Towards continuous verification and validation of multi-domain system designs

Towards Continuous Verification and Validation of Multi-Domain System Designs

Yon Vanommeslaeghe, Bert Van Acker, Milan Cornelis, and Paul De Meulenaere
Cosys-Lab (FTI)
University of Antwerp
AnSyMo/Cosys
Flanders Make
Antwerp, Belgium
{yon.vanommeslaeghe, bert.vanacker, milan.cornelis, paul.demeulenaere}@uantwerpen.be

***Abstract*—The design and development of cyber-physical systems (CPS) inherently involves multiple engineering domains. As these systems become more complex, the different domains involved in their design become intertwined. In such situations, insufficient knowledge can lead to costly inconsistency and integration problems. In previous work, we proposed the use of a cross-domain knowledge model (CDKM) to capture these dependencies to support the architectural and detailed design stages of the development process. In this paper, we present early-stage research aiming to extend this approach to also support the verification and validation (V&V) stages, with the goal of enabling continuous V&V. To this end, we propose a megamodeling approach to provide traceability between different models, using the CDKM to provide a system-level overview of the CPS under development.**

***Index Terms*—Cyber-Physical Systems, Model-Based Systems Engineering, Co-Design, Continuous Verification and Validation, Megamodeling**

I. Introduction

The design and development of cyber-physical systems (CPS) inherently involves multiple engineering domains. For example, the design of a drone may involve a mechanical engineer to design the frame, an electromechanical engineer to design the propulsion system, a control engineer to develop the control algorithms, and an embedded engineer to deploy these algorithms. To improve the efficiency of the design process, engineers from different domains often work in parallel, in what is referred to as a co-design process. However, as these CPS become more complex, the different engineering domains become increasingly intertwined, with design decisions made in one domain significantly affecting another. In these cases, traditional co-design approaches can be inefficient or even fail in finding good overall designs. Indeed, insufficient knowledge about these cross-domain dependencies often leads to inconsistencies and integration problems, which often require costly re-engineering to fix.

In previous work [1], we presented an approach which uses ontologies to assist the co-design process by explicitly capturing these dependencies, both within and across engineering domains. As such, this provides a system-level overview of the system under design. We refer to this as a cross-domain knowledge model (CDKM). Figure 1 shows an excerpt of such a CKDM for the design of a drone. This model consists of two main parts, a *property ontology*, and a *dependency model*.

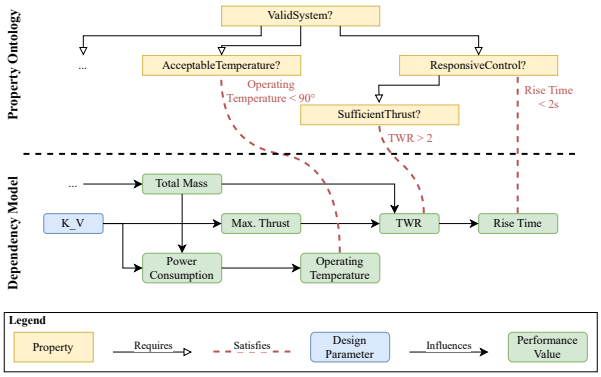

Fig. 1. Excerpt of a cross-domain knowledge model for the design of a drone.

In short, the *property ontology* mainly supports the system requirements and architectural design phase of the development process. It contains properties relating to requirements of the system under design. For example, the property *AcceptableTemperature?* relates to a requirement stating *"The temperature of the system shall not exceed 90 °C during operation"*. For properties to be considered *True* (satisfied), they may *Require* other properties to be *True* as well. The *dependency model* then supports the detailed design of the system by capturing design parameters (DP) and performance values (PV) relating to system components, and how these influence each other. For example, the velocity constant of the motors used for propulsion (*K_V*) is a design parameter, which affects, a.o., the *Power Consumption* and subsequently the *Operating Temperature* of the system. Additionally, the two parts of the CDKM are linked using *Satisfies* relationships. These relationships are used to define when a property is considered *satisfied* based on one or more performance values. For example, *AcceptableTemperature?* is considered satisfied if the PV *operating temperature* is less than 90 °C, as stated in the previously mentioned requirement. For a more detailed explanation of these concepts, the reader is referred to previous work [1].

While this approach succeeds in providing a system-level overview, it necessarily does so at a high level of abstraction. The dependency model shows which design parameters and performance values affect which other performance values, but it cannot necessarily provide an exact value for a PV based on the value of an influencing DP or PV. Consequently, this alone is not enough to determine whether the properties, and thus requirements, are satisfied or not. Indeed, during the development process, engineers will perform different evaluations, using different test scenarios, to determine the exact value of these PVs, often using modeling and (co-)simulation to run experiments in-silico, to determine if requirements are met. As such, while the CDKM as previously presented can support the architectural en detailed design stages of a development process, it does not yet explicitly support the verification and validation stages. In the current paper, we present early-stage research to extend on the CDKM to support this.

We propose to explicitly model these test scenarios, explicitly linking test parameters and results to design parameters and performance values in the dependency model. Moreover, we propose a megamodeling approach to explicitly capture the dependencies between these tests, the (co-)simulation models used therein, the functional mockup units (FMUs) these contain, and the (domain-specific) models from which these FMUs have been generated to provide traceability. The ultimate goal of this is to enable continuous testing in the context of co-design processes, using the CDKM to maintain a system-level overview of the overall status of the design.

II. Related Work

Larsen et al. [2] recognize the challenges associated with CPS design due to their heterogeneous nature. They state that the required dependability of CPS means that there is a need for well-founded validation and verification (V&V) techniques. However, combining the diverse modeling paradigms used during CPS design is challenging in this regard. In the INTO-CPS project, they developed a toolchain to support the multidisciplinary, collaborative modeling of CPS. Here, they rely on the functional mock-up interface (FMI) standard [3] to enable the integration of domain-specific models through co-simulation. In the current approach, we rely on the concepts proposed in the INTO-CPS project to define co-simulation models comprising domain-specific models of the CPS under design.

Van Acker et al. [4] presented the Validity Frame (VF) structure to (i) capture the range-of-validity of a model and (ii) provide methods/processes to assure that a model faithfully represents the source system. In [5], this VF concept was used to reduce the effort (and cost) of the V&V phase of the engineering process of complex CPSs, this by enhancing the knowledge about the system components. It provides valuable insights in how system properties are explicitly linked to the model and how they are evaluated by means of validation experiments. However, the focus of VFs is solely on the validity evaluation of a single model, not on a model composition.

In systems engineering, engineers often rely on textual requirements to drive the integration verification and validation (IVV) of the system, with manually created traceability links between requirements and associated test cases [6]. However, Voirin et al. [6] recognize that, as the complexity increases, this textual requirements-based approach reaches its limits for multiple reasons: requirements are not able to formally describe the solution, traceability links are unreliable, test campaigns remain informal and thus subject to interpretation, it is difficult to identify and localize problems, etc. They state that one of the goals of model-based engineering (MBE) approaches, such as Acadia [7], is to overcome these limitations of textual requirements by formalizing them in a shareable form, complementing them with models. One major advantages of such model-based approaches is that they can provide traceability links between models. For example, in Arcadia, links between requirements and IVV tests are derived from requirements-to-models and model-to-test links, which makes them more reliable and easier to check [6]. They identify a number of benefits of this approach. First, the traceability provided by this approach allows tools to identify which tests need to be run, or not run, based on the availability of system components. Similarly, regression tests can also be optimized based on which components have changed since the previous version of the system (impact analysis). Additionally, they state that this should allow optimization of the IVV across different engineering levels, based on the organization and the links between models at different levels. However, to the best of our knowledge, in Arcadia, this IVV concerns the *correct* implementation and integration of system components, and not necessarily their *optimal* implementation. As such, to the best of our knowledge, there are no explicit links to design parameters and performance values.

Bézivin et al. [8] first describe the need for megamodels in the context of model-driven engineering. They define a megamodel as "a model of which at least some elements represent and/or refer to models or metamodels". As such, they consider megamodeling to provide a global view on different artifacts, i.e., models. Over the years, many further definitions for megamodels have appeared in literature. Hebig et al. [9] unify these different definitions by defining a megamodel as "a model that contains models and relations between them". It is this definition that we follow in the current paper. Regarding the use of megamodeling in the design of CPS, the MegaM@Rt2 project aims to create an integrated framework for continuous system engineering and runtime V&V [10]. The major challenge they aim to address is enabling traceability between design-time and runtime. They do this by providing a megamodeling approach to manage system artifacts, such as different types of models, workflows, configurations, etc., to provide a complete view of a CPS. Regarding runtime, they consider traces produced by, e.g., online monitors, which are linked to system artifacts for validation purposes with the goal of detecting possible design deviations [11]. In the current paper, we mainly focus on enabling traceability and continuous V&V at design time.

III. Proposed Approach

The proposed megamodel is illustrated in Figure 2, which provides a high-level overview of the overall approach. Currently, we consider six different types of models, shown in different sections in the figure, separated by dashed lines. Models in the different sections are linked using different types of relationships, all of which denote some form of dependency between the models. The top two sections, *Property Ontology* and *Dependency Model*, correspond to the two parts of the cross-domain knowledge model as described in the introduction. The other sections and their relationships are defined as follows:

- *Test Scenarios* These models are further detailed in Section III-A. In short, they contain sequences of different types of evaluations and related artifacts, which are designed to evaluate specific aspects of the design.
- *Multi-Models* These are (co-)simulation models as defined in the INTO-CPS project [2]. They define instances of functional mock-up units, as well as how these are connected, settable parameters with their default values, simulation setting, etc. The mentioned settable parameters are linked to parameter variables of the FMU instances.
- *Functional Mock-up Units (FMUs)* This essentially defines a library of FMUs available to the engineers. It contains information about each FMU itself, as well as its available ports, parameters, etc., obtained from the model description of the FMU.
- *Models* This is essentially an index of (possibly domain-specific) models developed by the engineers. They may represent different parts of the system, potentially implemented in different tools, such as Simulink, AMESim, Modelica, etc. Different models may also represent different aspect of the same part of the system. For example, separate electromechanical and thermal models of the propulsion system of a drone.

Additionally, we define different types of relationships (dependencies) between the different sections. Similarly, the *Requires*, *Satisfies*, and *Influences* relationships correspond to those defined for the cross-domain knowledge model as described in the introduction. The other relationships are defined as follows:

- *Input/Output (I/O) Dependency* These dependencies are used to link artifacts used in the test scenarios to specific design parameters or performance values in the dependency model. Additionally, they can be used to model dependencies between test scenarios. For example, if some output of one scenario is required as input for an other.
- *Model Dependency* These are used to define which multi-models are used in which test scenario, in the case that such a scenario involves running a (co-)simulation. Moreover, they link parameters defined in the multi-models to artifacts in a test scenario. For example, to further link a design parameter through to a parameter value defined in the multi-model.

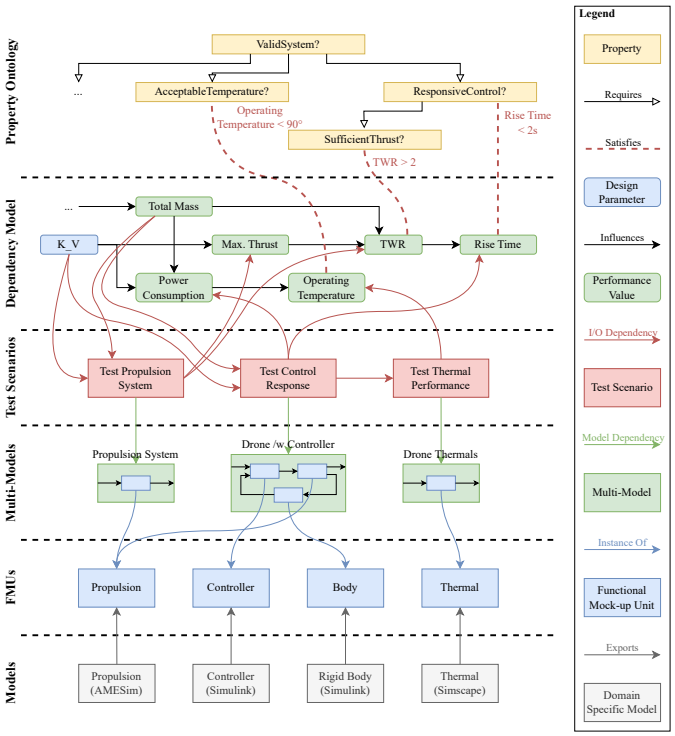

Fig. 2. An illustration of the overall proposed approach, showing the different types of models and their relationships.

- *Instance Of* This links an FMU instance defined in a multi-model to a specific functional mock-up unit.
- *Exports* This defines from which model each FMU is exported.

As can be seen in Figure 2, explicitly capturing the relationships between the different sections in this way provides traceability, all the way from the (domain-specific) models to the requirements as captured in the property ontology. In Section IV, we discuss how these explicitly modeled relationships can be leveraged to provide different benefits.

A. Test Scenarios

Figure 3 shows a simplified class diagram of a *test suite* with test *scenarios*. We define a test scenario as a sequence of *evaluations* to be performed. These evaluations can require one or more *artifacts* as input and produce one or more *artifacts* as output. Currently, we discern two different types of evaluation: *(co-)simulations* and *scripts*. Here, (co-)simulations reference specific multi-models, while scripts are mainly used to provide pre- or post-processing capabilities. For the artifacts, we also currently discern two types: *values* and *traces*. Here, values represent variables which are considered constant over the course of an evaluation, such as a test parameter. These can be linked to design parameters or performance values in the dependency model. Conversely, traces are sequences of values. We mostly use traces to define input stimuli for simulations or to capture simulation outputs over time. Performance values are often obtained from traces after post-processing. Additionally, artifacts produced in one scenario may be required as

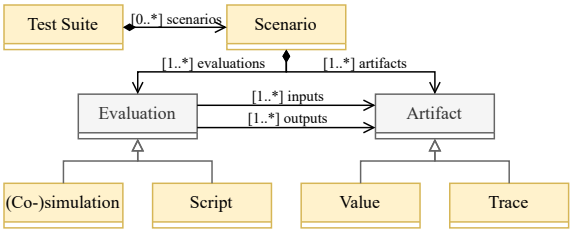

Fig. 3. Simplified class diagram of the test suite with test scenarios.

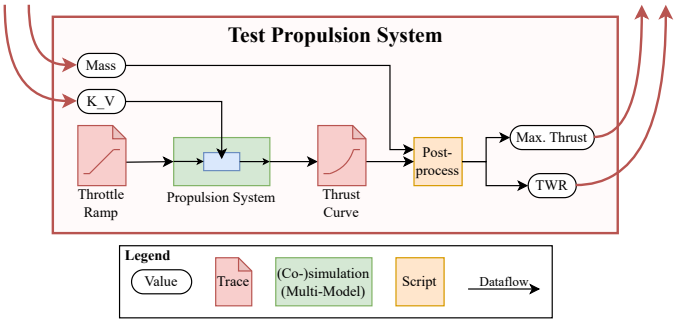

Fig. 4. Illustration of a scenario to evaluate the design of a propulsion system.

input for an other, leading to a dependency between the two (not shown).

An example of such a scenario, designed to evaluate the propulsion system of a drone, is shown in Figure 4. More specifically, to evaluate its thrust. The scenario contains two input values, a motor velocity constant for the motor used in the propulsion system (*K_V*) and the total mass of the drone (*Mass*). Additionally, it contains an input trace (*Throttle Ramp*). It also contains a (co-)simulation, which references the multi-model *"Propulsion System"*, and a post-processing script. The input trace contains a throttle curve which serves as input for the (co-)simulation, while the value *K_V* is linked to a parameter of an FMU instance in the multi-model. From the (co-)simulation, we obtain a new trace (*Thrust Curve*). This new trace, together with the value *Mass*, serves as input for the post-processing script, which calculates the values *Max. Thrust* and *TWR* (the thrust-to-weight ratio). In this case, input and output values are linked to design parameters and performance values in the dependency model.

IV. Preliminary Results & Future Work

A prototype version of the proposed approach has been implemented in the eclipse modeling framework (EMF) [12]. This covers everything from FMUs up to I/O dependencies (Figure 2), with the CDKM portion being reused from previous work [1]. With the goal of continuous testing in mind, we have already identified two major benefits of the proposed approach, similar to those identified by Voirin et al. [6]:

a) Generating Testing Workflows: First, the information contained in the megamodel allows us to determine the correct order in which to run the different tests, i.e., taking into account direct or indirect dependencies between the different scenarios. For example, given the example illustrated in Figure 2, we see that *Test Control Response* should be executed before *Test Thermal Performance*, as there is a direct dependency between the two scenarios. Additionally, as the property *ResponsiveControl?* also *requires* the property *SufficientThrust?* to be satisfied, it might be advantageous to execute *Test Propulsion System* before *Test Control Response*. However, the optimal ordering of these two test would depend on, e.g., their individual computational requirements. Moreover, the megamodel allows us to identify which actions need to be taken before a certain test can be executed. For example, before executing *Test Propulsion System*, the *FMU "Propulsion"* needs to be exported from the relevant model. As such, the presented megamodel should allow us to generate a full test workflow for a system under design, for example, captured in a process model (PM) [13]. However, it is currently considered future work to fully implement this, and to investigate which criteria could be used to generate an *optimal* workflow, i.e., taking into account the effort, computational requirements, etc., associated with executing each test.

b) Impact Analysis: The traceability provided by the explicit relationships between the different models also allows for automated impact analysis. It allows us to determine which FMUs, tests, performance values, properties, etc., are invalidated by changes to any model, design parameter, performance value, etc., thus indicating, a.o., which tests needs to be re-run for a particular change. For example, if the model *"Thermal"* has been modified, we know we need to re-export the FMU *"Thermal"*, and re-execute *Test Thermal Performance*, to ultimately re-evaluate the property *AcceptableTemperature?*. Moreover, modeling the test scenarios at a lower level of abstraction, i.e., as a series of evaluations, allows us to identify which *part(s)* of the test scenario need(s) to be re-executed. For example, if the design parameter *Total Mass* changes, the megamodel as illustrated in Figure 2, would indicate that *Test Propulsion System* needs to be re-executed. However, looking at Figure 4 shows that only *Post-process* needs to be re-executed. We believe this would allow for more efficient testing when changes are detected. However, the current implementation does not explicitly support versioning or change detection mechanisms. As such, it is currently considered future work to further develop this.

In general, the integration of the presented approach in an actual continuous integration (CI) pipeline is currently considered future work. In this case, we envision that the *exports* relationships between the models and FMUs would include build scripts to enable the automatic generation of FMUs from the relevant models.

Additionally, we envision other possible applications for the presented approach. However, these all require further research to determine their viability:

- As a first step towards validity of composition models as an extension of validity frames. This may include the introduction of validity monitors on the links between the different types of models.
- In the context of DevOps for CPS, similar to the MegaM@Rt approach [11], parts of the presented ap-

proach may be reused to determine performance values from logged data instead of from simulation traces. This should make it possible to detect when certain properties (and thus requirements) are no longer satisfied, which could mean that the system has diverged from the original models, e.g., due to wear or modifications [14]. Alternatively, this could reveal edge-cases which are not yet part of the test scenarios.

- The information captured in the megamodel, particularly the test scenarios, may also be used in the context of design space exploration (DSE) to optimize the CPS under design. In previous work [15], we already presented a method for determining DSE workflows for a given system under design, starting from, a.o., a dependency model and a library of evaluation functions. Here, the test scenarios would correspond to the evaluation functions in previous work. However, information regarding their computational requirements would need to be added to the current approach.
- Explicitly capturing the dependencies between different models, test scenarios, properties, etc., also reveals the dependencies between the different engineering teams. This information might be used to determine an efficient co-design workflow for the system under design, as it reveals where teams can work in parallel or where they need to wait for each other, e.g., because models from multiple different teams are required for a specific test, to evaluate whether a specific requirement is met.

However, while the proposed approach provides several (potential) benefits, a drawback is that it may require significant additional modeling effort during the development process. As such, it remains to be seen to what extent the benefits provided outweigh the additional effort required during development.

V. Conclusions

We presented early-stage research wherein we aim to extend on our previously developed cross-domain knowledge models (CDKMs) to also support the system verification and validation stages of the development process for cyber-physical systems. We do this by explicitly modeling test scenarios used to evaluate the performance of the system under design. Moreover, we propose a megamodeling approach, wherein we not only link these test scenarios to the CDKM, but also to co-simulation models, functional mock-up units, etc., to provide traceability from models to requirements. With the goal of continuous verification and validation in mind, we have already identified two major benefits of the proposed approach. Namely, the generation of testing workflows and automated impact analysis. However, both require further research to fully implement them and to integrate this into an actual continuous integration pipeline. Additionally, we have identified further possible applications of the proposed approach in the context of validity frames, DevOps, design space exploration, and co-design workflows. However, these all require further research to determine their viability.

Acknowledgment

This research was supported by Flanders Make, the strategic research center for the manufacturing industry in Belgium, within the Flexible Multi-Domain Design for Mechatronic Systems (FlexMoSys) project.

References

[1] M. Cornelis, Y. Vanommeslaeghe, B. Van Acker, and P. De Meulenaere, "An ontology dsl for the co-design of mechatronic systems," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 633–642.

[2] P. G. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis *et al.*, "Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project," in *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. IEEE, 2016, pp. 1–6.

[3] A. Junghanns, C. Gomes, C. Schulze, K. Schuch, R. Pierre, M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer *et al.*, "The functional mock-up interface 3.0-new features enabling new applications," in *Modelica conferences*, 2021, pp. 17–26.

[4] B. Van Acker, P. De Meulenaere, J. Denil, Y. Durodie, A. Van Bellinghen, and K. Vanstechelman, "Valid (re-)use of models-of-the-physics in cyber-physical systems using validity frames," in *2019 Spring Simulation Conference (SpringSim)*, 2019, pp. 1–12.

[5] B. Van Acker, B. J. Oakes, M. Moradi, P. Demeulenaere, and J. Denil, "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*, 2020, pp. 1–10.

[6] J.-L. Voirin, S. Bonnet, V. Normand, and D. Exertier, "Model-driven ivv management with arcadia and capella," in *Complex Systems Design & Management: Proceedings of the Sixth International Conference on Complex Systems Design & Management, CSD&M 2015*. Springer, 2016, pp. 83–94.

[7] P. Roques, "MBSE with the ARCADIA Method and the Capella Tool," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, Jan. 2016. [Online]. Available: https://hal.science/hal-01258014

[8] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *proceedings of the OOPSLA/GPCE: best practices for model-driven software development workshop, 19th Annual ACM conference on object-oriented programming, systems, languages, and applications*. Citeseer, 2004, pp. 1–9.

[9] R. Hebig, A. Seibel, and H. Giese, "On the unification of megamodels," *Electronic Communications of the EASST*, vol. 42, 2012.

[10] J. G. Cruz, A. Sadovykh, D. Truscan, H. Bruneliere, P. Pierini, and L. L. Muñiz, "MegaM@Rt2 EU project: Open source tools for megamodelling at runtime of CPSs," in *Open Source Systems: 16th IFIP WG 2.13 International Conference, OSS 2020, Innopolis, Russia, May 12–14, 2020, Proceedings 16*. Springer, 2020, pp. 183–189.

[11] H. Bruneliere, S. Mazzini, and A. Sadovykh, "The MegaM@Rt2 approach and tool set," in *DeCPS Workshop, 22nd International Conference on Reliable Software Technologies-Ada-Europe 2017*, 2017.

[12] Eclipse Foundation. (2023) Eclipse Modeling Framework (EMF). [Online]. Available: https://www.eclipse.org/modeling/emf/

[13] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukss, "FTG+PM: An integrated framework for investigating model transformation chains," in *International SDL Forum*. Springer, 2013, pp. 182–202.

[14] J. Mertens and J. Denil, "The digital twin as a common knowledge base in devops to support continuous system evolution," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2021, pp. 158–170.

[15] Y. Vanommeslaeghe, J. Denil, B. Van Acker, and P. De Meulenaere, "Automatic generation of workflows for efficient design space exploration for cyber-physical systems," in *2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2021, pp. 346–351.