring in the future. The controller interacts with the plant through the use of sensors and actuators. A first level of formal verification techniques can be applied (Tim Lange, 2013), (B. Fernández Adiego, 2015) to study the controller by considering a very abstract plant in which any behavior of the input/output is possible. These approaches are sufficient to find bugs in the code, but cannot be used to identify bugs in the design, since this requires reasoning by considering both the controller and the plant. Work in this area has focused on the concept of Hybrid systems verification. However, one of the challenges in performing these types of verification, comes from the difficulty of specifying the formal model describing the plant. In our opinion, this is the biggest road-block hindering the adoption of formal methods within these production systems.

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, 2015), (Tiwari, 2012). Depending on the level of abstraction of the model, we might run into spurious counter-examples, i.e., counter-examples that do not exist in the real system but only in the abstract model. Spurious counter-examples can then be validated against the simulation model and, when shown spurious, the formal representation can be refined to exclude them.

Our second step is to use the simulation to validate the formal model, and validating counter-examples is a simple example of how this can be beneficial for the formal side. More in general, from the formal model of the plant we can build runtime monitors (S Mitsch, A Platzer, 2016), that can be executed in the simulated environment for a massive amount of scenarios. This process will allow us to validate the formal model, and would not be possible if we had to execute these monitors directly on the physical plant. The process becomes significantly more useful if the simulation model has been fine-tuned and synchronized against the physical plant. A typical example of this would be changes to the code of the controller after the system has been deployed. These changes can be formally verified against a formal model that has been extensively validated through simulation. The simulation, in turn, would have been validated against the system through hours of execution.

Many questions 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

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

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

Cimatti, A. e. (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.

CP1616 & DK-16x. (n.d.). Retrieved from https://support.industry.siemens.com/cs/document/21626318/cp-1616-and-dk-16xx-pn-io-development-kit-released-for-delivery?dti=0&lc=en-WW

F.T. Chan, T. C. (1996). Proportional sampling strategy: guidelines for software testing practitioners. *Inf. Softw. Technol. 38 (12)*, 775–782.

Freese, E. R. (2013). V-REP: a Versatile and Scalable Robot Simulation Framework. *Proc. of The 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.

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

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

PROFINET System Description. (2009) PROFIBUS Nutzerorganisation e.V.

S Mitsch, A Platzer. (2016). Verified runtime validation of verified cyber-physical system models. *Verified runtime validation of verified cyber-physical system models.*

S. R. Dalal, A. J. (1999). Model-based testing in practice. *Proceedings of the 21st international conference on Software engineering.*

Sargent, R. G. (2005). Verification and validation of simulation models. *In Proc. of the 37th conference on winter simulation*, 130–143.

Silva Filho, R., & Budnik, C. (2012). An Integrated Model-Driven Approach for Mechatronic Systems Testing. *IEEE International Conference on Software Testing, Verification and Validation*, (pp. 447-457).

Snap 7. (n.d.). Retrieved from http://snap7.sourceforge.net/

TIA Openness. (n.d.). Retrieved from https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness%3A-introduction-and-demo-application?dti=0&lc=en-WW

TIA Portal/STEP 7. (n.d.). Retrieved from https://www.industry.siemens.com/topics/global/en/tia-portal/Pages/default.aspx

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

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

W. T. Tsai, X. L. (2005). Simulation verification and validation by dynamic policy enforcement. *In Proc. of the 38th annual Symposium on Simulation*, 91–98.