Model-Based Systems Engineering Tool-Chain for Automated Parameter Value Selection

Jinzhi Lu, Dejiu Chen, Senior Member, IEEE, Guoxin Wang, and Martin Törngren, Senior Member, IEEE

Abstract—Cyber-physical systems (CPSs) integrate heterogeneous systems and process sensor data using digital services. As the complexity of CPS increases, it becomes more challenging to efficiently formalize the integrated multidomain views with flexible automated verification across the entire lifecycle. This article illustrates a model-based systems engineering tool-chain to support CPS development with an emphasis on automated parameter value selection for co-simulation. First, a domain-specific modeling approach is introduced to support the formalizations of CPS artifacts, development processes, and simulation configurations. The domain-specific models are used as the basis to generate a Web-based process management system for automated parameter value selections, which coordinates Open Services for Lifecycle Collaboration (OSLC) as the basis to generate a Web-based process management system for automated parameter value selections, which coordinates OSLC and web technologies to support CPS development with an emphasis on the efficient formalization of multidomain views and dependencies. The result shows that parameter values are selected more efficiently and effectively when implementing co-simulations using our tool-chain.

Index Terms—Automated parameter value selection, cyber-physical systems (CPSs), model-based systems engineering (MBSE).

I. INTRODUCTION

Cyber-physical systems (CPSs) refer to collections of nodes embedded with sensors that are connected through the network [1]. During CPS development, the dependency management of system architectures and parameter spaces is challenged because of the wide variety of stakeholders, particularly for co-design. First, CPS architecture development requires a comprehensive approach to specify, detect, and resolve parameter dependencies across alternative solutions because of increasing architecture complexity [2]. Second, stakeholders of CPS development are from different domains and hierarchies. Well-managed dependency design of architecture and parameter across stakeholders promotes the efficiency and effectiveness of CPS development [3]. Third, the current document-based approach leads to too many ad hoc decisions, increasing the risks of mismatching the relationships between parameters and architectures [4].

In addition to these general challenges, CPS with artificial intelligence (AI) algorithms also need simulations for testing, which requires parameter dependencies between the system architecture and data used for AI algorithm development [5]. First, the AI algorithm development always includes a considerable number of parameters and requires impractical run times to explore all their permutations in order to train the functional behaviors of AI algorithms and to verify their functional performances with different architecture alternatives [6], [7]. Second, optimizations of simulation models for different architecture solutions are needed to speed up calibration and to deliver more efficient simulation-based analysis and verification of AI algorithms [8]. Third, automated decision making based on optimizations, especially during the entire lifecycle, saves time on result review and parameter selections for AI algorithms.

Based on the previous challenges, three key perspectives are identified for managing the dependencies between architecture alternatives, parameter spaces, and data. First, data and information interoperability evolved from multiple domains and levels is the basis to manage complex information throughout system architecture and AI algorithm development [9]. Second, together with data integration, process management is required to support traceability management of data across the whole lifecycle. Stakeholders access the required data for different parameter configurations and architecture design expected as the digital thread [10]. Third, formalizations of the complex dependencies between architectures and parameter spaces are prerequisite for automated parameter value selection [11]. For example, decision-making mechanisms, system architecture, and parameter alternatives are necessary for parameter value selections. Thus, their formalisms are required when executing the related selection processes.
In this article, we provide a model-based systems engineering (MBSE) tool-chain based on domain-specific modeling (DSM) and a service-oriented approach, which at the core formalizes domain-specific views of CPS, manages co-design of CPS development, and selects parameter values for AI algorithms. The fundamental contributions of this study are as follows.

1) **Support for Automated Parameter Value Selection During CPS Development:** The DSM models are first adopted to formalize development processes, design information, and a decision-making algorithm for automated parameter value selection at abstraction levels. The DSM models are the input to the MBSE toolchain to implement parameter value selection for co-simulations automatically.

2) **Decision Making for Automated Parameter Value Selection Based on Simulations:** Simulations are used to select the design solutions for CPS and to verify their functional behaviors. In order to promote efficiency, the MBSE tool-chain executes simulations in the WPMS that implement automated parameter value selection during CPS development, particularly for AI algorithms.

The remainder of this article is organized as follows. Then, we discuss the research methodology and literature review in Section II. In Section III, details of our MBSE tool-chain are illustrated. In Section IV, we present an auto-braking case for evaluating our MBSE tool-chain. Section V evaluates our approach by comparing the case study with traditional manual methods. Finally, we offer our conclusions in Section VI. The glossary in this article is shown in Table I.

### A. Literature Review

This article mainly focuses on the MBSE tool-chain supporting automated parameter selections for which several challenges are described in the previous research. CPS have complex architectures consisting of heterogeneous components [12]. Thus, a better understanding of dependencies between requirements, architecture, and parameters enables to promote the understanding of CPS performance and development, because of having a clear logic flow to demonstrate why the system is developed [13], [14]. Second, the heterogeneous components challenge the integrated verifications of architecture alternatives [1]. Thus, data and tool interoperability is needed to address difficulties during co-design across organizations [15]. Third, the basis of automated decision making for designing the AI algorithm for CPS is the consistency and traceability management between architectures and parameters. The correct architecture models and configurations are necessary inputs to the simulation, which results are the clue to the verifications, particularly during developing AI algorithms for CPS [16].

MBSE is proposed to overcome the previous challenges for automated parameter selection during CPS development. MBSE refers to a new technical tendency, which makes use of models to replace textual requirements to support CPS development [4], [17]. Costa et al. [18] proposed a SysML4CPS profile to support CPS development based on ISO 15288, which was used by the SysML models to formalize the CPS development for the architecture and dependency design. Hause and Hummell [19] proposed a SysML profile to formalize a smart city that supports code generation for real CPS. In addition to CPS, Mordecai et al. [20] made use of OPM to formalize the interoperability of civil aviation systems of systems (SoS). These authors made use of current MBSE languages, such as SysML and OPM, to formalize domain-specific views of CPS.

In addition to MBSE formalisms, data and tool interoperability is also important to integrated simulations for CPS development. Some researchers used simulation techniques to verify integrated CPS performances. Modelica and Ptolemy are two modeling approaches to support integrated verification of hybrid systems in one modeling environment [21], [22]. Except for the independent languages and tools, the INTO-CPS project provides a co-simulation approach for integrated verification [23], [24]. The INTO CPS tool-chain integrates MBSE and co-simulation, which SysML models support co-simulation executions automatically. The functional mock-up interface (FMI) [25] and distributed co-simulation protocol (DCP) [26] are two industrial standards for co-simulation. These techniques analyze the entire system performances, providing interoperability solutions for system-level verification.

A new trend in CPS is adoption of AI algorithms for cyber computations leading to system development integrated with AI algorithm development. Currently, several simulators are used for data analytics of AI algorithm development [27], [28]. However, system-level verification for AI algorithm design and testing is important because AI algorithms are always developed for manipulating system behaviors. These

### II. RESEARCH METHODOLOGY

In this section, we first review the current challenges in CPS development, particularly in parameter selections, then focus on how MBSE to deal with such challenges and summarize the findings. Finally, case study is introduced for evaluation.
simulations are all iterative and difficult to manage, which leads to the low efficiency of the whole verification processes for the AI-enabled CPS [29]. Researchers provide solutions for improving the consistency and traceability management for automated verifications, such as Huang et al. [30] and Kapos et al. [31]. Bagnato et al. [32] proposed a tool-chain to support CPS design and co-simulation using a developed model-transformer. OpenMeta is another tool-chain to support CPS development [33]. It supports CPS formalisms using a DSM approach and transforms domain-specific models into simulation models for verification and other design work. These tool-chains support CPS formalisms and generate simulation models for automated verification; however, architecture models and simulation models are not completely linked with the development process resulting in poor consistency management. Moreover, process management with automated parameter configurations and heterogeneous data integration are not covered by these studies.

B. Summary of Literature Review

The existing works show that MBSE and tool-chains are widely used for CPS development. However, most of them were developed for specific purposes, limiting further extension. Compared with the existing methods, we can summarize several key motivations related to our work.

1) Domain-Specific Formalisms: CPS artifacts and development are formalized not only using SysML or UML but also using domain-specific views, which are the basic compositions to construct a DSM language (DSML) for architecture development and parameter configurations. For example, the automated parameter selection requires correct architectures, parameter settings, and decision-making algorithms for each case. Thus, a more powerful DSM approach is expected to formalize all the views across the lifecycle with decision making.

2) Data and Tool Interoperability: Data, models, and tools are required to be integrated across the lifecycle with better interoperability. In order to automate parameter selections for integrated verifications, domain-specific data is described as unified formats to implement co-simulation and information exchange among design tools.

3) Automated Decision Making for Parameter Value Selection During Developing AI Algorithms: Simulations promote the flexibility of AI algorithm design, because simulation results can replace the data required from the real world for machine learning algorithm training. Most simulations support data training and system verification of CPS development, but it often takes too much time for repeating operations. Thus, an automated decision-making approach is required to improve the consistency and traceability management for parameter value selection in more complicated development processes.

C. Evaluation Using Case Study

A case study is a way to understand complex issues in real-world applications [34]. Based on it, autonomous driving systems (ADSs) development is adopted to evaluate our MBSE tool-chain prototype in the SAFER SARMITS project.1 Focusing on the concerns in this article, we design a scenario for developing a decision-making algorithm in an auto-braking scenario for cooperative vehicle-infrastructure systems (CVISs). The algorithm is used to identify the available parameters to control the auto-braking of vehicles in a CVIS. To support this scenario, an MBSE tool-chain based on Simulink is developed that is evaluated by the automated parameter value selection compared with the manual processes both qualitatively and quantitatively.

Internal validity and external validity are two important aspects of the case study approach [35]. The internal validity means ensuring that the claims made by a study are correct, e.g., how sure a researcher is that there are no plausible alternative explanations to the results of an experiment. It refers to a process to identify whether the MBSE tool-chain prototype can realize the motivations of this case study by measuring different metrics. The external validity of the case study refers to the extent to which the MBSE tool-chain can be replicated in other scenarios. We make use of the scalability to analyze this aspect.

The measurements for evaluating the tool-chain include the efficiency and effectiveness of the ADS development. The efficiency refers to the relative amount of time spent when stakeholders implement their design jobs by the tool-chain, which is evaluated using a quantitative approach. The efficiency metrics include: 1) time-saving rate focusing on the scenario that automated decision-making supports parameter value selection; it refers to the ratio of the total time needed with automated parameter value selection to the total time needed without automated parameter value selection and 2) automation-level focusing automated tool operations [36]. It refers to the number of tool operations that are implemented automatically in the tool-chain.

The effectiveness refers to how well stakeholders implement their design jobs by developed tool-chains, which is evaluated using a qualitative approach. The effectiveness measurement includes the following.

1) Domain-specific formalism refers to the capability that CPS development is formalized by domain-specific views. We measure this by identifying domain-specific views represented by the models.
2) Capability of process management refers to the capability that the co-design processes are managed and controlled. This capability is measured by the situation if the development process can be controlled and managed through IT techniques.
3) Traceability of development process and technical resources refers to the capability that developers trace the technical resources from the development processes. The traceability is an important metric to support consistency management of the automated parameter

---

1The case study is from the SAFER SARMITS project. https://www.saferesearch.com/projects/sarmits-systematic-approach-risk-management-its-context.
configurations which is measured by how the traceability is created and managed.

4) Interoperability of data and tools refers to the capability that tools and data can be accessed by other tools which are measured by how the data and tool exchange information with each other.

III. MBSE TOOL-CHAIN SUPPORTING AUTOMATED PARAMETER VALUE SELECTION

In this section, an overview of the MBSE tool-chain for ADS development is first introduced. Then, a GOPPRR (Graph, Object, Port, Property, Relationship, and Role) approach is proposed to formalize the domain-specific views of ADS and decision-making algorithms for selecting the parameter values of AI algorithms. Finally, a service orchestrator is demonstrated to implement automated parameter value selection.

A. Overview of the Proposed MBSE Tool-Chain

Fig. 1 represents how an automated decision-making algorithm is implemented in an MBSE tool-chain for parameter value selection during ADS co-design. The MBSE tool-chain is constructed based on DSM, co-simulation, and a service-oriented approach. When using the tool-chain, system developers build DSM models in MetaEdit+. Then, the DSM models are transformed into XML files. Based on the information in XML, a decision-making algorithm is developed to support service orchestration for the parameter value selection. In the MBSE tool-chain, a WPMS is used to implement co-simulation through OSLC services. The decision-making algorithm automates the processes of parameter value selection in the WPMS so that co-simulations are implemented based on the previous simulation data. The co-simulation is used to support integrated verification of ADS, particularly data training and verification for AI algorithms, including MATLAB/Simulink, Carmaker, and MWorks.

The co-simulation is implemented in Simulink based on S-functions for Carmaker models and functional mock-up units (FMUs) [25], which are used at the following phases.

1) In conceptual design phases, Simulink models (SMs) are used for verifying a conceptual ADS at the SoS level. Compared with co-simulation models (CSMs), SMs are not accurate without vehicle dynamics and detailed subsystem dynamics, but their execution speeds are high.

2) In system design phases, CSMs in Simulink, integrated with Carmaker models, are used to verify the ADS performance. The CSM with vehicle dynamics are used to verify the ADS concepts at the system level.

3) In subsystem design phases, Modelica models of subsystems are generated into FMUs. After integrating FMUs and Carmaker models, detailed CSMs (DCSMs) are used for ADS verification at the subsystem level.

B. DSM Formalizing ADS Development

In order to formalize and manage the system artifacts and development of ADS, DSM models are developed based on a GOPPRR approach. In previous research [37], the GOPPRR approach is proposed based on an M0–M3 modeling framework, which includes meta-metamodels: Graph, Object, Point, Property, Relationship, and Role. The details are as follows.

1) Graph refers to a collection of Object and Relationship bindings describing the connection rules between Objects and Relationships.

2) Object refers to a concept that can exist on its own (independent of relationships and roles).

3) Point refers to one port connecting a Relationship with Objects.

4) Property refers to one attribute of other metamodels.

5) Role specifies how a Point in one Object is linked with a Relationship.

6) Relationship refers to one connection existing between Objects. Objects are connected by Relationships through Roles.

Based on these meta-metamodels, metamodels are developed to formalize CPS development and system artifacts based on BPMN [38] and SysML [39].

In order to realize automated parameter value selection for AI algorithms, the DSM models are designed to include two parts: 1) process pattern and 2) information pattern in Fig. 2. The process pattern is used to formalize the development process of conceptual, architecture, and subsystem designs, and making decisions for parameter value selections. The metamodels of the process pattern are developed based on BPMN.

1) Start represents a Start event in BPMN referring to a start node in the process.

2) Worktask represents an Activity in BPMN. The tasks have two types: a) human work task (HWT), a task implemented by one human and b) automated work task (AWT), a task implemented automatically.

3) Decision gate represents a Gateway in BPMN, to define one node forking of paths depending on the
expressed conditions including two points. \textit{Point $S$} refers to the next action when the condition is satisfied. \textit{Point $U$} refers to the next action when the condition is not satisfied.

4) \textit{End} represents to an \textit{End event} in BPMN referring to an end node in the process.

5) \textit{Sequence} represents to a \textit{Sequence flow} in BPMN referring to one connection in the process.

The information pattern represents system artifacts and decision makings in the \textit{Worktask Object} and \textit{Decision gate Object} separately. There are two Graph types for information patterns: 1) \textit{Worktask Graph}, which aims to describe system artifacts, including requirements, functions, system structures, verifications, and their interrelationships and 2) \textit{Decision gate Graph}, which is used to describe the decision makings for the parameter selection whose metamodels are shown in Fig. 3(c).

The \textit{Worktask Graph} formalizes design information of ADS using the following \textit{Objects}.

1) \text{\textit{Requirement Object}} refers to one requirement in work tasks.

2) \textit{Function Object} refers to one function refining the requirement satisfied by system architecture and components.

3) \textit{System Architecture Object} refers to one ADS model [Fig. 3(b)].

4) \textit{Component Object} refers to one model component to define the compositions in the System Architecture [Fig. 3(b)].

5) \textit{Verification Object} refers to a set of simulation tasks to verify the requirements. Each task represents one simulation execution [Fig. 3(c)].

The syntax of the \textit{Worktask Graph} is extended based on the SysML requirement diagram. Therefore, relationships between these objects include \textit{Refine}, \textit{Derive}, \textit{Satisfy}, \textit{Containment}, \textit{Trace}, \textit{Verify}, and \textit{Copy} relationships.

Compared with previous work [36], Fig. 3 shows a decision-making process and related metamodels to formalize and to support automated parameter value selection. In Fig. 3(a), an analysis process for decision makings of parameter value selection with three inputs is formalized by metamodels in Fig. 3(d) and all its inputs are provided by separated developed models.

1) \textit{System architecture constraints} refers to the system architecture used for the analysis process describing the system architecture and its compositions. The inputs are obtained from system architecture models which are constructed based on metamodels in Fig. 3(b).

2) \textit{ConfigurationsInVV constraints} refers to the configurations in the verification for the analysis process, including simulation and parameter configurations. The inputs are obtained from verification models which are constructed based on metamodels in Fig. 3(c).

3) \textit{Objective} refers to a decision-making objective, which defines \textit{GoalFunctions} for decision making, goals, and results of verification. The \textit{GoalFunctions} are implemented based on simulation results. The inputs are obtained from decision-making models which are constructed based on metamodels in Fig. 3(d).

Based on these inputs, the analysis process performs two outputs: 1) decisions for implementing a process in the WPMS and 2) design space of parameter values for the next-round verification.

In Fig. 3(b), structure models are constructed by the model components representing the ADS at the SoS, system, and subsystem levels in the System Architecture Graph. Each model component is decomposed into its subsystem graphs. \textit{SimulinkBlock Objects} referring to the Simulink blocks representing detailed compositions of the entities at the SoS level. In Fig. 3(c), metamodels of the verification Graph are used to describe a sequence order of tasks and related simulation and parameter configurations, and simulation results. The parameter types are used to define types of specification concepts illustrated in Table II.

In Fig. 3(d), metamodels for formalizing a decision-making process are demonstrated. The \textit{GoalFunction} follows these rules in this article. First, if all the simulation results are not satisfied by the \textit{Objective}, the process terminates and returns to the Task connected with \textit{Point $U$} in the Decision gate. If some of the simulation results satisfy the \textit{Objective}, the next-round actions connected with \textit{Point $S$} in the Decision gate are implemented until the next Decision gate. The parameter value that satisfies System Architecture constraints and ConfigurationsInVV constraints of the given \textit{Objective} are recorded in the design space. During the next-round actions, verification is implemented under the configurations of parameter values in the defined design space (details introduced in Section III-C).

C. Service Orchestration for Automated Parameter Value Selection

In this section, a specification supporting parameter value selection is first introduced. Based on the specification, an algorithm is developed for decision making in selecting parameter values. Finally, a service orchestrator is developed to support automated parameter value selections in the MBSE tool-chain.

1) Specification for Parameter Value Selection: As mentioned in Section III-A, a virtual environment is proposed to support ADS development using SM, CSM, and DSM. In
the conceptual design phase, SMs are used for defining the AI algorithms to identify the parameter values that satisfy the initial requirements (e.g., the parameter values that two cars cannot crash in the case study). In the system design phase, CSMs are proposed to verify the system performance using the ADS models with vehicle dynamics under the identified parameter values. During this phase, the parameter values obtained from the previous phase are verified as well. In the subsystem design phase, DSMs are used to verify the subsystem components and to identify the parameter values at last.

Parameter value selection is defined as one automated process for identifying parameter values based on decision making. We make use of a formal method to specify this process, fundamentally shown as follows:

\[ D_{\text{Opt}}^t = \text{Ansys}_t(SysArcConstraint}_a, ConfigConstraint}_b, Objective}_c ) \]  

(1)

where \( D_{\text{Opt}}^t \) refers to one decision for the \( t \)-th parameter value selection in the process. \( Opt \) refers to an output type used to identify the outputs of the decision-making process: 1) decision for process management (\( Opt = 1 \)) and 2) design space of identified parameter sets (\( Opt = 2 \)). \text{Ansys}_t() \) refers to an analysis process to obtain the \( D_{\text{Opt}}^t \). In a decision-making process (including architecture modeling, verification, and decision makings) modeled based on the process pattern, \( SysArcConstraint \) refers to the \( t \)-th system architecture used for the \( b \)-th verification before the decision gate. \( ConfigConstraint \) refers to simulation and parameter configurations in the \( b \)-th verification task. In the \( ConfigConstraint \), there are two types of variables: 1) \( gV \) refers to the parameters whose values we aim to select for the next round co-simulations and 2) \( gT \) refers to the variables whose values are configured, and is used to identify whether the simulation results under \( gT \) and \( gV \) satisfy the \( Objective_c \). The \( Objective_c \) refers to the objectives for the analysis of the \( c \)-th \( GoalFunction \). In this article, a parameter value selection algorithm \( PS() \) is adopted as a \( GoalFunction \)

\[ Objective_{PS} = PS(gV, gT, Goal, Res) \]  

(2)

where \( PS() \) refers to a \( GoalFunction \) for parameter value selection. \( gV \) refers to all variables that need to be selected in the \( ConfigConstraint \). For example, in the case study, \( gV \) refers to the parameters in SM selected for CSM, and DSM. Thus \( gV = P_{\text{Wait}} \cup P_{\text{Acc}} \cup R_{\text{Acc}} \cup R_{\text{Wait}} \cup R_{\text{Dec}} \), where \( P_{\text{Wait}}, P_{\text{Acc}}, R_{\text{Acc}}, R_{\text{Wait}}, \) and \( R_{\text{Dec}} \) (details introduced in Section IV-A) denote the input variables of the AI algorithm in SM, CSM, and DSM. These parameters are expected to be selected in order to verify if the algorithm can support the auto-braking scenario. \( gT \) refers to variables for configuring the models which are not required for selection. It is used to verify whether the models under \( gV \) and \( gT \) satisfies \( Goal \). For example, in the case study, the initial states of these two vehicles are used to test if the auto-braking algorithm can work. Thus, \( gT = v_{1i} \cup a_{1i} \cup d_{1i} \cup v_{2i} \cup a_{2i} \), where \( d_{1i} \) is an initial distance, \( v_{1i} \) and \( v_{2i} \) are initial speeds, and \( a_{1i} \) and \( a_{2i} \) are initial accelerations of V1 and V2 (details introduced in Section IV-A). The \( Goal \) refers to a target function
for the decision making. For example, the Goal in the case study is that two cars do not crash (distance between two cars is more than 1.5 m during entire simulations). Res refers to the simulation results under gV and gT. Using PS(), one simulation model SysArcConstraint under each given parameter value of gV and gT in ConfigConstraint is executed to obtain Res of the simulation. Then, the Res is used to implement the analysis process. PS() is formally defined to support decision making based on \( \zeta \)

\[
\zeta_i = \frac{\text{Size}(\text{SysArcConstraint}_SM, PS(gV_i, gT, \text{Goal, Res}_i))}{\text{Size}(\text{SysArcConstraint}_SM, \text{ConfigConstraint}_VV_SM)} \tag{3}
\]

where \( \zeta_i \) refers to the ratio between the simulations satisfying ObjectivePS and all the simulations under ConfigConstraintVV_SM and SysArcConstraint_SM. Size() refers to the number of simulations under all the parameters of ConfigConstraintVV_SM or the parameters which are selected by PS(gV, gT, Goal, Res) based on SysArcConstraint_SM. ConfigConstraintVV_SM refers to all the configuration scenarios for simulation and parameter setting for each model. gV refers to one value set of gV that satisfy the ObjectivePS which is stored in the design space. Res refers to the simulation results obtained under gV and gT.

2) Decision-Making Algorithm Supporting Parameter Value Selection: Based on the defined specification, an algorithm is proposed to support automated parameter value selection in Algorithm 1. SysArcConstrainta, ConfigConstraintb, gV, gT, Res, and t are inputs to the algorithm. The analysis process sets up first to check whether this is the first occurrence of the algorithm. If it is implemented for the first time, Ansys is executed to calculate \( P_\zeta \) directly referring to a probability (the number of simulation results in which gV satisfy the Goal) (the total number of simulation results). If the analysis process is implemented for a subsequent time, \( De_i^2 \) should be added the previous \( De_i^{2,1} \). The decision-making algorithm is defined based on a logical causality of \( P_\zeta \). If \( (P_\zeta == 100\%) \), then gV and SysArcConstrainta are added into \( De_i^2 \), meaning that the SysArcConstrainta under gV can be used for the later verification. Another logical causality is used to make decisions for process management: if \( De_i^2 \) is empty, the process returns to the previous node through Point U. For example, in Section IV, under environmental conditions gT and AI algorithm parameters gV, \( P_\zeta \) refers to a rate of simulations at which two vehicles do not crash. When \( (P_\zeta = 100\%) \), gV can be configured for the CSM and DCSM at later phases.

3) Orchestrating Automated Parameter Value Selection: In our proposed MBSE tool-chain, DSM models are transformed to a WPMS with OSLC services. First, the DSM models are transformed to XML files. Then, using such XML files, a service compiler is developed to support the service orchestration with the following steps, illustrated in Fig. 5.

1) OSLC services of technical resources (e.g., models, data, and APIs of tools) are generated through related OSLC adapters. Such OSLC services, referring to RESTful services, support technical resources to be accessible to multiple tools via URLs [40].

\[
\begin{align*}
\text{Algorithm 1 Decision-Making Algorithm for Parameter Value Selection} \\
\text{Input: SysArcConstraint}_a, \text{ConfigConstraint}_b, gV, gT, \text{Res}, t \\
\text{Output: } De_i^1, De_i^2 \\
\text{Check if this is the first time the parameter value selection of SysArcConstraint}_a is being done:} \\
\text{The parameter value selection is NOT the first time:}
\end{align*}
\]

2) The process pattern represented in XML files is transformed to a WPMS by a compiler based on BPM Camunda [38]. Each work task in the WPMS is linked to the related OSLC service provider through a defined URL.

3) The process pattern in XML files is also transformed to OSLC service providers and are mapped to the related work tasks in the development process. The information pattern referring to design information is transformed to OSLC services in the related service providers. Such
Fig. 4. MDP algorithm supporting auto-braking in the CVIS infrastructure. (a) Problem statement. (b) Markov decision process model for the decision making in CVIS to control the auto-breaking system in V2.

4) Service orchestration templates\(^4\) are generated by its adapter to delivering end-to-end services from the development process to technical resources. Since OSLC services and service providers are generated from the XML files, their information is updated to Neo4j, a graph database \([39]\). Through the service orchestration templates defined in Neo4j, each OSLC service provider is linked to the required OSLC services of information pattern and technical resources. Technical resources are accessible to the WPMS through their OSLC services. Developers implement related technical resources without any manual operations.

5) Finally, Algorithm 1 is implemented with the service orchestration templates to realize automated parameter value selection.

\(^4\)This article \([41]\) illustrates the XML files of the service-orchestration template.

IV. CASE STUDY

A. Problem Analysis

1) Scenario Definition: We define a scenario to illustrate how our MBSE tool-chain supports automated parameter value selection during ADS development in the SAFER SARMITS project. This scenario is about the development of an AI algorithm, specifically, a Markov decision process (MDP) algorithm, to control an auto-breaking system in one vehicle in order to satisfy the Three-Second Following Distance Rule during the CVIS development. The rule is a measurement of the time interval for the vehicles to pass the same fixed point on a road: if the vehicle reaches a point within 3 s of the previous vehicle, the distance is too short. For icy or snow-covered roads, the corresponding interval can be longer. The purpose of the case-study process is to verify the availability of the MDP algorithm.

In Fig. 4(a), vehicles V1 and V2 send their real-time data to the infrastructure manipulated by the AI algorithm. The algorithm controls the behaviors of V2 in order to satisfy the Three-Second Following Distance Rule. V1 is an environmental vehicle with uncertainties, whose behaviors can be changed manually during simulations. Through communications between infrastructure and vehicles, V2 receives commands from the infrastructure and manipulates its auto-breaking system to brake. In this case, we define PS() to verify if the following vehicle can brake automatically to prevent these two cars from crashing under given environmental conditions.

Fig. 4(b) shows an MDP model to control the auto-breaking system in the following vehicle. The details are as follows.

Definition 1: We define status \(S_t\) at time \(t\) as a status of the distance \(d\) (distance between vehicles V1 and V2), which can be Green, Red, or Black. We assume that vehicles V1 and V2 are moving in the same direction (x-axis) on the road. \(p1, p2, v1, v2, a1,\) and \(a2\) refer to the positions, velocities, and accelerations of V1’s and V2’s geometric centers in the X direction, respectively. At \(t\), if \(d\) is not less than \((v2 \times 3)\) s, then \(S_t\) is Green. If \(d\) is less than \((v2 \times 3)\) s or less than 8 m, then \(S_t\) is Red. If \((d < 1.5)\), then \(S_t\) is Black (Crash).

The MDP algorithm is used to control V2 to take an action \(Act_t\) of auto-braking at \(t\). \(Act_t\) can be acceleration...
where \( V = \text{constant-speed} \), \( \alpha = 3 \text{ m/s}^2 \), constant speed \( \alpha = 0 \text{ m/s}^2 \), or deceleration \( \alpha = -4 \text{ m/s}^2 \) in order to keep the two vehicles in the status Green.

**Definition 2:** In Fig. 5(b), an MDP model is proposed for controlling the actions of the auto-braking system. When \( S_i = \text{Green} \), we define \( \alpha_{\text{Acc}}, \alpha_{\text{Wait}}, \) and \( \alpha_{\text{Dec}} \) as probabilities, where after V2 takes an action \( \text{Actt} \) (acceleration, constant speed, or deceleration), \( S_{i+1} \) is still Green. When \( S_i = \text{Red} \), \( \beta_{\text{Acc}}, \beta_{\text{Wait}}, \text{and} \beta_{\text{Dec}} \) are defined as probabilities, where after V2 takes an action \( \text{Actt} \) (acceleration, constant speed, or deceleration), \( S_{i+1} \) is still Red.

**Definition 3:** We define \( y \) as one learning factor in MDP, \( V_1 \) is the probability that \( S_i \) changes to Green at \( t \) and \( V_2 \) is the probability that \( S_i \) changes to Red at \( t \).

**Definition 4:** Based on reinforcement learning concepts [42], we define a transition from \( S_{i-1} \) to \( S_i \) referring to \( (S_{i-1}, \text{Actt}, S_i) \). \( R_{\text{Acc}}, R_{\text{Wait}}, \) and \( R_{\text{Dec}} \) are awards associated with this transition for \( \text{Actt} \) (acceleration, constant speed, or deceleration). Therefore, if \( S_{i-1} = \text{Green} \), after a transition \( (S_{i-1}, \text{Actt}, S_i) \), the probability of \( S_i \) changing to Green is referred to as \( V_{\text{green},i} \) at \( t \); if \( S_{i-1} = \text{Red} \), after a transition \( (S_{i-1}, \text{Actt}, S_i) \), the probability of \( S_i \) changing to Red is \( V_{\text{red},i} \) at \( t \).

If \( S_i = \text{Green} \), \( V_{\text{green},i} \) is calculated as follows:

\[
V_{\text{green},i} = \begin{bmatrix} R_{\text{Acc}} + y \times (\alpha_{\text{Acc}} \times V_{1,i-1}) \\ R_{\text{Wait}} + y \times (\alpha_{\text{Wait}} \times V_{1,i-1}) \\ R_{\text{Dec}} + y \times (\alpha_{\text{Dec}} \times V_{1,i-1}) \end{bmatrix}
\]

(4)

where \( V_{1,i} \) is the maximum value of \( V_{\text{green},i} \).

\[
V_{1,i} = \text{Max}(V_{\text{green},i}(1:3)).
\]

(5)

**Definition 5:** We define \( P_{\text{Acc}} \) and \( P_{\text{Wait}} \) as punishments associated with the transition \( (S_{i-1}, \text{Actt}, S_i) \) aiming to punish the \( \text{Actt} \) (acceleration and constant speed).

Therefore, if \( S_i = \text{Red} \), \( V_{\text{red},i} \) is calculated as follows:

\[
V_{\text{red},i} = \begin{bmatrix} P_{\text{Acc}} \times \beta_{\text{Acc}} + R_{\text{Acc}} \times (1 - \beta_{\text{Acc}}) + y \times (\beta_{\text{Acc}} \times V_{1,i-1} + (1 - \beta_{\text{Acc}}) \times V_{2,i-1}) \\ P_{\text{Wait}} \times \beta_{\text{Wait}} + R_{\text{Wait}} \times (1 - \beta_{\text{Wait}}) + y \times (\beta_{\text{Wait}} \times V_{1,i-1} + (1 - \beta_{\text{Wait}}) \times V_{2,i-1}) \\ R_{\text{Dec}} + y \times (\beta_{\text{Dec}} \times V_{1,i-1} + (1 - \beta_{\text{Dec}}) \times V_{2,i-1}) \end{bmatrix}
\]

(6)

where \( V_{2,i} \) is the maximum value of \( V_{\text{red},i} \).

\[
V_{2,i} = \text{Max}(V_{\text{red},i}(1:3)).
\]

(7)

Based on the previous equations, a tradeoff algorithm is proposed to implement \( \text{Actt} \) at \( t \) in Algorithm 2.

**Algorithm 2 Tradeoff Algorithm for Implementing \( \text{Actt} \)\)**

**Input:** \( S_i, V_{\text{green}}, V_{\text{red}} \)

**Output:** \( \text{Actt} \)

1. Check status of current distance between \( V1 \) and \( V2 \):
   1. if \( S_i = \text{‘Green’} \) then
   2. if \( V_{\text{green},i}(1) = \text{Max}(V_{\text{green},i}(1:3)) \) then
      3. \( \text{Actt} = \text{‘acceleration’} \)
   4. end if
   5. if \( V_{\text{green},i}(2) = \text{Max}(V_{\text{green},i}(1:3)) \) then
      6. \( \text{Actt} = \text{‘constant-speed’} \)
   7. end if
   8. if \( V_{\text{green},i}(3) = \text{Max}(V_{\text{green},i}(1:3)) \) then
      9. \( \text{Actt} = \text{‘deceleration’} \)
   10. end if
   11. else
      12. if \( V_{\text{red},i}(1) = \text{Max}(V_{\text{red},i}(1:3)) \) then
         13. \( \text{Actt} = \text{‘acceleration’} \)
      14. end if
      15. if \( V_{\text{red},i}(2) = \text{Max}(V_{\text{red},i}(1:3)) \) then
         16. \( \text{Actt} = \text{‘constant-speed’} \)
      17. end if
      18. if \( V_{\text{red},i}(3) = \text{Max}(V_{\text{red},i}(1:3)) \) then
         19. \( \text{Actt} = \text{‘deceleration’} \)
      20. end if
      21. end if
   22. return \( \text{Actt} \)

**Table II Parameter Values of Simulation**

| Specification Concepts | Parameter | Input candidates |
|------------------------|-----------|------------------|
| \( gV \)               | \( P_{\text{acc}} \) | \{-7, -8, -9\} |
|                        | \( P_{\text{wait}} \) | \{-1, -2\} |
|                        | \( \beta_{\text{acc}} \) | \{0.155, 0.156, 0.157\} |
|                        | \( \beta_{\text{wait}} \) | \{0.45, 0.46\} |
| \( gT \)               | \( \alpha_{\text{ini}} \) | \{1, 0, 1\} |
|                        | \( \alpha_{2\text{ini}} \) | \{1\} |
|                        | \( v_{1\text{ini}} \) | \{15, 16, 17, 18, 19, 20\} |
|                        | \( v_{2\text{ini}} \) | \{\} |
|                        | \( d_{\text{ini}} \) | \{40\} |

**parameter value selection in the MDP algorithm. Several models are used to realize the parameter value selections from the input candidates and verify the parameters with more detailed system dynamics.**

1. In the conceptual design phase, one SM is used for initial verification of the MDP algorithm at the SoS level and selecting the parameter value candidates for the future verifications based on co-simulations in later phases.

2. In the system design phase, Simulink and Carmaker are integrated into CSM for co-simulations in order to verify the MDP algorithm with vehicle dynamics at the system level.

3. In the subsystem design phase, one Modelica model of the transitioning system in the following vehicle is built and transformed into one FMU. After integrating FMUs, Simulink, and Carmaker, DCSM is used for verifying the MDP algorithm with more detailed subsystem features (multibody dynamics of the transitioning system).

**3) DSM Models Supporting Automated Parameter Value Selection:** In order to formalize the automated parameter
value selection, we develop DSM models based on metamodels (introduced in Section III-B). The DSM models include process and information patterns describing development processes and required information in each work task. To support the parameter value selection, the system architecture, verification, and decision-making graphs represent the basis of one parameter value selection with associated goals and constraints in Fig. 6(a). The process pattern (A1) represents an algorithm development process, including HWT and AWT objects. Decomposed from each work task, the information pattern represents detailed information introduced as follows.

1) 1, 5, and 9 define the requirements used for the algorithm development at the SoS, system, and subsystem levels.

2) 2, 6, and 10 define the system architecture. For example, in Fig. 6A2, an architecture model includes the two vehicles, sensors, infrastructure, and environment objects.

3) 11 and 12 define the information for FMUs of the subsystem in order to support co-simulation between FMU, Carmaker, and Simulink. Developers manipulate MWorks to generate one FMU from Modelica models and upload it to the defined path through the WPMS in these two tasks.

4) 3, 7, and 13 refer to the AWT for verification at SoS, system, and subsystem levels. As illustrated in Fig. 6A3, P_Wait, P_Acc, R_Acc, R_Wait, and R_Dec are four parameters of the MDP algorithm referring to gV in the Verification Graph, and a1_ini, a2_ini, v1_ini, v2_ini, and d_ini are five parameters of environmental conditions referring to gT. Different environmental conditions refer to different scenarios in which each simulation task is implemented under the parameter values (Table II).

5) 4, 8, and 14 refer to Decision gates for implementing the analysis process. In the Decision gate Graph (Fig. 6A4), System Architecture Constraint, Configuration Constraint, and Objective Objects represent an analysis process for making decisions based on the simulation results. The System Architecture Constraint Object defines an architecture model for the verification. The Configuration Constraint Object defines gT and gV using parameters defined in Fig. 6A3. The Objective Object defines the GoalFunction and Goal referring to the case when the distance between
two vehicles is larger than 1.5 m. The Analysis Object defines the parameters used in the analysis process, such as the \textit{id} of the analysis process. The Decision Object defines the parameters used for the design space and process management system, such as the \textit{id} of the design space.

4) Workflow Supporting Automated Parameter Value Selection in the WPMS: In this article, the DSM models are built in MetaEdit+ and then are transformed into XML files in Fig. 6(b). A compiler is developed to transform XML files and technical resources into a WPMS linked with OSLC service providers and services through the corresponding URLs. Then, developers implement each work task in the WPMS to deploy the required technical resources (e.g., FMUs) and to access design information (e.g., requirements, features, and functions) represented in the DSM models through the related OSLC services.

In Fig. 6(c), the Decision gates in the WPMS are implemented for parameter value selection through OSLC services. Algorithm 1 is implemented by the service orchestrator (as shown in Fig. 5) for automated decision making in parameter value selection.

1) In the WPMS, each work task links to the related OSLC service provider through its URL. For example, the \textit{Verification in the conceptual phase} work task in the WPMS is linked to the related OSLC service provider where OSLC services representing the information pattern is mapped to the \textit{Verification in the conceptual phase Graph} and required technical resources. In the verification work tasks, developers implement the related OSLC service providers to implement simulations automatically.

2) In the \textit{Decision gate} after verification, its OSLC service provider links to a set of OSLC services representing simulation results under \textit{gV} and \textit{System architecture constraint} (the executed simulation model represented as a system architecture graph in DSM models) (see step 4). For example, one OSLC service represents simulation results under \textit{gV} (e.g., \{\textit{P\_acc} = -7, \textit{P\_wait} = -1, \textit{\beta\_acc} = 0.115, \textit{\beta\_wait} = 0.45\}) and \textit{gT} (e.g., \{\textit{a1\_ini} = -1, \textit{a2\_ini} = 15, \textit{v1\_ini} = 15, \textit{v2\_ini} = 15, \textit{d\_ini} = 40\}) in Table II. The OSLC service is accessed by URI to implement the analysis to check if the result satisfies the \textit{GoalFunction}.

3) Each OSLC service of results is linked to the OSLC services of the related parameter values under \textit{gV} and \textit{gT} and the OSLC service of the \textit{System architecture constraint}.

4) The analysis is implemented based on OSLC services representing simulation results linked with OSLC services of \textit{ConfigConstraint} (\textit{gV} and \textit{gT}) and \textit{System architecture constraint} (the system architecture Graph used for the verification before the \textit{Decision gate}). The \textit{GoalFunction} is that the two cars cannot crash (the distance between them must be larger than 1.5 m) under 30-s simulations. Results of the distance during the entire simulation time is obtained from each OSLC service. Then, other input required by Algorithm 1 is obtained through their OSLC services to calculate the \textit{P}($\zeta$). If (\textit{P}($\zeta$) = 100%), the OSLC services of \textit{gV \{P\_acc, P\_wait, \beta\_acc, \beta\_wait\}} and \textit{System architecture constraint} are linked to one OSLC service representing the design space for this analysis.

5) During the following verifications, such as \textit{verification at system phase} and \textit{verification at subsystem phase} work tasks, the OSLC services of the previous design space is linked to implement the decision-making algorithm as well.

\textbf{B. Simulation Results}

Through the WPMS, developers implement the parameter value selection and system verification automatically. First, the parameters in Table II are implemented using SM for selecting the initial parameters of the MDP algorithm. Then, CSM and DCSM are implemented to verify the selected parameter values with more detailed system dynamics at later phases.

Based on the automated parameter value selection using SM, \textit{gV} (8/36 parameter values) satisfy the \textit{GoalFunction} (no crash) in Table III. These parameter values are defined in the design space for the later verification \textit{Worktasks} to configure the CSM and DCSM for further co-simulations. In Fig. 7, simulation results under CSM and DCSM are similar and are slightly different from SM. This is because CSM and DCSM include complex vehicle dynamic models and detailed subsystem models, whereas SM is a simplified model without such detailed dynamic features.

Fig. 7 shows the distance and speed differences between V1 and V2. The results show that the curves of SM, CSM, and DCSM are the same under \textit{gV} and \textit{gT} (with the same environmental conditions) in Table III. This is explained by the fact that \textit{gV} decides if the MDP algorithm works but it does not influence the MDP performances, no matter which scenarios they are in. However, under different environmental conditions, the curves are different. When (\textit{a1} = 1), the distance between the two cars increases until the two vehicles reach the speed limitations. The speed variations increase then decrease, until finally these two vehicles are at the same speed. When (\textit{a1} = 0), the distance increases slightly then stays at a constant value. The speed variations increase then decrease quickly. When (\textit{a1} = -1), the distance decreases until the two vehicles stop.

\textbf{V. DISCUSSION AND EVALUATION}

Quantitative and qualitative approaches are used to evaluate the case study and compare the proposed procedure with manual processes without the MBSE tool-chain (Section II-C).

\begin{table}[h]
\centering
\caption{Selected Parameter Values (X represents No Crash)}
\begin{tabular}{ccccccc}
\hline
\textit{P\_acc} & \textit{P\_wait} & \textit{Beta\_wait} & \textit{Beta\_acc} & \textit{SM} & \textit{DCSM} \\
\hline
.9 & .1 & 0.45 & 0.156 & X & X \\
.9 & -1 & 0.46 & 0.156 & X & X \\
-9 & -1 & 0.45 & 0.156 & X & X \\
-9 & -2 & 0.46 & 0.156 & X & X \\
-9 & -1 & 0.45 & 0.157 & X & X \\
-9 & -2 & 0.45 & 0.157 & X & X \\
\end{tabular}
\end{table}
TABLE IV
SIMULATION DURATIONS

| Model   | Average time per simulation (AT) | Times for simulation (Times) | Total time consumption |
|---------|----------------------------------|------------------------------|------------------------|
| SM      | 7s                               | 648                          | 78min54s               |
| CSM     | 9min3s                           | 144                          | 1303min12s             |
| DCSM    | 10min57s                         | 144                          | 1572min                |

A. Quantitative Evaluation

Table IV illustrates average simulation durations using different models, where $AT$ refers to an average time to implement the SM, CSM, and DCSM separately, calculated by one timer during simulations. Total simulation times are defined referring to how many times the simulation was executed under $ConfigConstraint (gV$ and $gT)$ for each model as follows:

$$Times_{Total} = Times_{SM} = Size(P_{acc}) \times Size(P_{wait}) \times Size(\beta_{acc}) \times Size(\beta_{wait}) \times Size(a1_{ini}) \cdots \times Size(d_{ini}) = 540 \quad (8)$$

where $P_{acc}, \ldots, \beta_{acc}, \ldots, d_{ini}$ are as listed in Table II.

After the first-time parameter value selection, the design space is defined using values of $gV_i$ (as shown in Table III) that satisfy the $GoalFunction$. Therefore, the simulation times using CSM and DCSM are defined as follows:

$$Times_{CSM} = Times_{DCSM} = Size(P_{acc}) \times Size(P_{wait}) \times Size(\beta_{acc}) \times Size(\beta_{wait}) \times Size(a1_{ini}) \cdots \times Size(d_{ini}) = 120 \quad (9)$$

where $P_{acc}, \ldots, \beta_{wait}$ refer to the parameters in Table III and $a1_{ini}, \ldots, d_{ini}$ refer to the parameters in Table II.

The total time consumption refers to the duration of total simulation times if there is not an automated parameter value selection. That means CSM and DCSM need to be executed by the same simulation times as SM. Therefore, we compared the total time consumption with the one using the automated parameter value selections under Table II. The time-saving rate is defined as

$$Rate_{time-saving} = 1 - \frac{Times_{Total} \times AT_{SM} + Times_{CSM} \times AT_{CSM} + Times_{DCSM} \times AT_{DCSM}}{Times_{Total} \times AT_{SM}} \times \frac{100\%}{100\%} = 78.1\% \quad (10)$$

From the result, we find that the time-saving rate is 78.1%, meaning less time is needed to implement all the simulations. When the developers design the MDP algorithm in the case study, our MBSE tool-chain captures 14 tool operations (shown in Table V) in order to support automated parameter value selections (e.g., loading model in the simulation tool, generating FMUs). Using these tool operations, developers can implement their work automatically without any manual operations. Therefore, we can infer that our tool-chain supporting parameter value selection promotes efficiency for ADS development. Moreover, we also ensure that our MBSE tool-chain can improve the automation level of ADS development.

B. Qualitative Evaluation

The availability of the WPMS generation is validated by previous work [36]. In this article, we mainly focus on the
automated parameter value selection, adopting SM to select candidate parameter values for co-simulation configurations.

When the ADS developers design the MDP algorithm in the case study, our MBSE tool-chain captures 14 types of tool operations to realize tool integration between the six tools or data: 1) MetaEdit+; 2) WPMS; 3) MWorks; 4) Simulink; 5) Carmaker; and 6) FMU. The interoperability between these is obviously promoted using OSLC services and FMI. Without using these two standards, the number of developed interfaces between them increases, because the interfaces of all the tools are required to be developed otherwise. Using DSM based on the GOPPRR approach, metamodels are developed in a more flexible way. Compared with other DSM approaches, the GOPPRR approach was proposed as the most powerful approach [43]. The WPMS promotes the process management capabilities for the co-design process. Developers implement their work automatically on the WPMS without any manual operations, such as loading the model in the simulation tool or generating FMUs. At the same time, the OSLC services can promote traceability between development processes and technical resources, decreasing errors caused by mismatching models, which support consistency management between requirements, architecture, and parameters.

In order to promote the scalability, the GOPPRR approach is used to support MBSE with meta-metamodels which means more domain-specific metamodels and models can be developed. OSLC is a specification for data integration using linking data referring to a standardized specification for different design tools across the lifecycle. FMI is also a standard for co-simulation, which has been widely used for different simulation tools. Thus, these standardized specifications provide the potentials to extend to other scenarios with a good scalability.

Except for the evaluation in this article, a previous paper provided an analysis about the advantages of the proposed tool-chain by its visualizations. In [44], compared with model-driven tool-chains, the service-oriented tool-chain is verified to have the potentials to support more complex scenarios for system development. Moreover, through the OSLC services and BPM Camunda, stakeholders can implement their design jobs in a more centralized and automated way. Through the OSLC services, more data and models are managed and manipulated by the MBSE tool-chain. Therefore, we infer that our MBSE tool-chain can improve the effectiveness of ADS development.

C. Summary

From the quantitative and qualitative analysis, the motivations mentioned in Section II-C are evaluated as internal validity. First, domain-specific views of CPS are formalized based on metamodels and models using the GOPPRR approach. Moreover, the extended metamodels for automated parameter value selection describe the parameter configurations, which are the basis to implement the decision making. OSLC and FMI are two specifications for data exchange and co-simulation, which improve the data interoperability. In the case study, the MDP algorithm is evaluated by Simulink and co-simulation models. The parameter configurations are automatically executed through the whole tool-chain leading to the improvement of development efficiency. Thus, compared with the existing model-driven approaches, this article proposed a new MBSE tool-chain that supports decision making for parameter configurations based on process-driven and data-centric consistency management. The external validity of the tool-chain will be evaluated by more complex scenarios in the future.

D. Limitations

There are some limitations of the proposed approach in the case study. The scenario for developing AI algorithms of auto-braking systems is adopted for evaluating the MBSE tool-chain. In the scenario, only two vehicles and an infrastructure are used to create a testing environment. Moreover, the case study is mainly used to verify the parameter value selection using our MBSE tool-chain. A set of parameters are configured during simulations that can evaluate the expected goal of the tool-chain. Communications between two vehicles and the infrastructure during simulations and more complex scenarios are not considered.

This approach has several technical limitations.
1) The metamodels mentioned in this article need to be extended in order to formalize other CPS development and to support further automation.
2) The service compiler should support change management of technical resources and design information using OSLC services in the future. For example, each design change can be represented as one OSLC service for change management.
3) The decision-making algorithm is required to extend the capability of system architecture tradeoffs in order to satisfy the demands of more complex scenarios.

VI. Conclusion

We have proposed an MBSE tool-chain for automated parameter value selection based on a DSM approach, a service-oriented approach, and co-simulation whose availability is evaluated by a case study on auto-braking system development. When using the tool-chain, DSM models are used to generate a Web-based process management system where developers execute co-simulations through the OSLC services of technical models, data, and tools automatically. During simulation execution, a decision-making algorithm is proposed to support automated parameter value selections for the auto-braking system. The results demonstrate this tool-chain is well placed to promote the efficiency and effectiveness of CPS development, when implementing co-simulations.

REFERENCES

[1] E. A. Lee, “CPS foundations,” in Proc. 47th Design Autom. Conf. (DAC), 2010, pp. 737–742.
[2] O. L. De Weck, A. M. Ross, and D. H. Rhodes, “Investigating relationships and semantic sets amongst system lifecycle properties (illities),” in Proc. 3rd Int. Eng. Syst. Symp. CESUN, Jun. 2012, pp. 18–20.
[3] A. M. Madni and M. Sievers, “Model-based systems engineering: Motivation, current status, and research opportunities,” Syst. Eng., vol. 21, no. 3, pp. 172–190, 2018.
A. Blekhman, J. P. Wachs, and D. Dori, “Model-based system specification with tesseract: Readable text from formal graphics,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 11, pp. 1448–1458, Nov. 2015.

C. Qing, B. Cai, Q. Yang, J. Wang, and C. Huang, “Deep learning for CSI feedback based on superimposed coding,” *IEEE Access*, vol. 7, pp. 93723–93733, 2019.

M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Calenda, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 1527–1533.

Z. Chen, S. Guo, J. Wang, Y. Li, and Z. Lu, “Toward FPGA security in IoT: A new detection technique for hardware trojans,” *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7061–7068, Aug. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8710614/

R. Schilling and T. Schultz, “Validation of automated driving functions,” in *Simulation and Testing for Vehicle Technology*. Cham, Switzerland: Springer, 2016, pp. 377–381.

K. Vanherpen et al., “Ontological reasoning for consistency in the design of cyber-physical systems,” in *Proc. 1st Int. Workshop Cyber-Phys. Prod. Syst. (CPPS)*, Apr. 2016, pp. 1–8.

V. Singh and K. E. Willcox, “Engineering design with digital thread,” *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2134–2143, Apr. 2018.

J.-J. Chen, J. Lu, M. Törngren, and D. Gürdür, “An investigation of functionalities of future tool-chain for aerospace industry,” in *Proc. INCOSE Int. Symp.*, 2017, pp. 1408–1422.

S. A. Seshia, S. Hu, W. Li, and Q. Zhu, “Design automation of cyber-physical systems: Challenges, advances, and opportunities,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1413–1434, Sep. 2017.

J. Tavčar and I. Horváth, “A review of the principles of designing smart cyber-physical systems for run-time adaptation: Learned lessons and open issues,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 1, pp. 145–158, Jan. 2019.

N. Mohan et al., “Challenges in architcting fully automated driving: With an emphasis on heavy commercial vehicles,” in *Proc. Workshop OpenSource Archit., Apr.* 2018, pp. 2–9.

F. Tao and Q. Qi, “New IT driven service-oriented smart manufacturing: Framework and characteristics,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 1, pp. 81–91, Jan. 2019.

C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “DeepDriving: Learning affordance for direct perception in autonomous driving,” in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2722–2730.

A. L. Ramos, J. V. Ferreira, and J. Barceló, “LITHE: An agile methodology for human-centric model-based systems engineering,” *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 43, no. 3, pp. 504–521, May 2013.

S. Costa, P. F. Pires, and F. C. Delicato, “Modeling IoT applications with SysML4IoT,” in *Proc. 42nd EuroMicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, 2016, pp. 157–164.

M. Hause and J. Hummell, “Making smart cities smarter—MBSE driven IoT:” in *Proc. INCOSE Int. Symp.*, no. 26, vol. 1, pp. 1675–1690, Jul. 2016.

Y. Mordecai, O. Horbo, and D. Dori, “Model-based interoperability engineering in systems-of-systems and civil aviation,” *IEEE Trans. Man., Cybern., Syst.*, vol. 48, no. 4, pp. 637–648, Apr. 2018.

P. Fritzson and V. Engelson, “Modelica—a unified object-oriented language for system modeling and simulation,” in *Proc. 12th Eur. Conf. Object-Oriented Program. (ECOOP)*, 1445, 1998, pp. 67–90.

B. P. Derler, A. L. Lee, and A. S. Vincentelli, “Modeling cyber—Physical systems,” *Proc. IEEE*, vol. 100, no. 1, pp. 13–28, Jan. 2012.

P. G. Larsen et al., “Integreted tool chain for model-based design of cyber-physical systems: The INTO-CPS project,” in *Proc. 2nd Int. Workshop Model. Anal. Control Complex CPS (CPS Data)*, Apr. 2016, pp. 1–6.

C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, “Co-simulation: A survey,” *ACM Comput. Surveys*, vol. 51, no. 3, p. 49, May 2018.

“Functional mock-up interface for co-simulation,” *MODELS/ISAR*, Sweden, Rep., 2010. [Online]. Available: https://fmi-standard.org/

D. Status and D. Type, “Distributed co-simulation protocol (DCP),” *MODELLICA Association*, Sweden, Rep., 2019. [Online]. Available: https://fmi-standard.org/

M. Chernyshev, Z. Baig, O. Bello, and S. Sezaldy, “Internet of things (IoT): Research, simulators, and testbeds,” *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1637–1647, Jun. 2018.
Dejiu Chen (Senior Member, IEEE) received the Ph.D. degree in mechanical engineering from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2004 with a focus on embedded computer control systems. From 2007 to 2009, he was with Enea Data AB, Kista, Sweden, as a Senior Technical Instructor. He is currently an Associate Professor with the KTH Royal Institute of Technology. His main research areas include systems- and software-architecture design, model-based engineering, and advanced situation-awareness and self-adaptation for safety- and time-critical cyber-physical systems.

Guoxin Wang received the M.Sc. degree in mechanical engineering with a specialization in mechatronics from the KTH Royal Institute of Technology (KTH), Stockholm, Sweden, in 1987, the Licentiate of Technology degree from the Department of Machine Elements, KTH, in 1992, the Ph.D. degree from the Department of Machine Elements, KTH, in 1995, the Docent degree in machine design from KTH, in 1999, and the Professional Preparation: Executive School in Industrial Management in 2007. He is an Associate Professor and the Director of the Industrial Engineering Institute, Beijing Institute of Technology, Beijing, China. His research interests include reconfigurable manufacturing systems and knowledge-based engineering.

Dimitris Kiritsis received the Diploma degree in mechanical engineering and the Ph.D. degree in mechanical engineering from the University of Patras, Patras, Greece, in 1980 and 1987, respectively. He is a Faculty Member with the Institute of Mechanical Engineering, School of Engineering, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, where he is leading a research group on ICT for sustainable manufacturing. He also serves as the Director of the Doctoral program of EPFL on Robotics, Control, and Intelligent Systems (EDRS). His research interests are closed-loop lifecycle management, IoT, semantic technologies, and data analytics for engineering applications.

Martin Törngren (Senior Member, IEEE) has been a Professor of Embedded Control Systems with the Mechatronics Division, Department of Machine Design, KTH Royal Institute of Technology, Stockholm, Sweden, since 2002. He has authored/coauthored more than 100 peer-reviewed publications, and also been in charge of developing and leading graduate and continued education courses. He has particular interest in cyber-physical systems, model-based engineering, architectural design, systems integration, and co-design of control applications and embedded systems.