DIGITAL TWINS FOR CONTINUOUS DEPLOYMENT IN MODEL-BASED SYSTEMS ENGINEERING OF CYBER-PHYSICAL SYSTEMS

Joost Mertens

joost.mertens@uantwerpen.be

ABSTRACT

Cyber-Physical Systems (CPS) are engineered systems that combine the cyber (control, software, networking) and physical (electronics, mechatronics) domains into one system. Historically, such systems followed a deliver and maintain lifecycle, but nowadays, they are expected to operate over a longer time. Rather than decommissioning them, they are upgraded to meet new requirements. Because hardware upgrades are expensive, there is a preference to implement new functionality in software whenever possible. In traditional software engineering, we find methodologies such as DevOps (Development and Operations), in which a piece of software is developed incrementally and continuously deployed. Carrying over these techniques to CPS has two caveats. Firstly, CPS are generally developed in a Model-Based Engineering way; the most appropriate models in the most appropriate formalism are used in the development process to capture various aspects, e.g. architecture, behavior, and physical design. Secondly, Cyber-Physical Systems undergo wear, tear and physical aging during their operation. Over time, a discrepancy between the design time models and the physical system may appear. These two caveats yield a problem during the testing phase of the new piece of software: techniques such as co-simulation, which are utilized to test a controllers' behavior on physics models, rely on accurate system models to function. Thus, we are faced with a challenge: our system models may become unrepresentative of the real-world system, in which case it becomes nonsensical or even dangerous to use in further development and testing. We might incorrectly conclude that a new piece of software will work flawlessly, while erroneous behaviour appears once deployed. Further complicating the matter is that CPS are commonly deployed in fleets where every instance lives its own life, in its own specific conditions. Therefore, tracking a discrepancy between the actual system and the design time models must be done at the instance level.

In industry, we see the adoption of the digital twin, which allegedly alleviates this issue. A digital twin is a virtual representation of an actual system. The actual system continuously feeds the digital twin with actual data, such that it mirrors the actual system. The digital twin offers many possibilities: replaying historic traces, inspecting the current system state, and predicting the future states. For the problems related to software deployment, we use the digital twin to test new software releases, as shown in Figure 1. This dissertation tackles two challenges regarding the application of twins for our purposes: (1) How can we detect when deviations occur between a system and its digital twin? If we can detect this, can we pinpoint and correct the parts of the changed digital twin? (2) Can these techniques be integrated into software release management in a continuous deployment workflow? In what follows, we briefly describe our contributions to these challenges.

We drew inspiration from model verification and validation techniques to detect deviations between a twin and the real-world system. Model validation is one of the stages of designing a model; it is a rigorous process

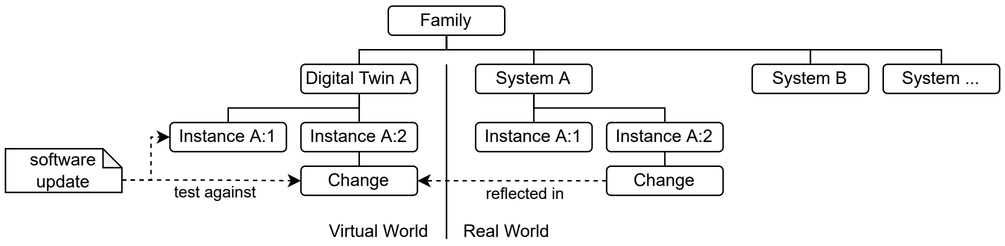

Figure 1: Premise of testing software updates utilizing the digital twin.

in which a set of validation experiments is executed on both the real-world system, possibly a prototype, and its model. The traces following that experiment can be further processed and compared to each other using model validation metrics. Based on this analysis, the model may be deemed valid, or, when deemed invalid, further steps can be taken: calibrate the parameters or refine the model. Afterwards, the model is ready for its intended use. To detect the deviations between a twin and the real-world system, we've applied these techniques at runtime, relying on random sampling methods and parallel simulation replication to generate data containing uncertainty, which is needed for some of the validation metrics. We've incorporated these techniques in a small framework containing an implementation of a set of relevant model validation metrics. Our results show that these techniques can successfully be applied during runtime, but we must be aware of potential pitfalls.

We drew inspiration from model-based fault localization to pinpoint and correct the part of the digital twin that changed. Model-based fault localization is an extension of model-based fault detection, where, based on fleet-leader inspection data, likely causes of faults can be classified. We identified Bayesian Networks and Machine-learning-based classification techniques as viable techniques for fault localization. In model-based fault localization, the knowledge of the fault would be used to start a correct recovery procedure, in our case we use this knowledge to recalibrate those models in the digital twin of the affected part. For this recalibration, we use the most recent operational data of the system. A potential issue arises when, after recalibration, the model behavior is still not satisfactory. In such cases, model refinement is needed, which still requires human intervention.

Currently, we are tackling the problem of software release management, where we use the previously discussed techniques of online validation to deploy over-the-air updates. Using the online validation technique in a continuous and automated fashion, we can detect previously unknown evolutions (unknown upgrades, wear-and-tear, etc.) of the actual systems. This continuous validation over the fleet of systems keeps track of variants in the fleet. It helps test engineers validate the software update on all the discovered variants of the actual system.

To make these processes more tangible, we integrated these techniques in a validation case study of a scale-model gantry crane. This gantry crane is a CPS that continuously moves containers back and forth, using its digital twin to plan its movements. We set up a continual monitoring scheme that checks for changes in the system, once detected, localizes the cause, and once localized, attempts to correct the model. When we update the gantry crane's controller, it can be tested against the latest model, reflecting the current state of the gantry crane.

ACKNOWLEDGMENTS

I would like to thank my advisor, prof. Anonymous , for his guidance during my career as Ph.D. student.