# TwinOps – DevOps meets Model-Based Engineering and Digital Twins for the engineering of CPS


Jerome Hugues
Carnegie Mellon University – Software Engineering Institute
Pittsburgh, PA
jhugues@andrew.cmu.edu

Anton Hristosov
Carnegie Mellon University – Software Engineering Institute
Pittsburgh, PA
adhristozov@sei.cmu.edu

John J. Hudak
Carnegie Mellon University – Software Engineering Institute
Pittsburgh, PA
jhudak@sei.cmu.edu

Joe Yankel
Carnegie Mellon University – Software Engineering Institute
Pittsburgh, PA
jdyankel@cert.org



## ABSTRACT

The engineering of Cyber-Physical Systems (CPS) requires a large set of expertise to capture the system requirements and to derive a correct solution. Model-based Engineering and DevOps aim to efficiently deliver software with increased quality. Model-based Engineering relies on models as first-class artifacts to analyze, simulate, and ultimately generate parts of a system. DevOps focuses on software engineering activities, from early development to integration, and then improvement through the monitoring of the system at run-time. We claim these can be efficiently combined to improve the engineering process of CPS.

In this paper, we present TwinOps, a process that unifies Model-based Engineering, Digital Twins, and DevOps practice in a uniform workflow. TwinOps illustrates how to leverage several best practices in MBE and DevOps for the engineering Cyber-Physical systems. We illustrate our contribution using a Digital Twins case study to illustrate TwinOps benefits, combining AADL and Modelica models, and an IoT platform.



## CCS CONCEPTS

• **Computing methodologies** → **Model development and analysis**; • **Theory of computation** → **Timed and hybrid models**.

**ACM Reference Format:**
Jerome Hugues, Anton Hristosov, John J. Hudak, and Joe Yankel. 2020. TwinOps – DevOps meets Model-Based Engineering and Digital Twins for the engineering of CPS. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3421446




*MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada*



ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3421446


## 1 INTRODUCTION

The increase in complexity of Cyber-Physical Systems (CPS) is causing a wide range of undefined behaviors. Key issues such as imprecise component characterization (in the functional, timing, or safety viewpoints) or emergent system behaviors such as system deadlocks or erratic behaviors are still discovered during testing or after the system has been deployed.

Models can address some of these concerns: systems engineering models to capture system requirements, its interface, and their decomposition into subsystems; simulation models to evaluate the system general behavior; engineering models to move forward towards the realization of the system. Engineering models can in turn lead to model transformations towards analytical models (e.g. for model checking, performance evaluation) and ultimately code generation. Finally, Digital Twins models can provide insights into the system execution.

The late discovery of issues in system design is detrimental. We note they are usually considered in isolation and developed concurrently. We claim that these models can be combined in a uniform process to improve the whole engineering process. We propose TwinOps, a process that combines DevOps practice and Model-Based code generation practice to facilitate system deployments for multiple targets: to build simulation testbench, validation platform, and Digital Twins of a Cyber-Physical System.

In the following, we present in section 2 an overview of our technological landscape. Then, we introduce the TwinOps process (section 3). We illustrate this process on a case study (section 4) and cover the various steps of TwinOps in sections 5, 6, and 7. Section 8 shows how to bind TwinOps to a Continuous Integration/Continuous Deployment pipeline, followed by the conclusion.

## 2 TECHNOLOGICAL LANDSCAPE

Our objective is to combine different technologies to gain the best of each. In this section, we provide a brief overview:

*Model-Based Engineering.* relies on models as first-class abstraction of a system under study [13]. In [6], the authors show the critical impacts of automated code generation on the engineering of embedded software in both increased confidence in produced software and fast delivery. Yet, this is usually a one-way process

with challenges in debugging generated software or inform model updates.

*DevOps.* is defined in [10] as a "collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability." As a process, DevOps focuses on automation to expedite specific steps such as software building, testing, or deployment. Beyond the human organization, DevOps focuses on 1) automation to discharge engineers from error-prone tasks so that they can focus on core activities such as feature update or debugging and 2) monitoring of the system execution to debug or update the software.

*Simulation.* capabilities [5] are important to perform early validation of CPS. Co-simulation supports model execution as a federation of executable models. Some standards like FMI [1] provides a standardized interface to interconnect models and associated simulation environment. It enables Digital Twins.

*Digital Twins.* consist "of three components, a physical product, a virtual representation of that product, and the bi-directional data connections that feed data from the physical to the virtual representation, and information and processes from the virtual representation to the physical" ([8]). These links support system continuous improvement and maintenance through the analysis of runtime logs and their comparison to system optimal performance.

In [3], the authors present a roadmap for Model-Based DevOps for CPS. They propose a collection of challenges to be addressed to better couple MBE and DevOps. They mention the coupling between "Dev" and "Ops" as hot research topics. In the following, we present TwinOps, our contributions to these topics.

## 3 TWINOPS – DEVOPS/MODEL-BASED/DIGITAL TWINS COMBINED

TwinOps refines the typical DevOps "infinite loop" steps:

- *Modeling* encompasses modeling and source code definition. We consider that source code is the ultimate machine-processable model of the function to be implemented. This step encompasses the following steps of DevOps:
  (1) *Plan, Requirements definition, and properties*: define the systems engineering models of the system, along with a validation plan;
  (2) *Modeling architecture and parts*: refine the models and define domain-specific models to cover the various parts. Models address specific concerns captured in the previous phases (e.g. need to model the environment, or control, or architecture of an eMBEdded system, etc.);
  (3) *Virtual Integration*: defines the interaction points between these models, e.g. how the realization of an architecture executes specific functions or associated engineering models and the environment model.
- *Test Bench/system realization* is an automated software factory that builds the various artifacts: simulation code, executables.
  (1) *Code Generation* produces code from models with multiple objectives: generating functional and middleware code to run on the target, generating simulation elements. Also, glue code is generated to 1) monitor properties such as resource consumption or data exchange, 2) detect specific execution patterns.
  (2) *Testbench Assemly* combines the various pieces to build the multiple targets: software to run on the target, or simulation of the system with multiple levels of fidelity.
- *"Ops"* deploys and executes the generated software. It collects data (*"Monitor"*) and confront the various data for accuracy and consistency (*"Data Analysis"*). This is made possible thanks to the probes generated in the previous steps from design patterns deployed during the modeling phase. The data can be collected by the execution of the simulation, the execution of the system, or the analysis of traces from its "Digital Twin".
  The outcome of the data analysis phase will inform follow-up updates to the system requirements and properties, and updates to the system design.

Hence, TwinOps is defined as a variation of DevOps, with MBE used during both the "Dev" part to generate part of the system, but also the monitoring part of the "Ops". This approach allows for a rapid and iterative cycle: probes are design patterns and code generation strategies that accelerate data collection. Code generation is an explicit step from "Dev" to "Ops", and data analytics from "Ops" to "Dev". This explicit step allows for fine-tuning of the target to monitoring objectives, the verification that system requirements are fulfilled.

The definition is generic enough to be instantiated in multiple ways and accommodate the nature of the system. In the next sections, we illustrate one instance of TwinOps based on a combination of SysML, AADL, Modelica, and FMI using GitLab CI/CD capabilities and the Azure IoT cloud platform.

## 4 CASE STUDY

In this section, we introduce our case study. It illustrates the TwinOps process on a small but representative demonstrator.

### 4.1 Sensor Processing – Introduction

Let us assume we want to build a monitoring system for a building. The system will monitor and collect environmental conditions to ensure the proper operation of an air conditioning system. The system participates in a Digital Twin of the building. We elicit the following requirements:

(R1) The system shall monitor the humidity and temperature in multiple points of a building, every 10 minutes during office hours, or every 30 minutes;
(R2) The system shall gather all data in a central repository;
(R3) The system shall detect and report any error in the reported data such as out-of-range or suddent surge in values;
(R4) The system shall monitor its health status and report issues.

From these considerations, an industrial survey shows that a platform built on the Azure IoT Cloud platform for data management and a Raspberry Pi platform with a BME280 sensor device

could deliver the expected functionalities[1]. The Azure IoT framekwork associated to a Raspberry Pi board supports building a Digital Twins of the building to control its temperature.

Several open-source projects provide similar capabilities, except for the error detection parts. In the following sections, we illustrate how Model-Based Engineering and DevOps could be combined to support the definition and engineering of this system, and how combining MBE code generation and Digital Twins provide feedback to improve the model.

## 5 TWINOPS MODELING ACTIVITIES

In this first step "Plan, Requirements definition and properties", we define the requirements of the system, its decomposition as subfunctions, and use case scenarios attached to it.

*TwinOps solution #1: "Use containers for delivering modeling environments".* Following a DevOps philosophy, a first concern is to ensure all team meMBErs use the same baseline for the modeling environment. Here, we propose to use docker containers to build a reproducible modeling environment [2]. We defined a container with our Eclipse baseline for modeling environment, comprised of Papyrus Sysml 1.6, OSATE AADL toolchain, and Modelica MDT tools. Their use scenarios are detailed below.

. Using this environment, we could capture the first set of models of our system: a collection of OMG SysML 1.6[12] diagrams that capture the high-level requirements of our system, use cases, and first-level system decomposition.

In a second step, "Modeling architecture and parts", we refine these models as AADL models. These capture the eMBEdded system architecture as a collection of processors, buses, devices, and software attached to it. Our choice for AADL has been dictated by our team expertise in AADL, and the direct availability of many tool capabilities to perform early analysis of the system.

*TwinOps solution #2: "Perform virtual integration of models".* We leverage the ALISA DSL [4] to refine requirements into verifiable items attached to target metrics. An ALISA verification plan binds requirements to verification methods to be executed (usually a verification plug-in) and reports on any discrepancy. This virtual integration ensures that the model – as currently engineered – can be integrated on the target platform and meet stated performance metrics.

. We combined the AADL model with an ALISA verification plan and evaluated some key metrics such as the nuMBEr of messages processed per unit of times, energy consumption, etc. ALISA verification plan can be executed from within Eclipse, or integrated as tests in a regular test suite environment such as JUnit.

*TwinOps lessons learned.* TwinOps builds on a Model-Based CI/CD pipeline that contains both models and reproducible modeling environment. Model-level analysis and evaluation of some metrics. Discrepancies can later lead to model refactoring. This results in the modeling pipeline (see figure 1). Initially, sysML and AADL modeling steps are performed, then ALISA validation may either detect an error or continue to the next step.

[1] Azure IoT/RPi demonstrator

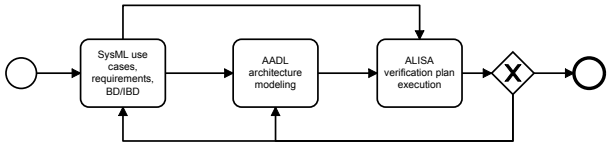


**Figure 1: TwinOps Modeling pipeline**

## 6 TEST BENCH/SYSTEM REALIZATION

This second step addresses dual objectives: support V&V activities and deliver the final system. One limit in the previous MBE CI/CD pipeline is that not all properties may be assessed at model-levels. Figure 2 illustrates some contributors to issues that can only be evaluated at runtime: timing budgets for end-to-end flows (highlighted flow in yellow) may not be respected by the implementation or communication bus, devices may experience some bias at runtime (in blue) that must be detected and mitigated or loss of the connection to the logging facility (in orange).

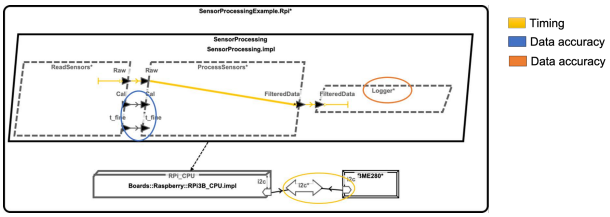


**Figure 2: Runtime monitoring points**

In this second step, we implement the system and enrich it with monitoring probes. We leverage existing models to perform extensive code generation from the architectural model description (figure 3). First, we implement software probes. Probes either validate input data or measure the execution time of functions. We also implement the core logic of our application. Then, we use the Ocarina AADL code generator [9] to generate code. These three source code elements are combined to produce the final binary.

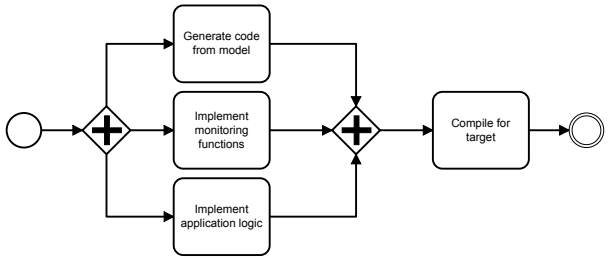


**Figure 3: TwinOps Code generation pipeline**

*TwinOps solution #3: "Multiple targets code generation".* Code generation is a generic process. From an AADL model, Ocarina generates minimal middleware that supports the execution of the model: tasks, communication buffers, ports, etc. The targeted language can be C (running on a variety of RTOS or POSIX), Ada, or

formal languages for simulation and model-checking like LNT [11]. Such multiplicity in targets allow for diverse means to evaluate the system:

(1) LNT supports executing functional C code embedded in a formal model of the system, and state-space exploration for safety or liveness properties.
(2) C allows for direct execution on the target, using devices drivers or a mock-up of the device implemented as a Functional Mock-up Unit [7].

We discuss these scenarios in the next section.

## 7 SYSTEM EXECUTION AND ANALYSIS AT RUN TIME

In the previous section, we illustrated how code generation can support multiple targets. They extend the model-level analysis with more precise evaluation:

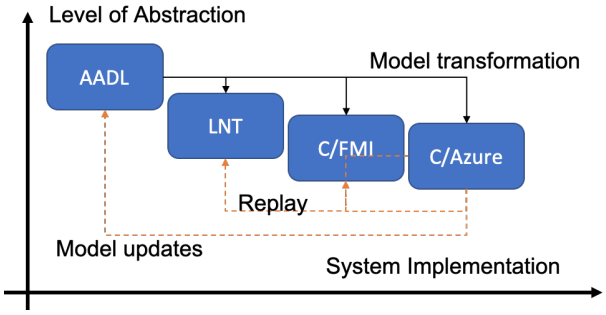


**Figure 4: TwinOps Feedback Loop**

### 7.1 From System Execution ...

*The LNT target* enables model-checking capabilities, weaving an abstract model of the environment and the execution platform with actual functional code. This allows for a systematic evaluation of the functional side of the system but may be limited to some platform-specific aspects: error in sensors, timing issues, etc.

*The C/FMI* target with device mock-ups leverage the FMI standard to build a simulated environment using a Modelica model to capture the physical environment. For our sensor demo, we used a first-principles model of the sensor device and the generation of temperature and pressure from a meteorological simulation. Using FMI allows us to define specific use scenarios by adjusting physical variables; while evaluating the actual execution on the target.

*The C/Azure target* with execution on the target platform allows for the execution of the system and its monitoring. We generated specific monitoring probes. These probes collect all data and send them on the cloud to an Azure IoT Digital Twin of the system. The Digital Twin is a representation of the system in terms of its state properties, telemetry events, commands, components, and relationships. It is a data stream that can be queried.

### 7.2 ...To System Analytics

All targets are ultimately combined to improve the system through data analytics: The *LNT* or *C/FMI* targets use data collected from the *C/Azure* to replay specific execution traces. Since all targets share the same code base, they are various representations of the same system at various levels of fidelity.

Finally, the same data can lead to model improvements. For instance, timing traces can be compared to theoretical time budgets used for latency or scheduling analyses, sensor biases can lead to a different mitigation policy, for instance, to force specific recalibration. Hence, such a comparison between execution traces and the initial model can inform updates of the system to improve its accuracy.

The combination of model transformation and code generation; and the automated integration of monitoring probes support the feedback loop prescribed by DevOps philosophy: the capability to monitor the system at "Ops-time" to inform updates during "Dev-time" (figure 4).

## 8 INTEGRATION AS A DEVOPS CI/CD PIPELINE

In the previous sections, we have presented a mapping of modeling, model transformation, and code generation activities to a notional DevOps pipeline. We integrated these steps in a Continuous Integration/Continuous Delivery pipeline using the GitLab platform. This pipeline supports all steps that could be automated: model transformation or code generation, compilation, testing activities, containerization, and deployment on targets.

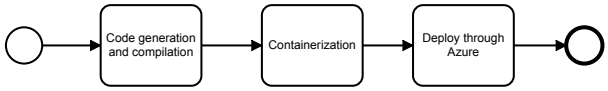


**Figure 5: TwinOps Deployment**

To facilitate deployment, we build a docker container that hosts the binary along with its dependencies and stores it in a container registry. This supports a reproducible runtime environment across multiple targets. The final step in our pipeline is the deployment of the container on the target. We leverage the Azure IoT capability to send a request to all targets to update and execute the latest released version in the container registry.

The configuration of the GitLab pipeline is currently a manual process. Future work will involve linking GitLab configuration to a model that configures the CI/CD pipeline and the set of deployment targets in a uniform way.

## 9 CONCLUSION

The development of Cyber-Physical Systems is facing multiple hurdles: functional correctness, but also adaptation between the software, the hardware platform, and its environment. This is impacting significantly the Validation and Verification of project development. We claim new strategies leveraging advances in MBE can provide significant benefits.

In this paper, we have introduced TwinOps, a process that combines the modeling semantics of MBE with the automation and process advocated by DevOps, and the coupling aspects in Digital Twins to improve the engineering of Cyber-Physical Systems.

TwinOps builds on Model-Based Engineering model transformation and code generation to bridge the Dev and Ops sides and generate multiple representations of the systems: formal model, instrumented software, or Digital Twins. These representations being automatically generated, they can be adapted to monitor specific run-time conditions and report relevant data for further analysis and system improvements.

In its current implementation, TwinOps leverages multiple AADL tool-based capabilities to build these systems instances, by connecting Digital Twins through Azure IoT capabilities. Future work will consider deciding on model updates from the data logs collected and propose other variants based on SysML and other modeling notations.

## REFERENCES

[1] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, Dietmar Neumerkel, Hans Olsson, Jörg-Volker Peetz, and Susann Wolf. 2011. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. *Proceedings of the 8th International Modelica Conference*, 105–114. https://doi.org/10.3384/ecp11063105

[2] Carl Boettiger. 2014. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Oper. Syst. Rev.* 49 (10 2014). https://doi.org/10.1145/2723872.2723882

[3] Benoît Combemale and Manuel Wimmer. 2019. Towards a Model-Based DevOps for Cyber-Physical Systems. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6-8, 2019, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12055)*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer, 84–94. https://doi.org/10.1007/978-3-030-39306-9_6

[4] Julien Delange, Peter Feiler, and Ernst Neil. 2016. Incremental Life Cycle Assurance of Safety-Critical Systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. TOULOUSE, France. https://hal.archives-ouvertes.fr/hal-01289468

[5] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2017. Co-simulation: State of the art. *CoRR* abs/1702.00686 (2017). arXiv:1702.00686 http://arxiv.org/abs/1702.00686

[6] Katerina Goseva-Popstojanova, Teme Kahsai, Matt Knudson, Thomas Kyanko, Noble Nkwocha, and Johann Schumann. 2016. *Survey on Model-Based Software Engineering and Auto-Generated Code.* Technical Report NASA/TM–2016–219443. NASA.

[7] Jérôme Hugues, Jean-Marie Gauthier, and Raphaël Faudou. 2018. Integrating AADL and FMI to Extend Virtual Integration Capability. In *Proceedings of 9th European Congress Embedded Real Time Software and Systems ERTSS 2018, Toulouse, France, Jan 31-Feb 2 2018.* https://dblp.org/rec/bib/journals/corr/abs-1802-05620

[8] David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. 2020. Characterising the Digital Twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology* 29 (2020), 36 – 52. https://doi.org/10.1016/j.cirpj.2020.02.002

[9] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérôme Hugues. 2009. Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Reliable Software Technologies - Ada-Europe 2009, 14th Ada-Europe International Conference, Brest, France, June 8-12, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5570)*, Fabrice Kordon and Yvon Kermarrec (Eds.). Springer, 237–250. https://doi.org/10.1007/978-3-642-01924-1_17

[10] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.* 52, 6, Article 127 (Nov. 2019), 35 pages. https://doi.org/10.1145/3359981

[11] Hana Mkaouar, Bechir Zalila, Jérôme Hugues, and Mohamed Jmaiel. 2020. A formal approach to AADL model-based software engineering. *Int. J. Softw. Tools Technol. Transf.* 22, 2 (2020), 219–247. https://doi.org/10.1007/s10009-019-00513-7

[12] OMG. 2019. *OMG Systems Modeling Language (OMG SysML) Version 1.6.* Technical Report formal/19-11-01. OMG.

[13] Alberto Rodrigues da Silva. 2015. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures* 43 (2015), 139 – 155. https://doi.org/10.1016/j.cl.2015.06.001