A Graph-Based Meta-Data Model for DevOps: Extensions to SSP and SysML2 and a Review on the DCP Standard

Stefan H. Reiterer[1] Clemens Schiffer[1] Mario Schwaiger[1]

[1]Department E, Virtual Vehicle Research,
{stefan.reiterer,clemens.schiffer,mario.schwaiger}@v2c2.at

Abstract

Computer simulation has become a vital tool for modeling complex systems. However, the development and deployment of simulation models often involve multiple stages, tools, and teams, which can lead to significant challenges in maintaining quality, reliability, and efficiency. DevOps, a set of practices that combines software development and IT operations, has emerged as a promising approach to streamline the simulation development. Although most system engineers are not DevOps specialists and there are a lot of manual steps involved when writing build pipelines and configurations of simulations. For this purpose, an abstract graph-based meta-data model was presented in Stefan H. Reiterer, Balci, et al. (2020) to provide an automation framework for DevOps with simulations (see also Stefan H Reiterer, Schiffer, and Benedikt (2022)). In this work we want to continue our investigations by expanding and harmonizing this approach to better work with established standards like SSP, SysML2 and DCP and demonstrating its application on real-life use cases.

Keywords: Continuous Integration, DevOps, MBSE, NoSQL Graph databases, DCP, SysML, UML, SSP

1 Introduction

DevOps is a set of practices, cultural values, and tools that aim to improve collaboration and automation between software development and IT operations teams, with the goal of delivering high-quality software products and services more efficiently and reliably. This approach emphasizes the integration of development, testing, deployment, and monitoring processes to enable faster and more frequent software releases, while maintaining stability and reliability.

Formally Bass, Weber, and Zhu (2015) introduced DevOps as a set of practices intended to reduce the time between committing a change to a system and the change being placed into production while ensuring high quality.

DevOps involves a range of practices, such as continuous integration and continuous delivery (CI/CD), infrastructure as code (IaC), automated testing and monitoring, and often includes agile development methodologies. It also emphasizes the importance of communication, collaboration and shared responsibility between Development and Operations teams and the use of the continuous improvement processes to assess quality and outcomes.

The benefits of DevOps include improved software quality and reliability, faster time-to-market, increased efficiency and productivity, enhanced flexibility and scalability, and better collaboration and communication between teams. It is increasingly being adopted by organizations across a wide range of industries, from startups to large enterprises, as a key enabler of digital transformation and innovation.

With the need to accelerate development cycles in other domains as well like Advanced Driver-Assistance Systems (ADAS) or mechatronics it is crucial to carry over DevOps practices to computer simulations as well. However, there are several difficulties arising when transferring these methods from pure software environments into the world of computer aided engineering (CAE).

The first difficulty arising is that most people working in engineering and scientific fields are not software engineers. This means it is important to democratize DevOps practices with several tools which allow to easily implement, abstract, and reuse the setup of build pipelines to enable better automatic testing. The next problem that arises is the simulation of specific needs, especially when dealing with simulation coupling from different domains. Furthermore, testing and evaluating the simulation quality is much more difficult than regular software applications due to norms and safety requirements which often are physical in nature. Although, there are assessment processes for dealing with those issues (see e.g. the UPSIM project described in **ahmann2022towards**) it is still necessary to seamlessly integrate proper tooling and methodology into the DevOps cycle for simulations.

For that matter a graph-based meta-data model was developed in order to provide a data structure which easily can be stored into modern database systems and is also able to represent dependencies, the topology of co-simulations and is able to be easily mapped from and onto pre-existing standards. In Stefan H Reiterer, Schiffer, and Benedikt (2022) an overview of the topic is given. Additionally, it must be abstract enough to describe a whole range of use cases, but still concrete enough to generate process descriptions out of it to make it accessible for automation.

In this work we will further investigate how to properly leverage the graph-based approach to harmonize it with established standards and will demonstrate its viability on a real-life use case. Furthermore, we will analyze potential shortcomings of the current state of affairs and will discuss potential extensions of the DCP standard to come by with these limitations.

For that matter we first start with a description of the goal we want to achieve with a specific ADAS use case as motivation, following a description of the established standards. We will then continue with a short summary of our graph-based approach and how to create mappings from high level system descriptions (e.g. SysML 2) to more concrete simulation descriptions (e.g. SSP) and how those are rolled out and integrated. In the end we will have a closer look at the DCP standard and what tools and extensions could be helpful in the future to support the proposed workflow.

2 Motivation and ADAS Use Case

In the development of complex systems there is a huge gap between the systems engineering point of view, the practical implementation of software, setup of simulations (Developers) and setting up the of the infrastructure (DevOps) as those require very different sets of skills. Especially, DevOps as the third pillar becomes much harder to perform with raising complexity as the automation pipelines need constant maintenance by experts. Expert knowledge of networking and understanding of containerization and virtualization technologies like Docker or Podman and services like Kubernetes. For that reason, it is important to provide abstraction between those layers to be able to seamlessly transfer the information from one end to the other without being confronted with to much detail. In order to lay out our approach we will describe in this section an example and how its automation will be handled.

2.1 ADAS Use Case Description

In this simple scenario we have 3 roles. The first role is a systems engineer which provides us with a system description and requirements provided in SysML 2.0 (see Figure 2). As the systems engineer does not know all aspects of the simulation the simulation engineer has to set up the simulation from the description and has to come up with a script to compute the desired Key Performance Indicator (KPI). Finally in order to improve on the workflow and raise re-usability the DevOps engineer has to automate the necessary steps to run (and potentially re-run) the simulation under different parameters and provide a pipeline which starts the simulation.

The ADAS use case consists of the following models: A simple vehicle dynamics written in C, a scenario player (EsMini) an FMU with an sensor perception, a simple ADAS function (in this case an FMU with an ACC implemented in C) a transform block for signal mappings. See Figure 1. Furthermore, a post processing script written in Python was used to analyse the driving comfort where we used the methodology described in de Winkel et al. (2023) to compute a suitable key performance indicator.

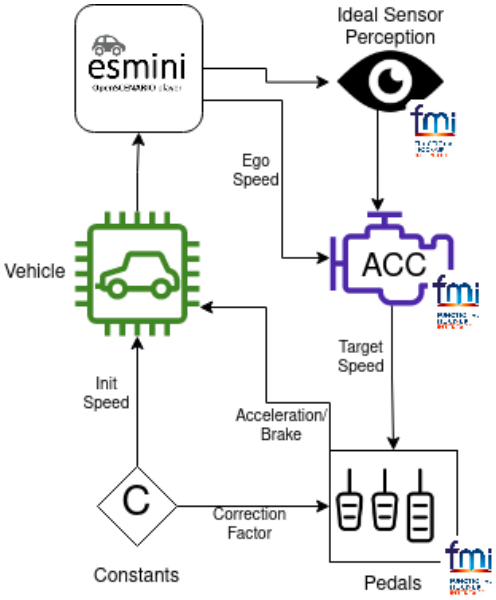

Figure 1. ADAS simulation architecture

2.2 Challenges

However, considering the complexity which arises with variations of the models when changing parameters or adding or removing models of the (co-)simulation this adds a layer of additional complexity to the problem which should not be underestimated. Even small changes to the model can be tedious to apply if they occur often. Hence, it is of utmost importance that these procedures are automated and made traceable as one could easily lose the oversight of the many steps that were taken.

In order to ease the load of the developer it is beneficial that the changes on parameters and architecture could be automatically mapped onto the different models so that the simulation engineers and DevOps engineers can focus on their main tasks. In the next sections we will discuss the standards and concepts which will be used to achieve that goal and how a potential implementation looks like.

3 Established Standards

We use this section to shortly introduce some of the more common standards we want to look at to tackle the arising challenges described in the previous section. We chose these standards due to their open nature and availability, which enables a broad spectrum of use cases. We also will have a short look at the not yet released SysML2.0 standard for system modeling, which is a successor to the widely accepted SysML standard.

3.1 SSP Standard

The Modelica SSP (System Structure and Parameterization) standard described in Hällqvist et al. (2021) is a comprehensive framework for developing cyber-physical sys-

DOI 10.3384/ecp204159

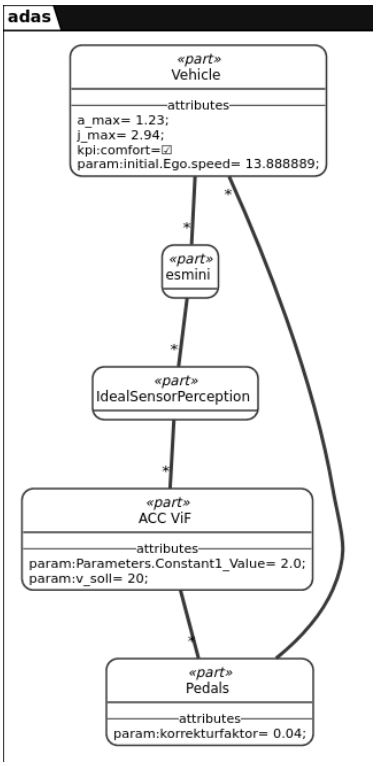

Figure 2. SysML2 model description with requirement

tems that enables the modeling and simulation of complex systems across various domains, including automotive, aerospace, and energy systems. It was designed to be compatible with major Modelica standards like Functional Mock-up Interface (FMI) and is based on the XML standard. The SSP standard provides a systematic approach to structuring and parameterizing system models, which facilitates model exchange and reuse, enhances interoperability, and enables the development of more accurate and efficient simulations.

The SSP standard includes a set of guidelines and conventions for modeling the structural and physical aspects of systems, such as components, connections, and parameterization. The standard also provides a well-defined syntax and semantics for describing the behavior and interactions of system components, which allows for the development of executable models that can be simulated using a range of simulation tools. In addition, it supports the integration of models with other software tools and platforms, such as control systems and optimization tools.

3.2 The SysML Standard 2.0

The widely used System Modeling Language (SysML) is a general-purpose modeling language for developing complex systems and SysML 2.0 is the latest version (which is under current development see OMG (2023)). It is designed to support model-based systems engineering (MBSE) and provides an integrated set of modeling concepts, notation, and semantics that are optimized for the specification, analysis, design, verification, and validation of complex systems. While its predecessor SysML is an extension of the Unified Modeling Language (UML), SysML 2.0 is based on the KerML metamodel.

The main objective of SysML 2.0 is to provide a comprehensive language for MBSE, which can be used throughout the entire system development life cycle. The language provides a standardized way of representing different aspects of the system, including its structure, behavior, requirements, constraints, and interfaces. Furthermore, support for the integration of different domains and perspectives, such as mechanical, electrical, and software, in a single model is provided.

The standard is open and is developed and maintained by the Object Management Group (OMG) with input from a wide range of stakeholders, including industry experts, academics, and users. The language is supported by a growing ecosystem of tools and frameworks, which enable users to create, analyze, and manage SysML models efficiently, e.g. plugins for Eclipse.

The standard can be used by a system engineer to represent the system in concise manner which can be used to generate graphical representations of the system (see Figure 2 for an example). Furthermore, this can be used to describe dependencies between different components of the system and the respective requirements which in return can be leveraged to extract the system architecture and its signal flows like we demonstrate in Section 5.1.

4 The Co-Simulation Process Graph

The Co-Simulation Process Graph concept was originally introduced in Stefan H. Reiterer, Balci, et al. (2020) and is an extension of the classical Process Graph Concept Tick (2007) which allows to not only map process steps (for e.g. a build and deploy pipeline) but also to map the structure of a co-simulation with inputs and outputs and necessary information of the setup steps. In this section we will give a brief introduction of the concept for reader which are unfamiliar with it and also will discuss methodologies to version changes of the Co-Simulation Process Graph which is important for many applications in engineering as traceability is a hard requirement in that sector.

4.1 Definition and an Example

The main problem when trying to map simulation configurations within a process graph is that the moment closed loop simulations are included this introduces cycles within the graph structure. However, this violates the main condition to compute execution orders namely that a process graph is cycle free. In order to solve the problem of cycles introduced by closed loop simulations and models without the need of separating the workflow sequence and the topology of the simulations the Co-Simulation Process is defined with the following properties:

- The set of nodes consists of data nodes, transformation nodes, master nodes, signal nodes and communication (or gateway) nodes.

- To represent the instantiation of a process or the usage of a signal inside a simulation, copies of the nodes which represent these instances are made. Instances must be directly connected to their originals.

- Instead of using the bi-partite structure to represent data transformations, only instances of processes can connect to data nodes to perform operations. In this way, the nodes which perform operations and their instantiation can be determined with a suitable algorithm, which determines a different partition of the graph with help of the defined structure, to provide the correct order of executions. This is necessary since it is allowed that transformation nodes are neighboring, e.g., a Docker container which is built and then used for executing a program.

- An information node can never be the successor or predecessor of another information node. A process must be placed in between. However, neighboring process nodes are allowed. This may happen if a program-performing transformation at a later stage is modified beforehand by another process (e.g., parameterization of tools).

- A simulation is a sub-graph with the following properties: a) It contains the instance of a master node. b) The instance of the master node is connected to all instances of signal nodes that belong to the simulation. c) All the other nodes inside the simulation (i.e., the simulation participants and communication gateways) neighbor a signal instance. d) Each instance of a signal is only allowed to appear once inside a simulation.

- Cycles are only allowed inside a simulation sub-graph.

A more detailed description of the data structure and analysis of the used algorithms can be found in Stefan H. Reiterer and Kalab (2021), which was recently accepted in the International Journal of Simulation and Process Modelling. An example is shown in Figure 3. In this example the nodes (of type **Node**) c_1 and c_2 represent software sources (e.g., source code of a model) b represents a build tool like CMAKE and b_1 and b_2 (of type **Bridge**) represent two processes of this build tool which are started, which leads to the simulation units P_1 and P_2 (nodes of type **Bridge**) while the node M represents a simulation master (a node of type **Master/Bridge**). After the build in stage 1) the simulation is executed and the master is configured by the information contained in the node M and gets additional parameters from node I, while the node O represents the output of the simulation. The **Signal** nodes i_j and o_j represent in- and outgoing signals like velocity or acceleration, while **Gateway** node g_j represents the communication protocols (e.g., a network protocol like IP) for $j = 1, 2$.

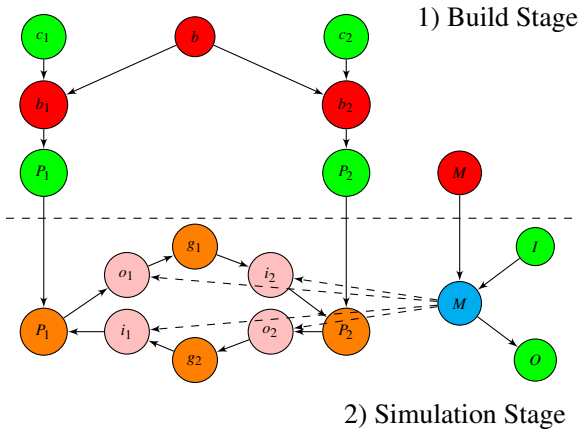

Figure 3. Simple example of a co-simulation process graph

4.2 Versioning Aspects

Graph Databases are to some degree able to use versioning, however this feature is in general not implemented (ArangoDB 2023). In stark contrast to relational databases, where several add-ons exist. Oracle Flashback or Postgres Time Travel are two notable examples. As the structure of a graph differs from the relational model a different approach has to be used. It is also required to take into consideration that not every backup-and-storage technology might be fitting to solve the challenge of versioning. The database used in this work is ArangoDB as it allows handling data more flexibly than its competitors.

When backup is discussed, there are mainly three strategies: Full vs. Incremental vs. Differential. The use case that we consider should cover the following aspects when handling the versioning of graphs which undergo many incremental changes:

- The current/latest graph has to be loaded the fastest as it is used in production.

- Fast loading of the previous versions which were recently added.

- Storage-benefits over full-backup as a lot of redundant data is created.

This is motivated by the fact that a model will undergo a lot of small incremental changes during the development process and hence will create a lot of redundant data. With regards to the solution is an inverse differential storage which has the latest version of the graph saved in an unmodified state. The previous versions only save the data changed over each iteration. For the sake of simplicity, currently this is done by using a signal-character which marks the unchanged data. One of the benefits of this model is the hybrid approach that still allows to implement a partial-full model to jump back to any given fully backed up version.

Almost any given record can be persisted in a collection of the database. After adding another collection of a different version of the data the versioning starts. All the fields are checked whether or not they are equal to the

DOI 10.3384/ecp204159

previous version. In case of unchanged data, the previous records will be overwritten by the signal character. The differing fields remain unchanged. By doing this version 1 and version 2 are obtained. On any other given data set the same algorithm is applied to create the next iteration.

In order to return back to a previous version either a full-backup-milestone or the most recent version are used to start from. All the signal-characters are reverted to the latest state. A prototype is currently under development in the scope of a bachelor thesis at Virtual Vehicle Research. First tests with a simple co-simulation process graph which underwent some changes over time already showed roughly a memory saving between 20-30% in contrast to full back ups of the different versions. See Table 1 for an overview of different cases and the saves for the file sizes. However, there is a lot of potential for optimization

Table 1. Difference in file sizes (FS) given in bytes

Records	FS Orig.	FS Compressed	Saving [%]
10	7094	5348	24.6
50	35156	24942	29.1
100	73455	49377	32.8
123	52964	43360	18.1
125	53822	44010	18.2
250	108036	88184	18.4
500	215884	176470	18.3
750	397946	315564	20.7
1000	668678	425724	36.3
2500	1394335	981182	29.6
5000	4168767	2938219	29.5
10000	15442067	12299693	20.4

which will be explored further in the future.

5 Mappings between the Graph and the Standards

In this section we will discuss shortly the mappings between the Co-Simulation Process Graph and the SysML 2.0 and SSP standard and their connection to the graph database.

5.1 Mappings between the Graph Database and the SysML 2.0 Standard

Since the SysML 2.0 standard is not finalized yet tools are not completely available yet. Since the new standard is based on the KernML meta language and not XML like its predecessor it was necessary to write a simple Python parser which parses the SysML file. We used the freely available Pyparsing module for this task. As example we use the simple SysML 2.0 model described in Listing 1 which refers to the model depicted in Figure 2 and is linked to the use case we described in Section 2.1. To modularize the SysML 2.0 document it is hierarchically stored into the graph database (where the hierarchy is with respect to nesting of the KernML blocks) with dependencies within the hierarchy stored as edges. See Figure 4 for the structure in the database.

Listing 1. "SysML 2.0. model"

```
package adas
{
  part Vehicle {
    attribute 'param:initial.Ego.speed'
        =13.888889;
    attribute 'kpi:comfort';
    attribute 'j_max'=2.94;
    attribute 'a_max'=1.23;
  }
  part esmini {
  }
  part IdealSensorPerception{
  }
  part 'ACC ViF' {
    attribute 'param:Parameters.
        Constant1_Value'=2.0;
    attribute 'param:v_soll'=20;
  }
  part Pedals {
    attribute 'param:korrekturfaktor'=0.04;
  }
  connect Vehicle to esmini;
  connect Pedals to Vehicle;
  connect esmini to IdealSensorPerception;
  connect IdealSensorPerception to 'ACC ViF
      ';
  connect 'ACC ViF' to Pedals;
}
```

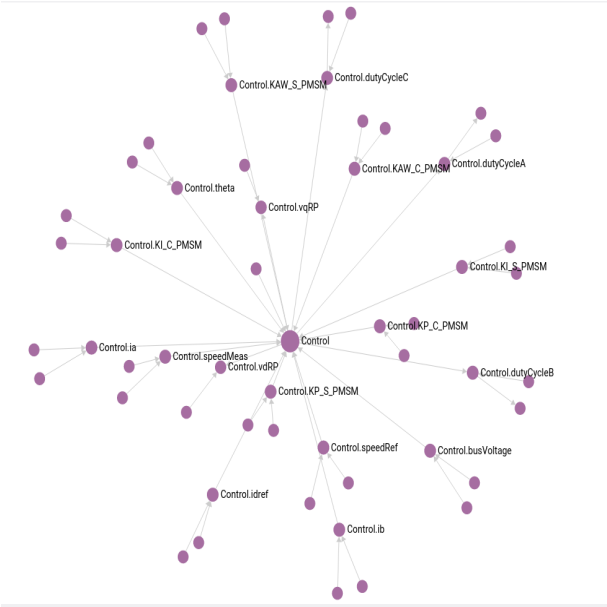

Figure 4. SysML2.0 model stored within the graph database

With help of this representation, we are able to extract building blocks and modules. We are also able to easily search and extract information by recursive search methods. We can use this to inject or update information of the co-simulation process graph which is used for setting

up, configuration and start of the test simulation. The co-simulation process graph for this model can be seen in Figure 5. It should be noted that in the graph the simulation with all its sub-nodes (participants, signals, and communication) is contracted within the "Simulation" node for the sake of simplicity. With help of the Arango Query

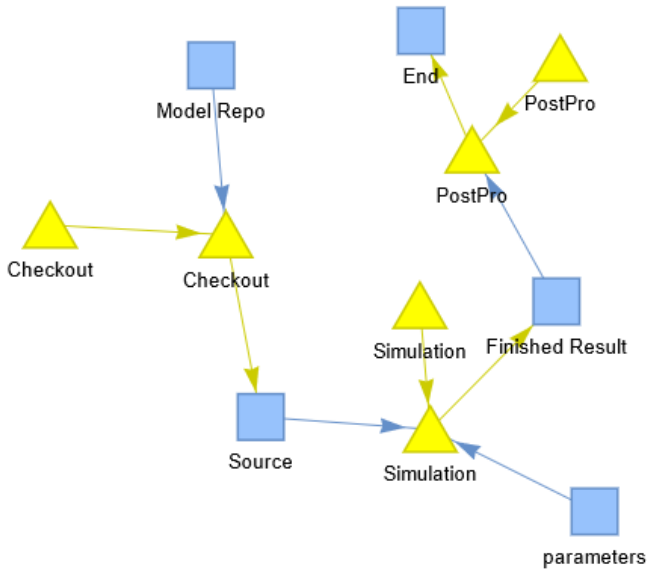

Figure 5. Co-Simulation process graph of the simulation model

Language (AQL) it is then possible to locate possible parameter changes within the SysML 2.0 model and map it onto the graph. After the computation is finished and the post-processing tools evaluated KPIs and performed quality checks the information can then stored back into the graph database. With an inverse mapping it can even be written onto the abstract SysML 2.0 model as information for the system developer. While the transfer of parameter changes in this example is rather trivial, we already demonstrated in Stefan H Reiterer, Schiffer, and Benedikt (2022) that also changes of participants, simulation settings or even the topology of the co-simulation is easily possible. We will also discuss in the next section how this can directly applied to SSP files.

5.2 Mappings between the Graph Database and the SSP 2.0 Standard

Since the SSP standard is designed to represent a co-simulation graph its mapping into the co-simulation process graph and the graph database is rather straight forward. We use the following transformation:

- The System itself and its parameters can be represented by the **Master** node of the co-simulation process graph, i.e., the **SystemStructureDescription** and its meta-data can be directly written into the (JSON-)dictionary representing the master node. Information like `ssd:DefaultExperiment` with start and stopping time directly go there.
- **Components** and their meta-data are directly mapped onto **Bridges** representing the simulation participants.
- The **Connectors** of the **Components** and their respective meta-data are mapped onto **Signal** nodes. The **kind** parameter which denotes if it is an input or output connector is indirectly mapped by the direction of the edge of between the participant and the signal node.
- Moreover, the metadata of **Connections** is directly mapped onto (communication) **Gateway** nodes, while the direction of the connection (**startConnector** to **endConnector**) is represented by the edges between the signals and the gateway connections. It should be noted that the keys **startElement** and **endElement** of the **Connections** is implicitly provided by the connectivity of the component, signal and gateway nodes and thus has not to be explicitly stored.
- Last but not least, necessary edges can be added afterwards as well,

With these mapping rules the inverse mapping is also rather clear. It may be necessary to use a tool for the geometry information if the graph comes from a different source than an SSP file, but such tools are vastly available. The mappings could be done either directly in XST or any programming language like Python. In Figure 6 we see the graph database view of the transformed SSP file representing the simulation architecture in Figure 1. Note that a lot of signals were filtered out in the view for better clarity in the representation as the simulation has a lot of signals which were not actively used during the simulation. Furthermore, it should be noted that the process graph is the master model in this scenario which collects the key information that is necessary to let the simulation run. It only has to deal with a portion of the SysML 2.0 description as the SysML 2.0 model may contain information which are not relevant for the simulation run or the DevOps processes (e.g. business relevant information), although, the graph based model is flexible enough to store information outside the scope of automation as well. On the other hand the graph model stores information about the setup of the simulation, hence, it contains more information than a regular SSP file. This means it may be necessary to enrich an extracted graph from an SSP file with additional information, except the simulation only consists of FMUs and the FMU master is predefined.

6 The Role of the DCP Standard in the Workflow and How to Enable Broader Adoption

The Distributed Co-simulation Protocol (DCP) is a Modelica standard (Modelica Association Project DCP 2019)

DOI 10.3384/ecp204159

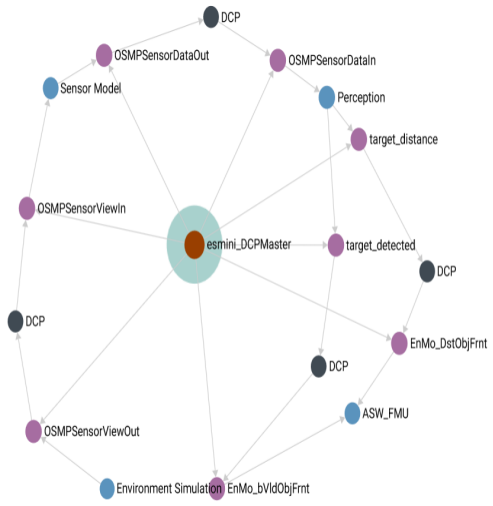

Figure 6. Graph database view

for real-time and non-real time system integration and simulation. It aims to augment the Modelica eco-system of FMI and SSP by adding distributed inter-operable simulation units, thus enabling the simulation of cyber-physical systems (CPS). This standardization is achieved by defining a common state machine and configuration together with the use of a transport protocol, either UDP, TCP are implemented in the reference implementation, while Bluetooth and CAN are specified but not yet implemented. The capabilities of a simulation unit (a DCP slave) are described in an XML-document known as a slave-description, which is intended to be shared with a DCP master before the simulation. The DCP master can then integrate a simulation scenario by configuring the simulation units according to their capabilities. The exchange of simulation data itself is achieved by protocol data units (PDUs) that are sent in a defined manner according to the transport protocol used. It aims to close the gap between software in the loop (SIL) and hardware in the loop (HIL) as DCP allows for a drop-in replacement of each component. The industry need for distributed simulation and standardized interfaces is in part covered by OPC UA (Schwarz and Börcsök 2013), ROS2 (Macenski et al. 2022) and similar technologies, however, only DCP specifically focuses on co-simulation. Recent use cases of DCP include (Rautenberg et al. 2023) which provides valuable input for possible extensions and use of DCP on coupled hardware test benches, while Segura, Poggi, and Barcena (2023) describe a generic interface using DCP in Simulink. Generally, in many applications where a distributed simulation is needed, it is implemented by either using proprietary technology or established standards with a different focus. Having a simplified – yet standard compliant – version of DCP available, such that developers only need to implement a minimal set of features, e.g., sending data via a TCP/IP port, would help in establishing DCP as a widely adopted protocol.

We identified DCP as a core technology for the proposed workflow as DCP provides us with a proper co-simulation standard for which configurations can be auto generated. While FMI has become the de-facto standard for the integration of co-simulation units and SSP for the description of systems of FMUs, there remain some issues: Either code for FMUs is generated resulting in a static artifact or the FMU has dependencies – such as installed programs, libraries or licenses. In the future we expect the development of the concept of on-line simulation platforms that enables the direct coupling of models using DCP without sharing the underlying model, as is partly discussed in Ahmann et al. (2022). The model needs to be available for a standardized distributed system simulation with a description available beforehand and the ability to be started remotely, which is essential for easy deployment on a big pool of workers either in the cloud or on premise.

However, during our work the high complexity of the DCP standard became more visible. While it is desirable that the standard covers a lot of use cases the vast range of options can become a hindrance when we want to provide basic tooling. Thus, we propose the idea of a reduced DCP core standard, which is able to cover most use cases, but enables the creation of easy-to-use tooling based on this minimal viable set of rules to accelerate the distribution of DCP. While we already proposed the use of an FMI to DCP wrapper to leverage the broad availability of FMI in our last work, we observed several times that packing FMUs can confront developers with several challenges to pack third party tools like open-source driving simulations such as esMini or Carla. A minimal standard could help in developing simple deployable DCP nodes but also a simplified master which covers a lot of use cases.

Ideally, the build and dependency of the model should be made explicit to allow for traceability as well as the ability to trigger a build to use the most recent version. This also would benefit the proposed workflow. The system description needs to be formulated in a standardized way, that allows for the description of the system architecture as well as the integration of the co-simulation system.

7 Summary

We have extended the methodology outlined in Stefan H Reiterer, Schiffer, and Benedikt (2022) how to make use of graph-based automation using dynamically generated build pipelines for co-Simulations by making use of mappings between standards for system description (SysML2.0) and system structure description (SSP) which can be used to configure a co-simulation master with a more practical example. Additionally, we demonstrated how to decompose SysML2.0 and SSP descriptions to properly store them into graph databases and how to map them properly into co-simulation process graphs to enable a more seamless workflow and proposed a method for graph versioning tailored for development workflows. Furthermore, we identified some shortcomings of the cur-

rent state DCP standard and proposed a potential solution in the form of a DCP core standard to address these issues.

8 Outlook

While the proposed graph-based methodology already addresses several issues like making standardized formats available on graph databases and some methods for versioning them were discussed, there is still a lot of potential to improve on the existing algorithms and how to better organize the pool of data which is created. Further, we have to explore the potential of data driven testing and validating the running simulations with help of the generated data over time to foster a more automated continuous improvement process over longer development periods.

Furthermore, the proposal of a core DCP standard for easier tooling has to be explored and properly formulated and activities regarding discussions with partners from academia and industry have to be initiated.

Acknowledgements

This publication was written at Virtual Vehicle Research GmbH in Graz, Austria. The authors would like to acknowledge the financial support within the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Labour and Economy (BMAW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorized for the program management. They would furthermore like to express their thanks to Prof. Eugen Brenner and Georg Macher from the Institute of Industrial Informatics at TU Graz for their support.

References

Ahmann, Maurizio et al. (2022-11). "Towards Continuous Simulation Credibility Assessment". In: *Modelica Conferences*, pp. 171–182. DOI: 10.3384/ecp193171.

ArangoDB (2023). *Data Modeling and Operational Factors*. URL: https://www.arangodb.com/docs/stable/data-modeling-operational-factors.html (visited on 2023-05-11).

Bass, Len, Ingo Weber, and Liming Zhu (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.

de Winkel, Ksander N. et al. (2023). "Standards for passenger comfort in automated vehicles: Acceleration and jerk". In: *Applied Ergonomics* 106. DOI: 10.1016/j.apergo.2022.103881.

Hällqvist, Robert et al. (2021). "Engineering domain interoperability using the system structure and parameterization (SSP) standard". In: *Modelica Conferences*, pp. 37–48. DOI: 10.3384/ecp2118137.

Macenski, Steven et al. (2022). "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66. DOI: 10.1126/scirobotics.abm6074.

Modelica Association Project DCP (2019). *DCP Specification Document, Version 1.0*. Linköping, Sweden: Modelica Association. URL: http://www.dcp-standard.org.

OMG (2023). *OMG Systems Modeling Language™ (SysML®) v2 Release*. https://github.com/Systems-Modeling/SysML-v2-Release. Accessed: 2023-05-13.

Rautenberg, Philip et al. (2023). "Electrified Powertrain Development: Distributed Co-Simulation Protocol Extension for Coupled Test Bench Operations". In: *Applied Sciences* 13.4. ISSN: 2076-3417. DOI: 10.3390/app13042657.

Reiterer, Stefan H, Clemens Schiffer, and Martin Benedikt (2022). "A Graph-Based Metadata Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations". In: *Electronics* 11.20. DOI: 10.3390/electronics11203325.

Reiterer, Stefan H., Sinan Balci, et al. (2020). "Continuous Integration for Vehicle Simulations". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE, pp. 1023–1026.

Reiterer, Stefan H. and Michael Kalab (2021). "Modelling deployment pipelines for co-simulations with graph-based metadata". In: *International Journal of Simulation and Process Modelling* 16.4, pp. 333–342. DOI: 10.1504/IJSPM.2021.118852.

Schwarz, M. H. and J. Börcsök (2013-10). "A survey on OPC and OPC-UA: About the standard, developments and investigations". In: *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*, pp. 1–6. DOI: 10.1109/ICAT.2013.6684065.

Segura, Mikel, Tomaso Poggi, and Rafael Barcena (2023). "A Generic Interface for x-in-the-Loop Simulations Based on Distributed Co-Simulation Protocol". In: *IEEE Access* 11, pp. 5578–5595. DOI: 10.1109/ACCESS.2023.3237075.

Tick, József (2007). "P-graph-based workflow modelling". In: *Acta Polytechnica Hungarica* 4.1, pp. 75–88.