The digital twin as a common knowledge base in DevOps to support continuous system Evolution

The Digital Twin as a Common Knowledge Base in DevOps to Support Continuous System Evolution [⋆]

Joost Mertens[1,2][0000−0002−8148−5024] and Joachim Denil[1,2][0000−0002−4926−6737]

[1] University of Antwerp, Faculty of Applied Engineering, Antwerpen, Belgium
[2] Flanders Make @ University of Antwerp
{joost.mertens,joachim.denil}@uantwerpen.be

Abstract. There is an industry wide push for faster and more feature rich systems, also in the development of Cyber-Physical Systems (CPS). Therefore, the need for applying agile development practices in the model-based design of CPS is becoming more widespread. This is no easy feat, as CPS are inherently complex, and their model-based development is less suited for agile development. Model-based development does suit the concept of digital twin, that is, design models representing a system instance in operation. In this paper we present an approach where the digital twins of system instances serve as a common-knowledge base for the entire agile development cycle of the system when performing system updates. Doing so enables interesting possibilities, such as the identification and detection of system variants, which is beneficial for the verification and validation of the system update. It also brings along challenges, as the executable physics based digital twin is generally computationally expensive. In this paper we introduce this approach by means of a small example of a swiveling pick and place robotic arm. We also elaborate on related work, and open future challenges.

Keywords: DevOps · Digital Twin · Cyber-Physical System.

1 Introduction

In multiple industries, there is a drive to develop faster and more feature-rich systems. That is also the case for Cyber-Physical Systems (CPS) that integrate computation, communication and physical processes in a single system. Examples of these systems can be found in trains, vehicles, industrial machines, etc. The software of CPS is usually very complex and tries to control the system in uncertain environments. However, at design time it is hard to predict all the possible circumstances where the system needs to operate in. Furthermore, user requirements of the system may change during the operation of the system. This results in the demand for more agile approaches to CPS development and operations [5, 3] that bridge these phases of the system lifecycle.

[⋆] Joost Mertens is funded by the Research Foundation - Flanders (FWO) under strategic basic research grant 1SD3421N.

DevOps (Development and Operations) is a set of practices that looks to link design and operations of a system with each other in a continuum. Through continuous integration, delivery and deployment, the common idea is to achieve faster lead times for system updates [12]. Within the context of CPS, which are mostly designed in a Model-Based manner, the application of DevOps is feasible but challenging [8].

Given the model-based engineering approach of CPS, performing the most accurate design iteration necessitates a synchronization of the existing design models with real-life. In contrast with software engineering, physical systems age and undergo revisions independently from changes to the design models. To achieve the synchronization, operational data must be incorporated back into the design models. Because CPS are usually engineered using a set of models (architecture, behavior, etc.), the idea of continuously synchronizing a model with the real-world is not far-fetched, but feasible. In fact,this idea closely resembles a digital twin, which is often known as an executable physics-based model that represents a physical system. It can be used in several scenarios, from monitoring in which case it is more accurately called a digital shadow [13], to product life cycle management [10] and prediction, where it becomes clear that a link can be made to the initial question of how to incorporate operational data back to design models.

In [16], the uses of a digital twin throughout a DevOps cycle are made explicit. The digital twin serves as the enabler for carrying operational data back over to the development part of the DevOps cycle, but is also employed in other phases, e.g. in the test phase for testing against the most correct representation of the current system, or in the build phase, to virtually commission the system. We share this vision, and like to consider a digital twin as the common knowledge base that can be used throughout the entire DevOps cycle.

One additional point of interest is that of system variants. At design time, designers can account for a multitude of variants in a deployed system, yet at run time system variants develop naturally, for example hardware revisions of parts or degradation of electrochemical components such as batteries. In principle, such variations are detectable events for a digital twin. To clarify, at design time, the interpretation of variants is the same as in software product lines: a system configuration with specific features. However, at run time, we additionally deem a system that is no longer accurately represented by it's design time model a variant, in which case the model requires updating.

In [15], we shared a vision and the challenges related to using a digital twin as a means to bridge the design and operational phases of a system lifecycle. In this paper, we start by introducing a small example of a robotic arm making a pick and place swivel movement. Next, we elaborate on our approach of combining a digital twin with agile methods to update systems. We apply a subset of the approach on a system update of the small example as a demonstration. We then elaborate on some challenges, and discuss our proposed approach to tackle them. Afterwards we discuss related work, and lastly a conclusion summarizes the paper.

2 Example System

In this example, an electric motor performing a repetitive movement is studied, more specifically a swiveling robot arm performing a pick and place move. The arm is deployed in various locations around the world, were temperatures and humidity levels differ. In the example, we update the controller to perform a similar, but more rapid movement. Figure 1 illustratively shows this system.

Note that the physical system does not actually exist. Instead, we work on a simulated environment, with a different parameter set for a warm, cold and "lab" environment. This yields 3 models, one development model, and two "real-life" models. Those "real-life" models are conveniently reused as digital twins in this case. In a real setup this would not be the case, and the "real-life" models would be the physical system, instead of a model.

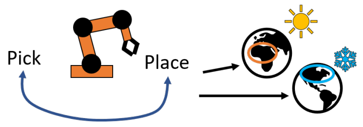


Fig. 1: Iconized example of the example system.

The system is modeled in Simulink®, shown in figure 2. The pick and place controller consists of a chain of PI(D) controllers that control the position of the arm and, the angular velocity and current of the motor. The controller generates the control signal to drive the inverter connected to the motor. The rotational movement is modeled using the SimScape library as a brushless DC motor that is driven by an inverter. The effect of the environment being cold or warm is incorporated as a viscous friction component in the rotational part of the BLDC model. In a cold environment, this friction component is higher than the in a warm environment, for example due to tighter tolerances or viscosity of lubricants. In the model this is characterized by the Coulomb friction. The only requirements for the system are for the pick and place movement to happen at a speed of 2 Hz, and with a steady state error of less than 0.75°. This is briefly summarized in table 1.

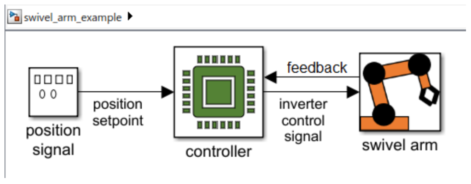


Fig. 2: Simulink®model of the system.

Description	Value
Pick & Place Move	-90° to 90°
Frequency	2 Hz
Error	<0.75°

Table 1: Model information.

During the development the friction component remains a question for the modeller, but we can assume that lab conditions suffice to tune the controller, that is, regular room temperature. After tuning and "deploying" the controller on the real system, that is, testing it on the models where the correct environmental conditions are applied, we find that the friction component matters little, and the swivel motion happens as expected. The movement of the swivel arm can be seen in figure 3. Model calibration reveals that to be entirely correct, the Coulomb friction must be 0.5 Nm in the cold climate and 0.2 Nm in the the warm climate.

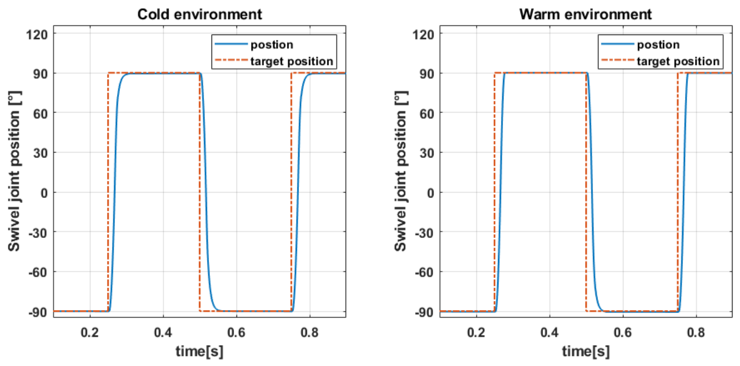


Fig. 3: Traces for the initial controller in both environments.

As can be seen in figure 3, the swivel arm reaches the setpoint reasonably quickly, and, inspecting the difference at steady yields worst case absolute errors of 0.46° and 0.53° for the cold and warm environments respectively. In other words, the controller tuned under lab conditions suffices for both the actual in-the-field conditions.

3 Approach

Our approach combines digital twins with agile methods to update systems, with a specific interest in the verification and validation of the updates. The foundation of the approach is multi-paradigm modeling [19]. Multi-paradigm modeling advocates the explicit modeling of all parts of a system, at the most appropriate level of abstraction, and in the most appropriate formalism. In other words, in our approach we thus look to model the various parts of the system: real-world representative system models (digital twins), deployment models, architectural models of the whole, networking models for communication. This allows to perform different processes on a deployed system, as the architectural overview in figure 4 depicts. Next we explain the process of release management and variant detection in more detail.

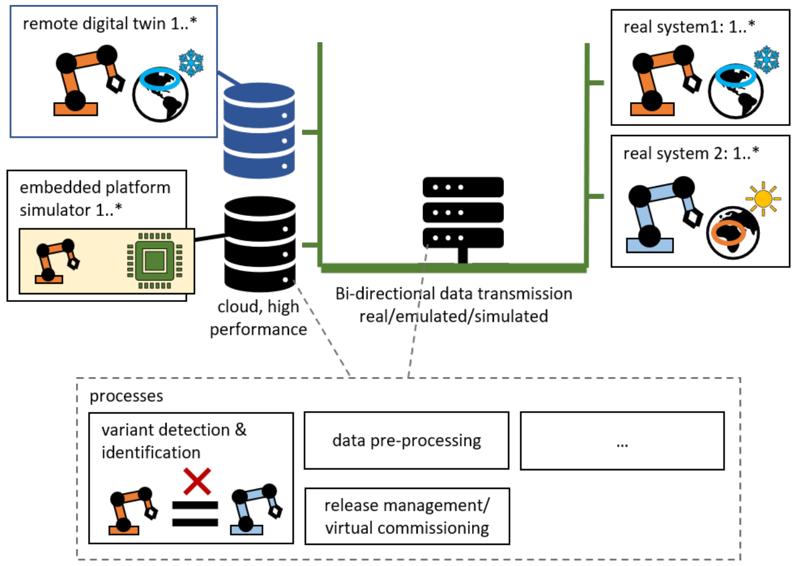

remote digital twin 1..*
embedded platform simulator 1..*
cloud, high performance
Bi-directional data transmission real/emulated/simulated
real system1: 1..*
real system 2: 1..*
processes
variant detection & identification
data pre-processing
...
release management/ virtual commissioning

Fig. 4: Architectural overview of the approach.

3.1 Variant Detection

Our approach for the detection of new, undocumented variants relies on the digital twin. Figure 5 shows the process of discovering new variants of the system configuration and environments. The first activity is the detection of deviations between the real system and the digital twin within a certain tolerance. The means to detect a deviation are dependent on the digital twin technology. For example, when using a Kalman filter-based digital twin, the observation over the Kalman filter gain is a possible means to detect deviations between the model and the real world. The deviation can be purely parameter-based. In such a case, the structure of the model is still valid. This results in a new calibration of the model using the data from the real-world. If the re-calibration is successful, a new variant is detected in the real world.

If the calibration fails, the model structure is not sufficient to describe the state of the real system. This can happen when a component of the real-system has changed (e.g. a different DC motor is used because of a repair). In this case, another model is needed to describe the state of the system. When a library of models is available (with different alternative models), the variant detection searches within the library for possible configurations. We rely on the validity frame [4, 1, 17, 18] to describe the validity range of the model. The Validity Frame covers the general concept of validity of a simulation model. In [4], four uses of the Validity Frame concept are analysed: (i) defining the validity of a model, (ii) model/component discovery, (iii) calibration of a model, (iv) defining the experimental process. In the context of this paper, (ii) and (iii) are of interest. When no valid model can be found, the developers need to review the data and find possible system configurations that explain the deviation from the data. This can happen when a non-standard component was used for the repair of the system or because of wear and tear of the system.

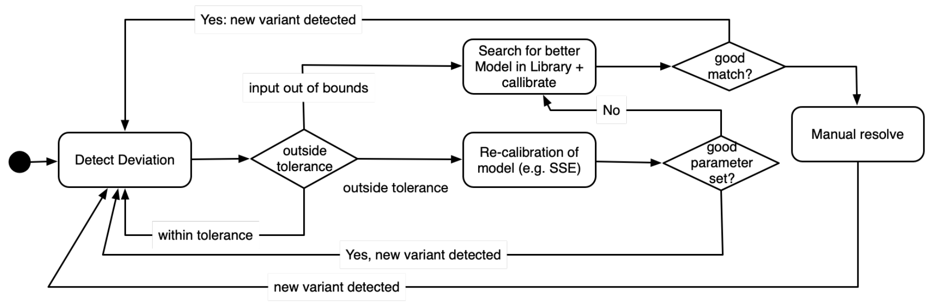


Fig. 5: Workflow for detection system variants using digital twins.

Similarly, another deviation happens when the input of the real system is outside the valid input bounds of the model. The same procedure with the validity frame is used to detect and resolve such a deviation. When no model is available, the environment in which the system now operates is not taken into account during design of the system. It is therefore necessary to resolve the deviation manually.

In each of the cases, a new variant is added to the library of variants that is used by the release management and virtual commissioning component.

3.2 Release Management and Virtual Commissioning

Figure 6 shows the simplified workflow for the virtual commissioning of the system update on the different variants. From the variant detection mechanism, the virtual commissioning component knows which different variants exist of the system. For each of the different variants, an automatic verification and validation experiment is set up. If the experiment passes for the variant, the software update can be deployed onto the systems that are represented by that specific variant. Note that for most systems, in the interest of safety, a human intervention is still required to sign off on the release. This can be provided by showing the proof of the V&V activity to the human to make a decision on deployment of the update.

Because of the nature of the system-under-test, the experimentation environment is build from a co-simulation setup using FMI co-simulation [2]. This is necessary as a typical CPS system is created with different models in different languages. However, some of the requirements can only be checked at the system level, for which the different sub-systems must be integrated to evaluate the system-level behavior, thus requiring co-simulation.

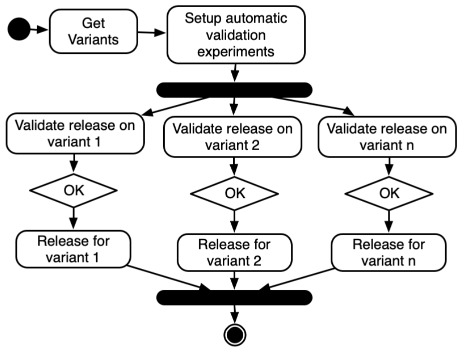

Get Variants
Setup automatic validation experiments
Validate release on variant 1
Validate release on variant 2
Validate release on variant n
OK
OK
OK
Release for variant 1
Release for variant 2
Release for variant n

Fig. 6: Simplified virtual commissioning workflow without error reporting paths.

We also expect that system requirements are available that can be automatically checked. Simulation typically outputs traces of the signals in the simulated system. The requirements specified over these traces should be evaluated. For temporal properties, we rely on monitors generated from temporal logics such as Signal Temporal Logic [6]. Other properties, such as energy performance, typically require an extra step to be computed from the traces (e.g. the integration of the signal in time).

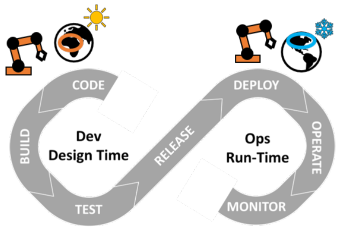
4 Application of the approach

As a proof-of-concept for the approach, we show a new release for the control mechanism on the arm. The user of the system wants more performance and demands for a more rapid movement of 10 Hz instead of 2 Hz. This is shown in the system information in table 2. We study the system update in two cases, one with and one without a DevOps loop with a Digital Twin as knowledge base.

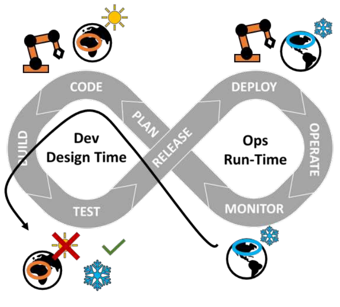
Description	Value
Pick & Place Move	-90° to 90°
Frequency	2 Hz
Error	<0.75°

Table 2: Updated system information.

Updating without DevOps In this first update, we assume no DevOps with a digital twin as common knowledge base is used. The concept is illustrated in figure 7a, which shows that the system in operation is monitored, but no feedback is given for future developments. The new controller is therefore still



(a) Development and deployment without feedback.



(b) DevOps assuming digital twin as common knowledge base.

Fig. 7: Update schemes for the system update.

created for, and tested in lab conditions, even though it may be deployed in different conditions.

Inspecting the results in figure 8, at first glance both systems seem to pass the requirements, but on closer inspection, in the cold environment the arm fails on the maximum allowable absolute error. It hits a worst case number of -1.57°, which is far past the 0.75° allowed value. The arm in the warm environment performs alright, with an error of only 0.61°.

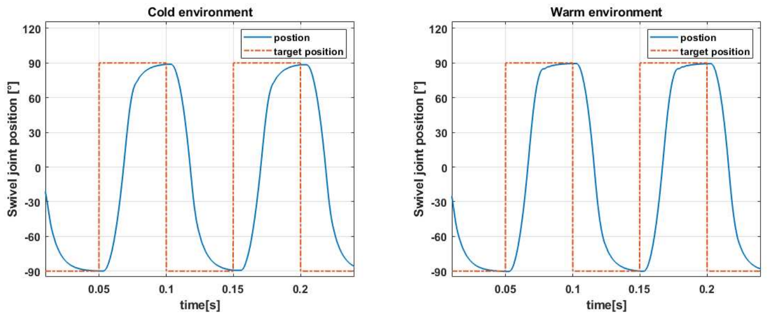


Fig. 8: Traces for the updated controller in both environments.

Updating with DevOps and digital twin In this second case, we assume a digital twin (or digital shadow), of the arm exists. Since the digital twin is continuously calibrated with the real system, the specific environment in which the robotic arm is located is identified. As per the flow in figure 5, a deviation detection notifies us of a difference between the model and the real world. In this case, given that structurally the model is valid, a re-calibration of the model suffices. After re-calibration, a proper new value for the friction in the cold environment is found and the software update verified and validated. Figure 7b shows this system where monitored data at runtime is used at design time. Since the DevOps loop is now closed, instead of relying on the design models only, the characterized environment is processed and utilized during the testing of the new controller. Upon deployment, each robot arm is tested in its own specific environment, yielding more accurate results.

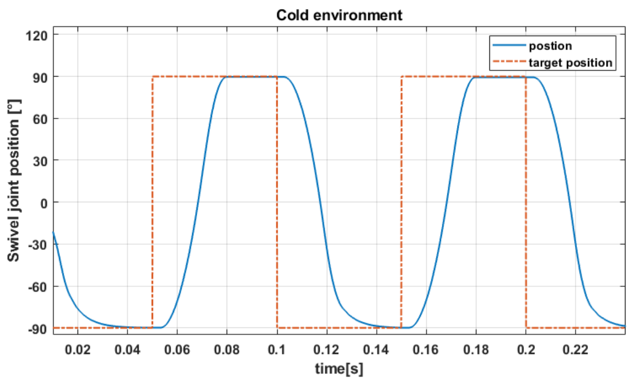


Fig. 9: Traces for cold environment, with corrected controller.

For the warm environment nothing changes, except that, if needed the controller can be adjusted even better to the actual condition. For the cold environment, testing against the calibrated digital twin would show the failed results from figure 8, and the designer is able to adjust the controller to that environment. Indeed, with some tweaking of the differential gain parameter of the position controller, the maximum absolute error in the cold environment can also be brought to 0.62°, passing the test. Those results are plotted in figure 9.

5 Discussion

Although the example is small, it demonstrates well how having feedback from existing systems can aid in the verification process of new feature or update for already deployed systems. We rely on DevOps in figure 7b since it the most complete set of practices for continuous feedback. One might argue that the approach does not specifically need the use of digital twins to detect the different environments, and that the problem could have been prevented by more thoroughly defining tests in section 2, but that is a moot argument, since one cannot account for all possible variations. Alternatively, instead of varying the environmental conditions, we could also have studied a case in which physical components such as the motor are replaced by equivalent but not 100% identical parts. From the approach, the example shows two possibilities that stem from using a digital twin as knowledge aggregator for data:

– Detecting undocumented variants in deployed systems or their environments. Once detected, those variants can be used to aid in the updating of existing or the development of new systems.
– Detecting run-time problems. A physics-based, executable digital twin, allows to detect run-time problems that cannot be identified on the deployed system itself, e.g. by being too computationally heavy.

Besides these possibilities, the approach brings along some new challenges:

C1 When using a digital twin as knowledge base, where should its executable parts be ran and where should its data be stored? Various options are possible, such as machine-local, on a centralized server in the cloud, or using some decentralized scheme of edge computing per operational site.
C2 Is it feasible to detect and uncover every potential variant in a computational sense, and test for every discovered variant? That's to say, is this scalable to larger number of systems, such as in a fleet of vehicles. Can heuristics be used to make this approach more scalable, e.g. by aggregating data from multiple systems into a single representative digital form. Distribution towards the edge and local system might help.

Some parts of the approach do remain unexplored by the example, more specifically the embedded platform simulator and along with it a more realistic process of release management/virtual commissioning. Our main solution for handling both challenges is by modeling these explicitly. Performance models are under construction to explicitly reason over the deployment issues. Sensitivity analysis on the different input and parameters will be used to automatically reason over lumping together individual variants. Furthermore, as certain subsystem tests are shared between variant classes, an analysis of the test set might increase the scalability of the approach. Finally, the challenges are not independent, as evaluation can be distributed based on the type of feature. Features that are much more customized benefit from distribution towards the edge and system, while features that are very common might benefit from centralization. This remains future work.

6 Related Work

Papers such as [3, 20, 5] make note of the ever faster recurring development cycle for CPS. In the CPS domain, where model-based engineering is widely applied due to its abstraction of complexity, there is thus a need for practices such as DevOps. In [3], the challenge of transferring data back from the running system to the models is specifically noted. In [14], it is clearly identified that model-based design practices and digital twin practices integrate well with eachother, and in [16], a description of how the digital twin can be used throughout the DevOps stages is elaborated on. That is also why we like to call the digital twin the common-knowledge base in the DevOps loop, and such a knowledge base can consist of executable models but also non-executable models. Specifically in the verification and validation stage, it is noted in [16] how the digital twin can be used to ensure the system is represented as best as possible, though no note is made on the identification variants, which is where we believe we can further contribute. On the topic of testing, we must note that this remains one of the larger challenges for the adoption of DevOps in the development for CPS, as noted in [21]. Indeed, the development of CPS relies on thorough testing, often followed by safety and/or security certification. In this respect, the potential contributions of the presented approach will be limited to identifying problems earlier on, since we remain in a modeled world, and certification happens on a codebase. Models can however help with producing certifiable code more easily. It's also on the model aspect that we differentiate from other variant management approaches such as [11], which operate on a code level, and uses emulation. This is interesting from the certification point of view, but less interesting from the early testing point of view. In [7] however, it is stated that for continuous integration chains, fast transaction level simulation is proven to work for the simulation of virtual hardware, and is more scalable than testing with hardware. Another aspect is that of the run-time validity monitoring. Where the runtime monitoring of Validity Frames is currently limited to input/output value/datatype and relation monitoring, for correct variant detection, it might be needed to incorporate other monitors as well. In this regard in [9], the concept of multilevel monitors is presented for detecting attacks or faults. A distinction is made between Data Monitors, Functional monitors and Network monitors. The data monitoring and functional monitoring are similar to the I/O monitoring in Validity Frames, but the overlapping monitors and network monitoring presents interesting insights that can be used to extend the monitoring capabilities of Validity Frames.

7 Conclusion

In this paper we elaborated on our vision of utilizing the digital twin of a system in the agile deployment of new system updates. The approach builds on the idea of using the digital twin as a common knowledge base throughout the DevOps cycle. We demonstrated a subset of the approach on a small example study, from which we elaborated some open challenges that remain to be researched. Lastly, we also elaborated on how related work matches our vision, and how our approach can contribute to or differs from that existing work.

References

1. Acker, B.V., Meulenaere, P.D., Denil, J., Durodie, Y., Bellinghen, A.V., Vanstechelman, K.: Valid (re-)use of models-of-the-physics in cyber-physical systems using validity frames. In: 2019 Spring Simulation Conference (SpringSim). pp. 1–12 (2019). https://doi.org/10.23919/SpringSim.2019.8732858
2. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al.: The functional mockup

interface for tool independent exchange of simulation models. In: Proceedings of the 8th International Modelica Conference. pp. 105–114. Linköping University Press (2011)

3. Combemale, B., Wimmer, M.: Towards a model-based devops for cyber-physical systems. In: Bruel, J.M., Mazzara, M., Meyer, B. (eds.) Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. pp. 84–94. Springer International Publishing, Cham (2020)
4. Denil, J., Klikovits, S., Mosterman, P., Vallecillo, A., Vangheluwe, H.: The experiment model and validity frame in m&s. In: SpringSim (2017)
5. Denil, J., Salay, R., Paredis, C., Vangheluwe, H.: Towards agile model-based systems engineering. In: MODELS (Satellite Events). vol. 2019, pp. 424–429. CEUR Workshop Proceedings (2017), http://ceur-ws.org/Vol-2019/
6. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for stl. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 264–279. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
7. Engblom, J.: Continuous integration for embedded systems using simulation. In: Embedded World 2015 Congress (2015)
8. Garcia, J., Cabot, J.: Stepwise adoption of continuous delivery in model-driven engineering. In: International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. pp. 19–32. Springer (2018)
9. Gautham, S., Jayakumar, A.V., Elks, C.: Multilevel runtime security and safety monitoring for cyber physical systems using model-based engineering. In: International Conference on Computer Safety, Reliability, and Security. pp. 193–204. Springer (2020)
10. Grieves, M.: Origins of the digital twin concept. In: Working Paper. pp. 1–7. Florida Institute of Technology (08 2016). https://doi.org/10.13140/RG.2.2.26367.61609
11. Guissouma, H., Lauber, A., Mkadem, A., Sax, E.: Virtual test environment for efficient verification of software updates for variant-rich automotive systems. In: 2019 IEEE International Systems Conference (SysCon). pp. 1–8 (2019). https://doi.org/10.1109/SYSCON.2019.8836898
12. Kim, G., Humble, J., Debois, P., Willis, J.: The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution (2016)
13. Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W.: Digital twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine **51**(11), 1016 – 1022 (2018). https://doi.org/10.1016/j.ifacol.2018.08.474, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018
14. Madni, A.M., Madni, C.C., Lucero, S.D.: Leveraging digital twin technology in model-based systems engineering. Systems **7**(1), 7 (2019)
15. Mertens, J., Denil, J.: Digital twins for continuous deployment in model-based systems engineering of cyber-physical systems. vol. 2822, pp. 32–39. CEUR Workshop Proceedings (2020), http://ceur-ws.org/Vol-2822/
16. Querejeta, M.U., Etxeberria, L., Sagardui, G.: Towards a devops approach in cyber physical production systems using digital twins. In: International Conference on Computer Safety, Reliability, and Security. pp. 205–216. Springer (2020)
17. Van Acker, B., Oakes, B.J., Moradi, M., Demeulenaere, P., Denil, J.: Validity frame concept as effort-cutting technique within the verification and validation of

complex cyber-physical systems. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. MODELS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3417990.3419226

18. Van Mierlo, S., Oakes, B.J., Van Acker, B., Eslampanah, R., Denil, J., Vangheluwe, H.: Exploring validity frames in practice. In: Babur, Ö., Denil, J., Vogel-Heuser, B. (eds.) Systems Modelling and Management. pp. 131–148. Springer International Publishing, Cham (2020)
19. Vangheluwe, H., De Lara, J., Mosterman, P.J.: An introduction to multi-paradigm modelling and simulation. In: Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal. pp. 9–20 (2002)
20. Warg, F., Blom, H., Borg, J., Johansson, R.: Continuous Deployment for Dependable Systems with Continuous Assurance Cases. In: 2019 IEEE International Symposium on Software Reliability Engineering, WoSoCer workshop. IEEE Computer Society (2019)
21. Zeller, M., Ratiu, D., Rothfelder, M., Buschmann, F.: An industrial roadmap for continuous delivery of software for safety-critical systems. In: 39th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Position Paper (2020)