CyPhEF: A Model-Driven Engineering Framework for Self-Adaptive Cyber-Physical Systems

Mirko D'Angelo
Linnaeus University
Växjö, Sweden
mirko.dangelo@lnu.se

Annalisa Napolitano
IMT School for Advanced Studies
Lucca, Italy
annalisa.napolitano@imtlucca.it

Mauro Caporuscio
Linnaeus University
Växjö, Sweden
mauro.caporuscio@lnu.se

ABSTRACT

Self-adaptation is nowadays recognized as an effective approach to deal with the uncertainty inherent to cyber-physical systems, which are composed of dynamic and deeply intertwined physical and software components interacting with each other. Engineering a self-adaptive cyber-physical system is challenging, as concerns about both the physical and the control system should be jointly considered. To this end, we present CyPhEF, a Model-Driven Engineering framework supporting the development and validation of self-adaptive cyber-physical systems.
Demo video: https://youtu.be/nmg-w2kfKEA.

ACM Reference Format:
Mirko D'Angelo, Annalisa Napolitano, and Mauro Caporuscio. 2018. CyPhEF: A Model-Driven Engineering Framework for Self-Adaptive Cyber-Physical Systems. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183483

1 INTRODUCTION

Cyber-Physical Systems (CPS) are realized as a dynamic composition of autonomous and heterogeneous components interacting with each other. Since CPS operate under highly dynamic conditions, the traditional stability assumptions made on systems' design are no longer valid. The dynamics introduce *uncertainty*, which in turn may harm the system and lead to incomplete, inaccurate, and unreliable results [12]. Managing the run-time uncertainty is then crucial for the dependability of CPS.

To this end, self-adaptation is widely considered as an effective approach to deal with the uncertainty and dynamic of the environment [4]. Self-adaptive systems typically consist of a *managed* subsystem that implements the system functionality and a *managing* subsystem that implements the adaptation logic by means of the MAPE-K loop model, with Monitor, Analyze, Plan and Execute components, complemented with a Knowledge that maintains relevant information about the managed subsystem [16]. When dealing with a large-scale distributed system, a centralized control is hardly adequate to manage the whole system. Hence, self-adaptation is achieved through decentralized control, where multiple loops interact with each other to address critical run-time conditions.

Developing self-adaptive CPS might result challenging, as the *adaptation dimensions* should be considered at the control level, as well as at the physical level. Indeed, developers are required to concern with cross-layer adaptation aspects, including [9]: (*i*) Why there is the need to change? (*ii*) What does (not) change? (*iii*) How does the change take place? (*iv*) Who does the adaptation? and (*v*) Where does the change take place? Therefore, structural, functional and non-functional properties of both the *managing* and the *managed* subsystems should be taken into account during all phases of the self-adaptive CPS life-cycle, from design to validation [4].

To this end, simulation is a common approach for efficiently and timely analyzing how a given self-adaptive CPS behaves with respect to all these dimensions. Indeed, simulation is less time and resource consuming as compared to exhaustive verification approaches, while still providing accurate results.

This paper presents CyPhEF[1], a CYber-PHysical dEvelopment Framework that leverages the Model-Driven Engineering (MDE) paradigm [5], and refers to the systematic use of models as primary artifacts for engineering self-adaptive CPS. In particular, CyPhEF promotes MAPE-K components [16] to first-class modeling abstractions and provides: (*i*) a Domain-Specific Environment (DSE) for specifying the desired MAPE-K control loop model, and (*ii*) a Cyber-Physical Simulation Platform for simulating the designed self-adaptive CPS as a whole, and producing simulation results.

Several research have been conducted on self-adaptive CPS. Arcaini et al. [1] propose a framework to describe complex MAPE-K loops according to patterns, and exploit validation and verification techniques for assuring correctness of components' interactions. Differently from our model-driven engineering approach, the work makes use of multiagent abstract state machine to specify self-adaptive systems. Authors in [15] envision a general framework for software systems to self-adapt with running environmental dynamics and fulfill various user requirements. In this work, the overall control architecture is implemented as a centralized control loop and the developer does not have the ability to design and develop his own control schemes. Finally, Krupitzer et al. [10] propose a framework for designing self-adaptive systems. Their system model offers an initialization of the system based on elements of a component library and the adaptation logic is based on MAPE-K patterns. Despite of the similarities with our work, the supporting tool is not implemented, but planned as future work.

This paper is organized as follows. Section 2 presents the model-driven engineering approach underpinning CyPhEF, and the main provided features. Section 3 shows how to use CyPhEF in practice. Finally, Section 4 draws conclusions and hints for future work.

[1] *https://mi-da.github.io/cyphef/*

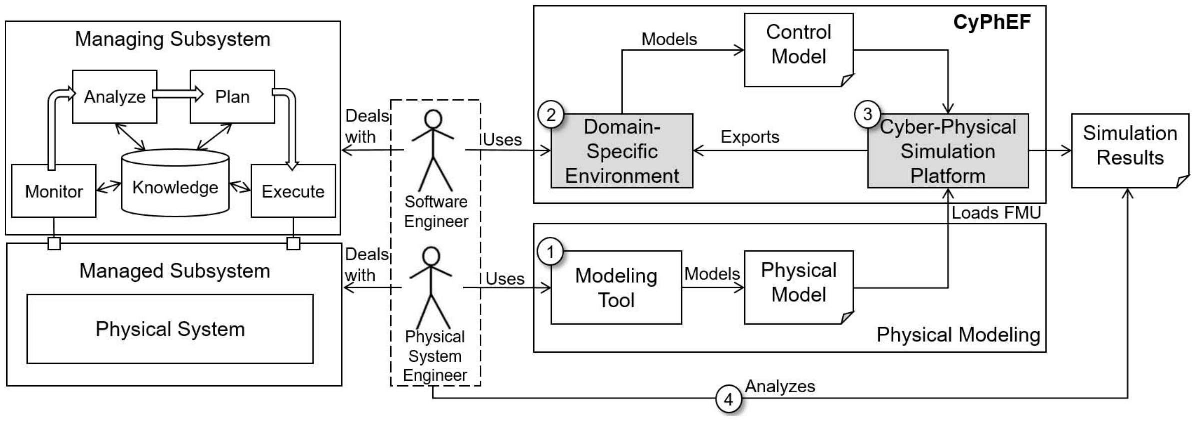

Figure 1: The Model-Driven Engineering approach

2 TOOL FEATURES

The premise of self-adaptive CPS development should be a multi-disciplinary approach that takes into account physical and control factors. Referring to Figure 1, on the one hand, the Physical System engineer deals with the managed subsystem that provides the system's domain functionality. On the other hand, the Software Engineer should deal with the managing subsystem that realizes the adaptation logic by implementing the MAPE-K control loop.

In order to master the complexity of such a class of systems, our approach adheres to MDE paradigm [5] and makes use of models to engineer self-adaptive CPS. Models are considered primary artifacts in all engineering activities and are used to analyze how things behave within specific areas of interest (e.g., mechanics, physics, and computing). Indeed, we exploit models at all stages of the development process, from design to validation. As highlighted in Figure 1, the core features of CyPhEF are the Domain-Specific Environment, and the Cyber-Physical Simulation Platform (see shaded boxes), which offer functionalities to provide engineers with a comprehensive support tool for designing and validating self-adaptive CPS, respectively.

In particular: ① the Physical System Engineer designs the physical model (e.g., a Modelica model [7]) and exports it as Functional Mock-up Unit (FMU), a standard for model representation [2]; ② the Software Engineer loads the FMU in the tool, where the properties of the physical model are automatically exported to the Domain-Specific Environment, and instantiates the Control Model for the Physical Model; ③ the Software Engineer simulates through the Cyber-Physical Simulation Platform the two models; and ④ the Cyber-Physical Simulation Platform produces results, which in turn are analyzed to investigate the behavior of the self-adaptive CPS under development. In particular, performance measures and quality attributes concerning both the physical system metrics (e.g., voltage level) and the network level metrics (e.g., number of exchanged messages, number of packets lost) can be investigated.

2.1 Domain-Specific Environment

DSE (see Figure 2) provides a graphical tool for defining the MAPE-K architecture for the cyber-physical system under control. The DSE is implemented as an Eclipse[2] plugin, and is based on the Eclipse Modeling Framework[3], which provides proper mechanisms for developing model-driven development tools.

DSE allows Software Engineers to specify a Control Model as a set of components, each implementing a specific MAPE-K activity. Control Models comply with a general meta-model providing a concrete perspective on the operational and distribution aspects concerning the MAPE-K components [6]. Constraints on the meta-model are defined in Object Constraint Language (OCL), a declarative language for describing rules applying to UML models. Further, such a general meta-model can be easily extended to define customized control patterns. In particular, DSE offers a catalog of ready-to-use control patterns [16], which extend the general meta-model, and define the specific role of involved MAPE-K components and interactions via OCL rules. OCL rules are then used to validate whether a concrete control loop is compliant or not with the given control pattern. That is, DSE allows the user to choose the specific control pattern to adopt, and to map MAPE-K components to the entities belonging to the physical system under control. Then, DSE validates the concrete architecture against the defined meta-model.

2.2 Cyber-Physical Simulation Platform

The Cyber-Physical Simulation Platform, see Figure 3, allows for co-simulating the *Physical Model* and the *Control Model*, designed in the physical modeling tool and in DSE respectively. Simulation is performed by a *Co-Simulation Engine*, which simultaneously executes the two models by allowing information to be shared among them. Simulation provides practical feedback about the system under development, and allows engineers to assess a design choice before the system is actually put in operation. Moreover, simulation can be used to evaluate different types of dependability concerns, e.g., reliability, performance, and adaptation time. Hence, the software engineer can assess, and possibly improve, the dependability of a self-adaptive CPS by simulating different solutions and/or configurations.

[2]http://www.eclipse.org/
[3]http://www.eclipse.org/modeling/emf/

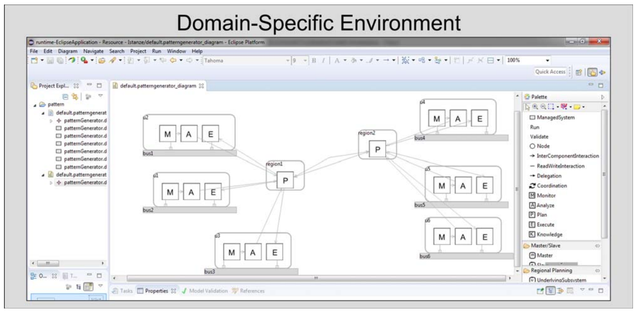

Figure 2: The Domain-Specific Environment

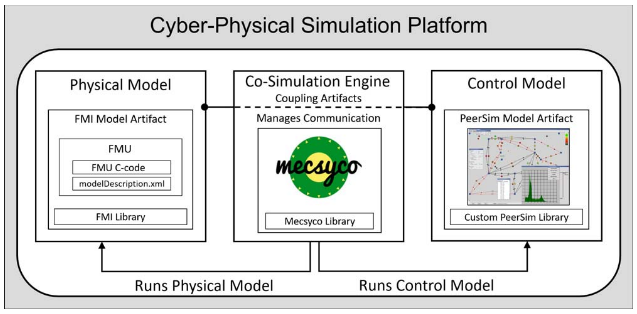

Figure 3: The Cyber-Physical Simulation Platform

The Cyber-Physical Simulation Platform is able to manage any generic Physical Model exported by external modeling tools (e.g., OpenModelica) as FMU (Functional Mock-up Unit) via FMI (Functional Mock-up Interface) [2], a standard for dynamic models co-simulation. The execution of the FMUs is delegated to the Co-Simulation Engine. The Control Model simulation relies on PeerSim, a scalable peer-to-peer network simulator [11], and specifies the designed control loop as a set of networked entities. In order to be integrated within the Co-Simulation Engine, PeerSim has been re-engineered and extended by defining a specific API that allows for controlling the network simulation.

The Control and Physical Models are then co-simulated by leveraging on MECSYCO [3], a co-simulation engine that enables to handle the models by providing synchronization and inter-models communication. MECSYCO adheres to the agents and artifacts paradigm [14] and describes and simulates complex systems as multi-agent systems. In particular, agents are autonomous and represents the different simulators in the Co-Simulation Engine. The artifacts are the reactive elements of the environment allowing the agents to communicate with each other. We integrated the control and physical subsystems by developing the respective models and making them interact via coupling artifacts. Specifically, for each network entity that needs to observe or interact with a physical entity, we establish a coupling artifact that allows the models to communicate each other.

Finally, CyPhEF defines an abstract and platform agnostic API to (*i*) export the details of the physical subsystem under management to DSE, (*ii*) perform model-to-code transformations of the control loop designed and validated by DSE, and (*iii*) deploy the control loop into the specific system. It is worth noticing that, as the API definition is generic and abstract, it can be easily customized and instantiated to different technologies[1]. This makes CyPhEF a modular and extensible tool.

CyPhEF currently instantiates such an API to include FMU and PeerSim: first, physical subsystem details are exposed to DSE as a list of managed subsystems, which are elicited from the set of physical entities inspected in FMUs; second, once the Control Model is designed, model-to-code transformation is performed to translate modeled entities and interactions into the network simulator, according to the PeerSim semantics.

3 IN ACTION

CyPhEF has been developed as part of a larger initiative on engineering self-adaptive systems funded by the Swedish Knowledge Foundation, Project *Software Technology for Self-Adaptive Systems – Model-Driven Engineering Applied to Smart Power Grids*.

CyPhEF has been used to design and validate two different Smart Power Grid (SPG) real case studies in the context of the project. Specifically, SPG has been designed as a self-adaptive CPS that integrates the electrical power grid (i.e., managed subsystem) with information and communication infrastructures (i.e., managing subsystem) to produce and manage stable and sustainable electric energy. The SPG's core mission is to provide a secure and sustainable electricity supply and quality of service in dynamic conditions.

The first system that we tested, is an extension of a real transmission grid, namely the IEEE 9 bus system, representing an approximation of the Western System Coordinating Council [6]. The second SPG that we tested, is the modified Consortium for Electric Reliability Technology Solutions (CERTS) microgrid [13]. Such SPG can be configured with an independent power supply, in connection with the utility, in a cluster, or even in P2P mode.

Figure 4 sketches how a user interacts with CyPhEF: ① the user designs the physical model with a specific modeling tool (e.g., OpenModelica) and exports the model as an FMU; ② the user imports the FMU into CyPhEF, designs the Control Model and maps it to the Physical Model; ③ the two models are co-simulated and the observed properties of interest (i.e., the behavior of the physical model) are shown in real-time during the simulation; finally, ④ the user analyzes the simulation results collected by CyPhEF and derives information about the complex CPS.

Thanks to the flexibility of CyPhEF, we have been able to easily and timely test different control patterns (e.g., master/slave, regional planning, information sharing and coordinated control), and verify the SPG self-adaptive behavior in case of arbitrary faults in the distribution network. In particular, we measured the *adaptation time* – i.e., the time between the detection of a fault inside the grid and the enactment of a reconfiguration plan – as it represents a key concern when dealing with safety-critical systems such as an SPG. Besides *adaptation time*, in both scenarios we obtained interesting simulation results concerning both the physical system metrics (e.g., voltage level, energy consumption) and the network level metrics (e.g., number of exchanged messages, number of packets lost). Moreover, we had the opportunity to start testing the performance of the

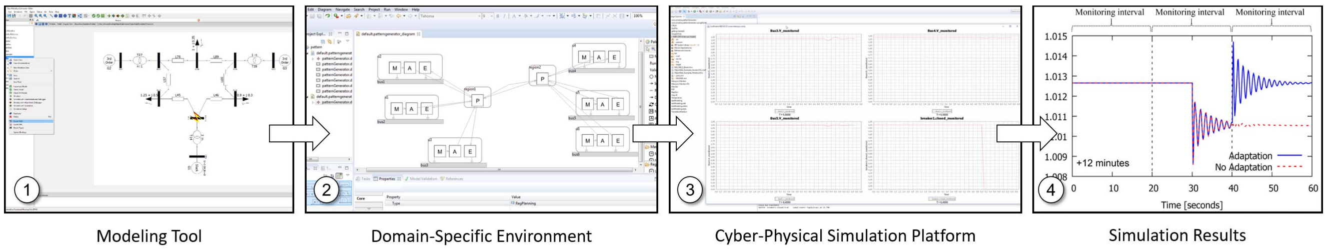

Figure 4: Tool Usage

tool itself and evaluating its usability. In all cases, the time taken to simulate and validate the different SPG scenarios was in the order of seconds. The obtained results show that engineers can benefit from using CyPhEF while dealing with complex self-adaptive CPS.

4 CONCLUSIONS AND FUTURE WORK

Developing self-adaptive CPS is challenging, as adaptation dimensions should be considered at both the control and the physical level. Since in large distributed settings centralized control is hardly adequate, self-adaptation might be achieved through decentralized control. This drives towards the need of defining a development process, where all the constituents of both the control and the physical systems are made explicit, and then jointly considered while designing, developing, and validating self-adaptive CPS.

To this end, we presented CyPhEF, a CYber-PHysical dEvelopment Framework that leverages the MDE paradigm. In particular, the core features of CyPhEF are the Domain-Specific Environment and the Cyber-Physical Simulation Platform. The former allows for defining the control architecture of the cyber-physical system under management, whereas the latter allows for simulating the designed self-adaptive CPS as a whole and producing simulation results. CyPhEF has been developed and used in the context of the research project *Software Technology for Self-Adaptive Systems – Model-Driven Engineering Applied to Smart Power Grids*, for designing and validating two different Smart Power Grid case studies.

Ongoing and future work proceeds toward different lines of research. First, we aim at finalizing the development of CyPhEF and improving the DSE. In particular, we envision the development/adoption of a validation technique able to test and analyze the control behavior and automatically detect unintended interactions. Further, we aim at providing developers with the ability of formally specifying the behavior of the MAPE-K components and automatically deploying them [8].

Second, CyPhEF currently allows for the co-simulation of a single physical model and a single control model (either centralized or decentralized). Future work is to improve the tool by allowing for the co-simulation of multiple entities, each one defined as a CPS and characterized by its own control and physical model. Such a modularization would greatly improve the architecture of CyPhEF and, make it possible to dynamically plug-and-play components at simulation time. Moreover, a deeper investigation of the tool's performance is needed. To this end, we aim at evaluating the scalability of the tool with respect to the size of the managed models. In addition, we plan to extend CyPhEF in order to deal with open systems, where the number of managed entities is not known in advance. Indeed, entities can dynamically enter/leave the execution environment at run time, as for example in Intelligent Transport System (ITS) scenarios. To this end, we are currently researching on how to integrate a traffic simulator [17] into CyPhEF.

Finally, we aim at investigating how to evolve/adapt the defined control architecture itself. In fact, run-time adaptation of the control architecture might be necessary due to changes in the system resources, the environment, or the system goals.

ACKNOWLEDGMENT

This research has received funding from the Swedish Knowledge Foundation, Grant No. 20150088: *Software Technology for Self-Adaptive Systems – Model-Driven Engineering Applied to Smart Power Grids.*

REFERENCES

[1] P. Arcaini, E. Riccobene, and P. Scandurra. Formal design and verification of self-adaptive systems with decentralized control. *ACM Trans. Auton. Adapt. Syst.*, 11(4):25:1–25:35, Jan. 2017.
[2] T. Blochwitz et al. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International Modelica Conference*, pages 173–184. The Modelica Association, 2012.
[3] B. Camus et al. Hybrid co-simulation of FMUs using DEV&DESS in MECSYCO. In *Proc. of the Symposium on Theory of Modeling &Simulation.*
[4] B. Cheng et al. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap. 2009.
[5] A. R. da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139 – 155, 2015.
[6] M. D'Angelo, M. Caporuscio, and A. Napolitano. Model-driven engineering of decentralized control in cyber-physical systems. In *Proc. of the 2nd International Workshops on Foundations and Applications of Self* Systems*, 2017.
[7] P. Fritzson and P. Bunus. Modelica – A General Object-Oriented Language for Continuous and Discrete-Event System Modeling. In *Proc. of the 35Th Annual Simulation Symposium*, 2002.
[8] M. U. Iftikhar and D. Weyns. Activforms: Active formal models for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2014.
[9] P. Inverardi and M. Tivoli. Software engineering. chapter The Future of Software: Adaptation and Dependability, pages 1–31. Springer-Verlag, 2009.
[10] C. Krupitzer, S. Vansyckel, and C. Becker. FESAS: Towards a framework for engineering self-adaptive systems. In *Proc. of the 7th International Conference on Self-Adaptive and Self-Organizing Systems*, 2013.
[11] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Sept. 2009.
[12] D. Perez-Palacin and R. Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation. In *Proc. of the 5th International Conference on Performance Engineering*, ICPE '14, 2014.
[13] F. Shariatzadeh, R. Zamora, and A. K. Srivastava. Real time implementation of microgrid reconfiguration. In *North American Power Symposium*, 2011.
[14] Y. Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, Mar. 1993.
[15] L. Wang, Y. Gao, C. Cao, and L. Wang. Towards a general supporting framework for self-adaptive software systems. In *Proc. of 36th Annual Computer Software and Applications Conference Workshops*, 2012.
[16] D. Weyns et al. On patterns for decentralized control in self-adaptive systems. *Lecture Notes in Computer Science*, 7475 LNCS:76–107, 2013.
[17] M. Treiber and A. Kesting. An open-source microscopic traffic simulator. *IEEE Intelligent Transportation Systems Magazine*, 2010.