Design Ontology in a Case Study for Cosimulation in a Model-Based Systems Engineering Tool-Chain

Jinzhi Lu, Guoxin Wang, and Martin Törngren

***Abstract*—Cosimulation is an important system-level verification approach aimed at integrating multidomain and multi-physics models during complex system development. Currently, the lack of integrating system development process with cosimulations leads to gaps between them, decreasing the effectiveness and efficiency of system development. Model-based systems engineering (MBSE) tool-chains have been proposed to facilitate the integration of complex system development and automated verification using a model-based approach. However, due to the lack of formal and structured specifications, development information sharing is difficult for supporting MBSE facilitating automated cosimulations. In order to formalize cosimulation in an MBSE tool-chain, a scenario-based ontology is developed in this paper, using formal web ontology language (OWL). Ontology refers to a specification expressing the cosimulation implementations as well as the development information represented in the models supporting the MBSE. It is illustrated by a case study of a cosimulation based on Simulink. Protocol and resource description framework (RDF) query language (SPARQL) and semantic query-enhanced web rule language queries are proposed for evaluating the ontology's completeness and logic for supporting cosimulations. The result demonstrates that the scenario-based ontology formalizes the information related to automated cosimulation development and configurations while using the proposed MBSE tool-chain.**

***Index Terms*—Cosimulation, model-based systems engineering (MBSE), ontology design, simulation automation, tool-chain.**

I. Introduction

COSIMULATION aims to integrate heterogeneous multidomains and multi physics models for supporting system-level verification. Developers in different domains use their own tools to build specific domain models. System engineers then integrate such models to predict the global behaviors of the entire system. For such complex system development, cosimulation serves as an important verification approach for supporting concurrent and collaborative development. Each cosimulation scenario implements the verification activities related to the development process and system artifacts. Therefore, well-managed configuration and change managements among cosimulation, development process, and system artifacts promote the effectiveness and efficiency of complex system development.

However, generation of cosimulation for complex system development is difficult because of complicated development and configurations across various domains. For example, interface definitions between different developers need to be consistent before integrating the models. Moreover, the interrelationship management of technical resources (data, models, tool application programming interfaces (APIs), and codes) is challenging for simulation automation. For example, the automated development of cosimulations requires well-managed traceability between different subsystems and cosimulation models. Furthermore, without unified representations of technical resources, integrating the descriptions of the related cosimulations is difficult. Existing techniques are limited in promoting tool interoperability for cosimulation, leading to difficulties in managing the tools and data without a specifically developed and integrated platform. Finally, without support of process and change management, cosimulation automation by integrating the development process with detailed cosimulation operations is challenged.

The main contribution of this paper is the design of a scenario-based ontology to support cosimulation automation in a model-based systems engineering (MBSE) tool-chain, which has been proposed for integrating technical resources, system development, and system artifacts using a service-oriented approach. The ontology design aims to promote the traceability and interoperability among technical resources, and effectiveness and efficiency of cosimulation development and configuration in different cosimulation scenarios, which are as follows. 1) The scenario-based ontology aims to formalize the interrelationships between related services supporting different types of cosimulations. 2) The scenario-based ontology describes configuration and deployment of the related services for formalizing cosimulation implementations based on the scenarios.

Totally, the scenario-based ontology is proposed for formalizing cosimulation as specifications for supporting service orchestration of technical resources and automated cosimulation execution in the proposed MBSE tool-chain.

In order to design the ontology, we propose a systems thinking approach to define its compositions and their interrelationships in different scenarios. Following four aspects are considered with respect to the MBSE tool-chain: 1) Open Services for Lifecycle Collaboration (OSLC) services of domain-specific modeling (DSM) models; 2) OSLC services of technical resources; 3) Service orchestrations; 4) A web-based process management system (WPMS) in the MBSE tool-chain.

Moreover, the ontology concepts are developed, based on formal web ontology language (OWL) in Protégé, referring to an ontology modeling tool [1]. Through a case study on the MBSE tool-chain, the completeness and logic of the ontology are evaluated using protocol and resource description framework (RDF) query language (SPARQL) [2] and semantic query-enhanced web rule language (SQWRL) queries [3].

Manuscript received August 31, 2018; revised January 1, 2019 and February 21, 2019; accepted April 11, 2019. Date of publication June 11, 2019; date of current version March 2, 2020. This work was supported by the China National Ministries (JCKY2014602B007). *(Corresponding author: Guoxin Wang.)*

J. Lu and M. Törngren are with the KTH Royal Institute of Technology, 10044 Stockholm, Sweden (e-mail: jinzhl@kth.se; martint@kth.se).

G. Wang is with the Beijing Institute of Technology, Beijing 100081, China (e-mail: wangguoxin@bit.edu.cn).

Digital Object Identifier 10.1109/JSYST.2019.2911418

The rest of this paper is organized as follows. We discuss the related work in Section II and present the problem statements in Section III. In Section IV, we introduce our research methodology. In Section V, a scenario-based ontology is proposed in detail. We utilize a case study to instantiate the ontology and evaluate its completeness and logic using SPARQL and SQWRL in Section VI. Finally, the conclusions are offered in Section VII.

II. Related Work

Various aspects supporting ontology design have been researched till date, including cosimulation taxonomies, ontology design for simulation automation, and service orchestration.

A. Cosimulation Formalisms

There are various taxonomies for defining cosimulation scenarios. Based on Gomes's taxonomy, cosimulations include discrete-event, continuous-time, and hybrid cosimulations [5]. The taxonomy is classified based on the cosimulation models. Local and distributed cosimulations are classified based on the geographical distributions [6]. Depending on the interfaces, standard-based and private-based cosimulations are defined based on whether they are developed as per the functional mocked-up interface (FMI) specification [7]. Coupling-solver and separated-solver cosimulations are defined based on whether the solvers are involved with the interface files [8]. Hardware-in-loop and software-in-loop simulations are two types of cosimulation supporting real-time system tests [9]. These taxonomies are basics to develop unified formalisms for supporting automated cosimulations in different scenarios. Without such unified formalisms, it is difficult to develop a general platform for managing and implementing different cosimulations.

B. Ontology Design for Supporting Simulation Automation

Ontology refers to content about the sorts of objects, their properties, and relationships, which represent domain-specific knowledge [10]. In the MBSE domain, basic foundation ontologies are mainly used for formalizing system information of system development and artifacts, such as Jet Propulsion Laboratory developed ontology based on OWL for SysML models [11]. Except for basic foundation ontologies, domain-specific ontology is proposed for representing the information of specific domains, such as ontology is developed for space system development [12].

In this paper, we focus on the cosimulation platform development, which researchers have proposed, ontologies for supporting information exchange. Miller and Baramidze [13] proposed an initial framework to formalize modeling and simulation based on the OWL, which was adopted to formalize different models and simulation types. Grogan and Weck [14] proposed an ISoS modeling framework for constructing system simulation infrastructure for system of systems (SoS) where a scenario-based ontology formalizes all its elements, based on mathematical theories and interfaces. Soyez *et al.* [15] proposed formalisms for agent representations, during modeling SoS. Formalisms refer to ontologies that describe the entities in the SoS for computing in each agent. Most of the formalisms are used for representing the entities of the systems themselves rather than the simulation frameworks. Therefore, such formalisms lack the capability for supporting automated operations in a cosimulation platform.

Some ontologies were developed to support simulation automation. Benjamin *et al.* [16] proposed an ontology-driven approach to support simulation automation. The approach integrated modeling, simulation, scheduling analysis, and optimization for describing the related information in a unified manner. Other than focusing on modeling and the simulations themselves, ontologies were developed to support simulations in the product lifecycle. Sirin *et al.* [17] proposed a model identity card for developing models and simulation in a collaborative design process. The model identity cards refer to one ontology for information exchange during product development. Ming *et al.* [18] proposed an ontology for reusable and executable decision templates. Such ontology supports automated design makings during engineering design, based on the defined unities related to different views. Although such ontologies are used in IT systems for simulation automation, they cannot describe tool operations in cosimulation scenarios.

Furthermore, some researchers design ontology focusing on cosimulation techniques. Silver *et al.* [19] proposed an ontology describing discrete-event models. Others proposed formalisms to support cosimulation based on FMI, which is a standard for defining cosimulation interfaces [6]. Gomes *et al.* [20] proposed semantic models for describing cosimulation based on the FMI. These models describe the cosimulation activities between different functional mock-up units (FMUs) and a master engine. Zeyda *et al.* [21] formalized a master algorithm for cosimulation. Other than such formalisms, Lu *et al.* [7] proposed a DSM language to formalize cosimulation tool-chains representing its compositions and relationships. These formalisms describe the cosimulations based on different views, particularly cosimulation itself and related tool-chains. However, lack of integrating cosimulations with product development and system artifacts leads to difficulties for automated implementations.

C. Ontology Design for Service Orchestration Supporting Cosimulation

Currently, service-oriented approaches are presented for dealing with tool interoperability during cosimulation [22]. The developed services, representing technical resources (e.g., models, APIs, data), are accessed by other tools or stakeholders using related uniform resource identifiers (URIs). Service orchestration refers to one technique for manipulating such services in order to support related operations. Through literature review, we found several frameworks that were proposed for defining, verifying, and deploying service orchestration in the software engineering domains. Foster *et al.* [23] proposed a model-based approach to support the automated deployment of service orchestrations. The models first formalize the service orchestrations and are then used to verify the service compositions based on the defined needs. Labidi *et al.* [24] proposed an approach to define ontology-based service-level agreements for cloud computing. This agreement include negotiations for generating ontology for cloud services and renegotiation for reconfiguring cloud services.

Compared to the software engineering domains, service-oriented approaches have been used in certain scenarios to support product development aiming to facilitate data exchange and to promote tool interoperability. Biehl [25] proposed a model-based approach to formalize tool-chains and to construct service-oriented tool-chains based on OSLC services. OSLC is an open community for creating specifications to support tool-integration that enable conforming independent software and tools to integrate their data, control, process, and presentation during the entire life cycle. In the tool-chain, technical resources, such as data and models, are presented as restful services accessed by other tools. Wu *et al.* [26] proposed a service-oriented platform for feature-based data exchange during product design and manufacturing. Based on these two cases, we found that technical resources were accessed by the developed services manipulated by service orchestrations for supporting the automated implementations of the related activities.

D. Summary

Based on literature reviews, the motivations for this paper are summarized as follows. 1) From Section II-A, different scenarios related to cosimulation are proposed. Therefore, a scenario-based approach is used to define the ontology for formalizing cosimulation implementations in the MBSE tool-chains. Based on different scenarios, ontology defines their implementations with required technical resources and development information. 2) From Section II-B, compared with current researchers from academia and industry, the scenario-based ontology needs not only to formalize the development process and system artifacts, but also to support process management and tool operations, particularly supporting cosimulation automation. 3) From Section II-C, a service-oriented approach is supported by the service orchestrations to manage the interrelationships between services for implementing cosimulations. Service orchestration is defined as the operational and functional execution processes involved in the design, creation, and delivery of end-to-end OSLC services for supporting cosimulation automation. Such service orchestrations are implemented based on the scenario-based ontology describing the entities and their end-to-end relationships using a first-order logic. Therefore, OWL is adopted for formalizing the related ontology concepts in this paper.

In total, we focus on ontology design to support cosimulation automation in an MBSE tool-chain, which involves the adoption of DSM models to formalize the development process, system artifacts, and cosimulation operations and of a service-oriented approach to generate OSLC services representing the DSM models and technical resources (model, data, and tool APIs). Moreover, the DSM models are transformed into a WPMS linked with OSLC services of the related development information and technical resources where stakeholders manipulate the cosimulations without any manual operations. The ontology is designed using OWL based on scenarios defined by systems thinking that aims to support service orchestration for the cosimulation implementations.

III. Problem Statement

A. Overview of the MBSE Tool-Chain

Fig. 1 [39] presents an MBSE tool-chain for supporting aero-engine co-design. The MBSE tool-chain is used to implement aero-engine simulation automatically [22]. Stakeholders make use of DSM models to formalize the development process and system artifacts. Then, the DSM models are transformed to one WPMS to deploy technical resources and to implement automated cosimulations in Simulink for different stakeholders.

In details, the DSM models are built based on developed meta-models describing the development process, system artifacts, and related cosimulation operations. The meta-models are developed based on Graph, Object, Port, Property, Relationship, and Role (GOPPRR) approach (introduced in

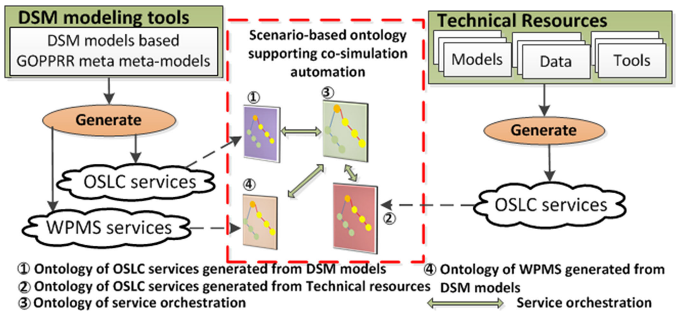

DSM modeling tools
DSM models based GOPPRR meta meta-models
Generate
OSLC services
WPMS services
Scenario-based ontology supporting co-simulation automation
①
②
③
④
Technical Resources
Models
Data
Tools
Generate
OSLC services
① Ontology of OSLC services generated from DSM models
② Ontology of OSLC services generated from Technical resources
③ Ontology of service orchestration
④ Ontology of WPMS generated from DSM models
Service orchestration

Fig. 1. Scenario-based ontology supporting cosimulation automation.

Section V-A) [28]. We adopt this approach as our MBSE solution, because GOPPRR is evaluated as one of the most powerfully expressive meta-meta models compared to the other DSM approach [29]. The DSM models include process (see [22, Fig. 2]) and information [22] patterns: The process pattern is used for formalizing the cosimulation development process, whereas the information pattern is used for representing the related information in each work task. Using the code-generators in the DSM tools, the DSM models are transformed into ontology for describing the process and information patterns.

Then, OSLC adapters are developed for transforming the ontology and technical resources (e.g., models, tools, and data) to OSLC entities including service providers and services. In addition, the ontology (process pattern) is also loaded by the compiler to generate a WPMS where the work tasks are linked to the OSLC service providers for each work task. The required OSLC services of DSM models and technical resources are linked to the OSLC service providers through service orchestration templates (SoTs). Finally, the developers log into their own accounts in the WPMS and access the required information and technical resources in each work task through the related OSLC services. Using the WPMS, they implement the cosimulations automatically without any manual operations.

B. Scenario-Based Ontology Supporting Service Orchestration for Cosimulation Automation

In the MBSE tool-chain, the WPMS is linked to the required OSLC service providers and services representing the DSM models and technical resources where developers access and manipulate the technical resources. In order to orchestrate the related OSLC entities in the WPMS, a scenario-based ontology is proposed for formalizing the MBSE tool-chain for supporting cosimulation automation, as shown in Fig. 1. It depicts the complete representation and logic of the cosimulation development and configuration, which includes following four aspects. 1) Ontology representing the OSLC services generated from the DSM models; this is used to formalize the development process, system artifacts, and cosimulation implementations in DSM models. 2) Ontology representing the OSLC services representing technical resources; this is used to formalize the OSLC services for manipulating and accessing the technical resources. 3) Ontology representing the SoTs; this aims to link the development process to cosimulation implementations. 4) Ontology representing the WPMS generated by the DSM models; this represents the WPMS for implementing the development processes linked with the development information and cosimulation configurations.

Based on these four aspects, cosimulation scenarios are defined for service orchestrations in the MBSE tool-chain.

IV. Research Methodology

In order to support cosimulation automation in different scenarios, we develop a ontology based on OWL using systems thinking and evaluate the ontology concepts using a case study. These ontology concepts serve as formal representations of the development process, system artifacts, and technical resources, and as a descriptive logic for tool operations to support cosimulation automation in the proposed MBSE tool-chain. The formal representations are used for sharing information as middle-ware in the tool-chain. The descriptive logic is used to orchestrate the required technical resources based on a first-order logic during the development and configuration of the cosimulation [30].

A. Authoring the Ontology Concepts Based on OWL in Protégé

The ontology concepts, including the entities and relationships, are defined by OWL based on the *RDF triple* (*subject-predicate-object*). While developing the ontology concepts, the *subject* and *object* indicating OWL *classes* are two entities linked by *predicates* referring to OWL *object properties*. The instances of the entities are OWL *individuals*. The attribute of the *subject* (*object*) is defined as a *data property* with a *data type*, e.g., string. The *data property assertions* define the rules that *data property* satisfies the *data type* for checking the consistency and reasoning. The ontology concepts are defined in *Protégé*, a graphical modeling tool based on OWL where the ontology concepts are developed for cosimulation scenarios defined in accordance with systems thinking.

B. Systems Thinking

Systems thinking is an approach for understanding systems by examining the interactions between the components within the system boundary [31]. In this paper, systems thinking is proposed for analyzing the compositions in cosimulation scenarios for ontology design.

1) *Identify the cosimulation scenarios:* The scenarios are defined in order to understand the scope of the contents (systems boundary) of the related ontology concepts. Each scenario is used to define one operational process for cosimulation, e.g., configuring simulation settings. 2) *Define the entities related to the scenarios:* In each scenario, the related entities are captured, such as the required *tool* and *model* for creating models. Such entities are used for representing the *subject* and *object* in the *RDF triple* of OWL [2]. 3) *Define the relationships between entities:* Relationships between entities refer to connections between entities. Such relationships are defined as the object properties in OWL, such as a *development process includes* several *work tasks*. 4) *Instantiate the entities using a case study:* Based on the designed ontology, a case study is proposed for evaluating the ontology concepts for cosimulation scenarios. In Protégé, OWL *individuals* are used to formalize the instances of the entities in the case study. 5) *Evaluation of the case study:* Through the case study, SPARQL and SQWRL queries are proposed for evaluating the ontology (introduced in Section IV-C).

C. Evaluation Based on Case Study

A case study is proposed to define a cosimulation scenario based on MATLAB/Simulink [32] for evaluating the completeness and logic of the developed ontology for service orchestration. In the case study, following two measurements are considered to evaluate the ontology's completeness and logic for service orchestration in order to promote effectiveness and efficiency of cosimulation using automatically as shown in Table I.

TABLE I
Measurement and Metrics for Ontology Evaluation

Measurement	Metrics	Algorithm	Evaluation
Completeness	Graph-include-objects	Algorithm 1	SPARQL
	Object(Relationship)-include-points		
	Object(graph and relationship)-include -properties		
Logic representing DSM models	Model-structure	Algorithm 2	SQWRL
	Process-sequence		
Logic presenting automated tool operations	Open & Close tool	Algorithm 3	
	Open & Save & Close models		
	Add component		
	Add FMU		
	Add model connectors		

1) *The Completeness of the Representing the OSLC Services Generated From DSM Models:* SPARQL is a query language to evaluate whether the ontology represents the OSLC services generated from the DSM models completely. In order to support this measurement, several metrics are defined, which are as follows.

a) *Graph-include-Objects (Relationships):* refers to a situation, where one *Graph* OSLC service or service provider (introduced in Section V-A) includes all the information related to the OSLC services of its *Object* (or *Relationships*). In the case study, Algorithm 1 is used to verify if one *Graph* OSLC service is linked to OSLC services of several objects and relationships.

b) *Object (Relationship)-include-Points (Roles):* refers to a situation, where one *Object* (or *Relationship*) OSLC service includes all the information related to the OSLC services of its Points (Roles). In the case study, Algorithm 1 is used to verify if one *Object (Relationship)* OSLC service is linked to several *Points (Roles)* OSLC services.

c) *Object (Graph and Relationship)-include-Properties:* refers to a situation, where one *Object* (*Graph* and *Relationship*) OSLC service includes all the information related to the OSLC services of its Property. In the case study, Algorithm 1 is used to verify if one *Object (Graph and Relationship)* OSLC service is linked to OSLC services representing its *Properties*.

Algorithm 1 based on SPARQL is used to query the relationships between OSLC services of meta-models in order to verify the metrics.

2) *OSLC-Service Logic Related to Cosimulation Implementation:* SQWRL is a query language combining OWL, DL, and SWRL for designing the rules to assign the *subject*, *predicate*, and *object* based on the defined *predicates* [3]. It is adopted to evaluate whether the ontology captures the information in the OSLC services to implement the cosimulation automatically. Following two metrics are considered.

a) *The logic representing the OSLC services from the DSM models:* indicates whether the ontology can describe the logic in the DSM models. Two main aspects are verified in Algorithm 2: first, *model-structure* referring to the model structures of the cosimulation models; and second, *process-sequence* referring to the sequence orders in the development process.

b) *The logic of the OSLC services supporting automated tool operations:* indicates whether the ontology can describe

Algorithm 1: SPARQL Algorithm for Completeness Evaluation.

```
// Query objects in graphs
SELECT ?Graph ?Object
WHERE {
// $Graph\_decomposite\_service$ refers to decomposite in GOPPRR approach.
?Graph
$foaf : Graph\_Including\_Relationship\_linking\_service$
?Object.
}
// Query WPMS
SELECT ?Process ?Worktask
WHERE {
// $ProcessHasEntities$ refers to an object property, where one process has entities.
?Graph $foaf : ProcessHasEntities$ ?Object.
}
// Query points of model components and connectors (the same as querying properties)
SELECT ?ModelComponent ?ModelConnector ?Point ?Role
WHERE {
// $foaf : Object\_Has\_point\_service$ refers to an object property, where one object has points.
// $foaf : Relationship\_To\_Input(Output)\_service$ refers to an object property, where one relationship has one or more input(output) roles.
?ModelComponent $foaf : Object\_Has\_point\_service$
?Point.
?ModelConnector
$foaf : Relationship\_From\_Output\_service$ ?Role.
?ModelConnector $foaf : Relationship\_To\_Input\_service$
?Role.
}
```

the logic for automated tool operations, such as the SoT for creating model that is verified by Algorithm 3.

V. Ontology Supporting Cosimulation in an MBSE Tool-Chain

In this section, we introduce the designed ontology (Fig. 1). First, the ontology representing the OSLC services and service providers from the DSM models, based on the GOPPRR approach, are introduced. Then, the ontology that supports the OSLC services formalizing the technical resource is proposed. In addition, the ontology for the OSLC services of the SoT is defined using systems thinking based on four types of scenarios, including development scenario; configuration scenario; execution scenario; and error scenario. Finally, the ontology supporting the WPMS generated from the DSM models is illustrated.

A. Ontology Formalizing the OSLC Services Based on the GOPPRR Approach

The GOPPRR approach is adopted to define and develop meta-models for constructing the DSM models. The GOPPRR

Algorithm 2: SQWRL Algorithm for Checking the Logic of DSM Models.

```
// Query input points of relationships
$ModelConnectorService(?Relationshipinstal)$
$\Lambda ModelComponentObjectService(?modelcomponent)$
$\Lambda Model\_Component\_InportService$
$(?ObjectinputInstal)$
$\Lambda Object\_Has\_point\_service(?modelcomponent,$
$?ObjectinputInstal)$
$\Lambda RoleConnectingPoint\_service(?ToRole,$
$?ObjectinputInstal)$
$\Lambda Relationship\_ToRole\_service(?Relationshipinstal,$
$?ToRole)$
$- > sqwrl : select(?Relationshipinstal,$
$?modelcomponent, ?ObjectinputInstal)$
// Query process sequences in the WPMS
$SI\_WPMS\_RelaBetweenPoint\_Sequence(?Seq)$
$\Lambda\ SequenceRelationshipFrom(?Seq, ?From)$
$\Lambda\ SequenceRelationshipTo(?Seq, ?to)$
$- > sqwrl : select(?Seq, ?From, ?to)$
```

meta-meta models include Graph, Object, Point, Property, Role, and Relationship, shown as follows.

1) *Graph* is a collection of *Object*, *Relationship*, and *Role* represented as one window (one integrated concept of a class diagram and package in unified modeling language (UML)). The graph can be a visual diagram on the top level or lower level decomposed by one *Object*.
2) *Object* is one entity in *Graphs* (class concepts in UML).
3) *Point* is a port in *Objects*.
4) *Relationship* is one connection between the different *Points* of *Objects*.
5) *Role* is used to define the connection rules mirrored to the relevant *Relationship*. For example, one *Relationship* has two *Roles*. Each is defined to connect with one *Point* in *Objects*. Then, the *Relationship* is connected with these *Points* in the *Objects*.
6) *Property* refers to one attribute of the other five meta-meta models.

Based on these concepts, ontology formalizing GOPPRR approach is developed based on OWL [33]. Then, the DSM model concepts are transformed into the related OSLC services through such ontology, as shown in Fig. 1. In order to define the ontology that formalizes the OSLC services of the DSM models, several transformation rules from the OSLC service providers and services of the GOPPRR meta-meta models to OWL concepts are defined as follows. 1) The OSLC services of *Graph*, *Object*, *Point*, *Property*, *Relationship*, and *Role* are transformed into the *Class* concepts in *OWL*. 2) The OSLC services of the meta-meta model concepts, other than the GOPPRR concepts, are defined as *Object properties* in OWL, such as the *Graph-including-Objects* refers to one OSLC service representing one *Graph*, including *Objects*. The details of *Object properties* are listed in Table II, which are read as that entities in the row do some entities in the column, such as $O\_graph_a^{id}$ *include* $O\_object_b^{id}$.[1] 3) The property values in the OSLC services are defined as data property assertions. The *Property* OSLC service links to the related OSLC entity of the data property assertions in the

[1]The following tables are the same.

Algorithm 3: SQWRL Algorithm for Checking the Logic for Create-Models.

```
// Query open & close tool
$Create\_cosimulation\_model\_ST(?create(close)$
$ModelTp1)$
$\Lambda\ ToolService(?Tool)$
$\Lambda 1\_1\_open(close)\_tool\_service(?open(close)Tool)$
$\Lambda\ cmTp\_implement\_Open(Close)$
$ToolAPI(?createModelTp1, ?Open(Close)Tool)$
$\Lambda\ Open/CloseToolAPI\_linking_{Tool}$
$(?Open(Close)Tool, ?Tool)$
$->sqwrl : select(?createModelTp1, ?OpenTool, ?Tool)$
// Query open&save&close model APIs
$Create\_cosimulation\_model\_ST(?createModelTp1)$
$\Lambda\ SM\_service(?Model)$
$\Lambda\ 2\_5(3/2)\_create(save/close)\_SM\_service$
$(?createModel)$
$\Lambda\ cmTp\_implement\_Create(save/close)ModelAPI$
$(?createModelTp1, ?create(save/close)Model)$
$\Lambda\ Create/Open/Save/Close\_model\_linking\_sm$
$(?createModel, ?Model)$
$->sqwrl : select(?createModelTp1,$
$?Model, ?Open(save/close)Tool)$
// Query add model component
$SM\_component\_service(?modelComponent)$
$\Lambda\ 3\_1\_add\_SM\_component\_service(?addComponent)$
$\Lambda\ Add/Delete\_component\_linking\_SM\_component\_$
$service(\ ?addComponent, ?modelComponent)$
$->sqwrl : select(?Model, ?modelComponent)$
// Query add FMU interface
$SM\_Interface\_service(?Interface)$
$\Lambda\ 5\_1\_insert\_SM\_interface_service(?addInterface)$
$\Lambda\ Add/Delete\_interface\_linking\_SM\_$
$interface_service\ (?addInterface, ?Interface)$
$->sqwrl : select(?Model, ?Interface)$
// Query add model connectors
$SM_connector\_outport\_service(?ConnectorOutput)$
$\Lambda\ SM_connector\_inport\_service(?ConnectorInput)$
$\Lambda\ SM\_component\_inport\_service(?ComponentInput)$
$\Lambda\ SM_component\_outport\_service$
$(?ComponentOutput)$
$\Lambda\ 4\_1\_add\_SM\_connector\_service(?addConnector)$
$\Lambda\ SM\_connector\_service(?SMConnector)$
$\Lambda\ Add/Delete\_connector\_linking\_SM\_connector$
$(?addConnector, ?SMConnector)$
$\Lambda\ SM\_ConnectorInput\_linking\_ComponentOutput($
$?ConnectorInput, ?ComponentOutput)$
$\Lambda\ SM\_ConnectorOutput\_linking\_ComponentInput($
$?ConnectorOutput, ?ComponentInput)$
$->sqwrl : select(?Model, ?SMConnector,$
$?ConnectorInput, ?ConnectorOutput)$
```

TABLE II
OBJECT PROPERTIES AMONG THE O_DSM (DETAILS IN [33])

$\Rightarrow$	$O\_graph_a^{id}$	$O\_object_b^{id}$	$O\_relationship_d^{id}$	$O\_role_c^{id}$	$O\_point_e^{id}$
$O\_graph_a^{id}$		decomposite explosion	explosion	explosion	
$O\_object_b^{id}$	include				
$O\_relationship_d^{id}$	include				
$O\_role_c^{id}$			startFrom endTo		
$O\_point_e^{id}$		include		connect	
$O\_property_f^{id}$	include	include	include	include	include

data property OSLC service. For example, one OSLC service of *Property Tool name* links to a data property OSLC service *Simulink* with assertions "Simulink."

Therefore, the required entities for the scenario-based ontology in Fig. 1 are defined as follows.

Definition 1: Token *::=* refers to a collection of ontology concepts. Token *:* refers to all the options of one variable. $O\_i_a^b$ refers to an ontology concept of OSLC service *i*. The *b* refers to variables of the ontology concepts; *a* (which can be defined as the name of O_i) referring to the ath O_i $(0 < a < n)$, where *n* is the total number of ontology concepts.

Definition 2: Token $\Rightarrow$ refers to the linking between two OSLC services, in this case, the $O\_property$ and the related object property assertions

$$O\_property \Rightarrow objectProperty^t \tag{1}$$

where *objectProperty* is the object property, and *t* refers to the type of object property

$$O\_dsm ::= \{O\_graph_a^{(id,t)}, O\_object_b^{id}, O\_role_c^{id}, O\_relationship_d^{id}, O\_point_e^{id}, O\_property_f^{id}\} \tag{2}$$

where O_dsm refers to the ontology concepts of the OSLC services and service providers generated from the DSM models. *id* refers to the instance types of ontology concepts (metamodel), such as the SysML requirement diagram (one metamodel based on Graph), which is an instance type of *Graph*. For example, $O\_graph_a^{(id,t)}$ refers to the OSLC services or OSLC service providers generated from the related *Graphs*; *id* refers to its *Graph* instance, such as the *Worktask Graph* (see Section VI); *t* indicates whether the related *Graph* is transformed into OSLC service (*s*) or service provider (*sp*). For example, $O\_graph_{Worktask1}^{(Worktask,sp)}$ refers to one OSLC service provider generated from *Worktask1 Graph* in the DSM models. $O\_object_b^{id}$, $O\_role_c^{id}$, $O\_relationship_d^{id}$, $O\_point_e^{id}$, and $O\_property_f^{id}$ are the OSLC services of the related DSM model concepts.

B. Ontology Formalizing Technical Resources

In Fig. 1, technical resources supporting cosimulation are formalized, which include the simulation models, FMUs, tools, tool APIs, data files, and entities in each simulation model. Therefore, the ontology of the OSLC services representing the technical resources is as follows:

$$O\_ts ::= \{O\_sm_a, O\_fmu_b, O\_tool_c^{tT}, O\_toolAPI_d^{(apiT,tT)}, O\_Datafiles_e^{dfT}, O\_EntitiesInModel_a\} \tag{3}$$

where *O_ts* refers to an OSLC service collection transformed from the technical resources; O_sm_a refers to the ontology of the OSLC service generated from the ath simulation model; O_fmu_b refers to the ontology of the OSLC service generated from the bth FMU; $O\_tool_c^{tT}$ refers to the ontology of the OSLC service related to the cth tool; *tT* refers to the following tool types: first, master (M), referring to a simulation tool for manipulating other tools during cosimulation; and second, slave (S), referring to a simulation tool controlled by the master. $O\_toolAPI_d^{(apiT,tT)}$ is the ontology of the OSLC service

TABLE III
TOOL APIs TO SUPPORT COSIMULATIONS

Id	Tool APIs	Tool type
1.1/1.2	Open/Close tools	M&S
2.1;2.2;2.3; 2.4;2.5	Open/Close/Save/Delete/Create simulation models	M&S
3.1;3.2	Add/delete model components	M&S
4.1;4.2	Add/delete model connectors	M&S
5.1;5.2	Add/delete FMU interface	M&S
6.1;6.2	Load/create simulation result files	M&S
7	Configure simulation model parameters	M&S
8	Configure interfaces	M&S
9	Configure simulation configurations	M&S
10.1/10.2	Error report of executions/ interface mismatch	M&S
11	Generate FMUs in simulation models	S
12	Generate FMU interface files	M
13	Trigger simulation executions	M

TABLE IV
RELATIONSHIPS (OBJECT PROPERTIES) BETWEEN THE OSLC SERVICES GENERATED FROM THE TECHNICAL RESOURCES

$\Rightarrow$	O_sm_a	O_fmu_b	$O\_tool_c^{tT}$
O_sm_a			*implement*
O_fmu_b	*implementedWith*		*generate*
$O\_tool_c^{tT}$	*implementedIn*		*have*
$O\_toolAPI_d^{(apiT,tT)}$			*ImplementedIn*
$O\_Datafiles_e^{dfT}$	*linkTo*	*linkTo*	*implement*

mirrored to the dth tool API, where *apiT* refers to one of the API types illustrated in Table III. $O\_Datafiles_e^{dfT}$ refers to the ontology of the OSLC service generated from the eth data file, where *dfT* refers to the two types of data files: first, interface files for cosimulation (IF); and second, result files for cosimulation (RF). $O\_EntitiesInModel_a$ is a collection of ontology concepts of the OSLC services generated from the entities in the ath simulation model. The object properties of the ontology representing the OSLC services generated from the technical resources are illustrated in Table IV.

The ontology of the OSLC services representing the entities in each simulation model are defined as follows:

$$\begin{aligned} &O\_EntitiesInModel_a \\ &\qquad ::= \{O\_smComponent_b^{type}, O\_smConnector_c, \\ &\qquad\quad O\_smConfiguration_d, O\_smParameter_e, \\ &\qquad\quad O\_smPoint_f^{Type}, O\_smResult_g\} \end{aligned} \tag{4}$$

where $O\_smComponent_b^{type}$ is the ontology of the OSLC services representing the bth model component (MC), including two *types*: first, MC and second, FMU interface (FI). $O\_smConnector_c$ is the ontology of the OSLC services transformed from the cth model connector. $O\_smConfiguration_d$ is the ontology of the OSLC service mirrored to the dth simulation configuration; $O\_smParameter_e$ is the ontology of the OSLC service mirrored to the eth simulation parameter; $O\_smResult_g$ is the ontology concept of the OSLC services from the gth simulation result; $O\_smPoint_f^{Type}$ refers to the fth point in the MCs, which includes four types: Connector input; connector output; component input; and component output. The *object properties* between O_sm_a and $O\_EntitiesInModel_a$ are illustrated in Table V.

TABLE V
OBJECT PROPERTIES BETWEEN *O_sm* AND ITS ENTITIES

$\Rightarrow$	O_sm_a	$O\_smComponent_a^{Type}$	$O\_smConnector_b$
$O\_smComponent_a^{Type}$	*include*		
$O\_smConnector_b$	*include*		
$O\_smConfiguration_c$	*have*		
$O\_smParameter_d$		*have*	
$O\_smPoint_e^{Type}$		*have*	*have*
$O\_smResult_f$	*have*		

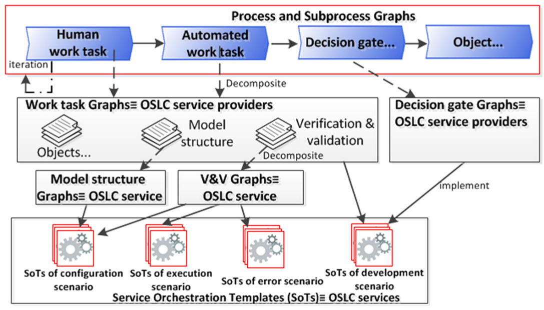

Process and Subprocess Graphs
Human work task
Automated work task
Decision gate...
Object...
iteration
Decomposite
Work task Graphs≡ OSLC service providers
Objects...
Model structure
Verification & validation
Decomposite
Decision gate Graphs≡ OSLC service providers
Model structure Graphs≡ OSLC service
V&V Graphs≡ OSLC service
implement
SoTs of configuration scenario
SoTs of execution scenario
SoTs of error scenario
SoTs of development scenario
Service Orchestration Templates (SoTs)≡ OSLC services

Fig. 2. Development process and subscenarios of the work task in a development scenario.

C. Ontology Formalizing the SoTs Supporting Cosimulation Scenarios

DSM models are adopted for formalizing the development processes and system artifacts, e.g., requirement. After model transformations, a WPMS linked with OSLC entities representing DSM models and technical resources is generated. To support cosimulation automation through WPMS, several linkings among the WPMS, OSLC services, and service providers are defined as shown in Fig. 3, which are:

1) Linking the human work task (HWT) and automated work task (AWT) in the WPMS to the related OSLC service providers mirrored to the work task *Graphs* in the DSM models (see Section VI).
2) Linking OSLC services of the model structure and verification & validation *Objects* in the work task *Graph* to the related OSLC services representing the model structure and verification & validation *Graphs*.
3) Linking the decision gates in the WPMS to the OSLC service provider of the decision gate graphs, which the DSM model concepts in the decision gate graphs are not included in this paper. The OSLC services of the information in the decision gate graphs are linked to the related OSLC services representing decision gate *Objects* in the process *Graphs*.
4) Linking OSLC service providers of the *Graph* concepts to the SoTs, including the following: a) SoTs of the development scenario; b) SoTs of the configuration scenario; c) SoTs of the execution scenario; d) SoTs of the error scenario.

The SoTs are designed based on the four cosimulation scenarios in accordance with systems thinking: Development scenario; configuration scenario; execution scenario; and error scenario. Such SoTs are linked with the OSLC services of the required technical resources and development information in each OSLC service provider of *Graph*. As shown in Fig. 2, the SoTs are defined for orchestrating services among the WPMS, OSLC services, and service providers generated from the DSM

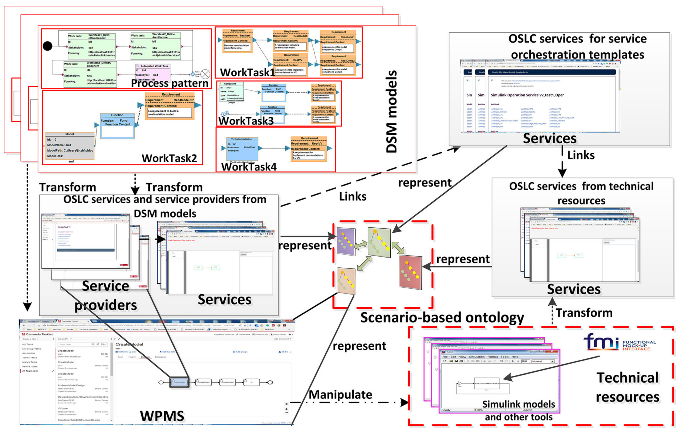


Fig. 3. Case study for evaluating cosimulation automation in the MBSE toolchain.

TABLE VI
OBJECT PROPERTIES BETWEEN O_ds, O_ts, AND O_dsm (CFM REFERS TO THE COMPONENT FROM MODE SHOWN IN SECTION VI)

$\Rightarrow$	$O\_DT_a^1$	$O\_DT_a^2$	$O\_DT_a^3$	$O\_DT_a^4$	$O\_DT_a^5$	$O\_DT_a^6$
O_sm_a	*DownloadingIn*	*DownloadingIn*	*GenerateFrom*			*linkingFrom*
$O\_smComponent_a^1$	*TemplateTo*					
$O\_smComponent_a^2$		*CheckingWith*				
O_fmu_b			*GenerateTo*			*linkingFrom*
$O\_Datafiles_e^2$				*linkingTo*	*linkingTo*	
$O\_smResult_f$				*linkingTo*	*linkingTo*	
$O\_object_b^{Gate}$				*linkingFrom*		
$O\_object_b^{FMU}$		*TemplateFrom*				*linkingTo*
$O\_object_b^{CFM}$	*TemplateFrom*					*linkingTo*

models and technical resources. In order to implement the four cosimulation scenarios, except for ontology representing OSLC entities generated from technical resources, the SoTs are linked to the related O_dsm through OWL *object properties* for capturing sufficient information and implementing cosimulations.

1) Development Scenario: In order to support the development scenario, the ontologies of the related SoTs are defined as

$$O\_ds ::= \{O\_dt_a^t\} \tag{5}$$

where O_ds is a collection of SoTs for the development scenario. $O\_dt_a^t$ refers to the ath SoT supporting the development scenario, where *t* refers to the SoT types, including. 1) downloading the component template for component development; 2) downloading a model template for the FMU; 3) generating the FMU; 4) making a decision in the decision gates; 5) reviewing the results; and 6) uploading the components for cosimulation.

The $O\_dt_a^t$ is defined where *t* refers to the number of each type, as listed in Table VI.

The six service templates are linked to two types of OSLC entities: first, OSLC services of the required technical resources; and second, OSLC services of the DSM model concepts (introduced in Section VI). Therefore, in the case study, *object properties* between the entities are illustrated in Table VI.

2) Configuration Scenario: This scenario is used to create cosimulation models using technical resources, based on the DSM models. During creating models, parameter, solver, and cosimulation interface configurations are implemented before each simulation execution. Fig. 1 represents configuration and execution scenarios defined based on two different cosimulation situations. The entities including tools, models, interfaces, and FMUs are used in the cosimulation. The dashed lines indicate the mapping relationships, for example, one interface is mapped to the related interface execution files. Several Tool Operation Service concepts (*toSer*) are defined to represent a set of tool operations for implementing the cosimulation configurations and executions. The black lines with arrows indicate the *toSer* concepts in the configuration scenario. The red lines with arrows indicate the *toSer* used in the execution scenario. The details are explained as follows.

1) Fig. 1(a) depicts tool coupled cosimulations. In this case, the tools are directly implemented to support cosimulation execution. During the cosimulations, there are two types of tools: first, $O\_tool_1^M$ (*Tool1*); and second, $O\_tool_2^S$ (*Tool2*).
2) Fig. 1(b) depicts FMU-based cosimulations. In this case, one FMU (O_fmu_1) is generated from one model in *Tool2* ($O\_tool_2^S$) as middle-ware for implementing the cosimulations in *Tool1* ($O\_tool_1^M$).
3) *Model1* (O_sm_1) is a model implemented in $O\_tool_1^M$. *Model2* (O_sm_2) is a model implemented in $O\_tool_2^S$.
4) The *interface1* ($O\_smComponent_1^{FI}$) refers to one interface block communicating with *interface2* ($O\_smComponent_2^{FI}$) in *model2*.
5) The *InterfaceExe1* ($O\_Datafiles_1^{IF}$) refers to one execution file supporting communications between interfaces.
6) The *Resultfile1* ($O\_Datafiles_1^{RF}$) refers to one result file for the cosimulations.

During cosimulation configurations, *toSers* for three operational situations in the configuration scenario are defined as

1) *Configuration toSer* refers to tool operations for configuring tools, e.g., *setting the solver.*
2) *Configuration toSer1* refers to tool operations for creating/deleting the models in the tools, including
   a) *toSer1.1* refers to the tool operations for models, such as open model and create models.
   b) *toSer1.2* refers to the tool operations for adding and deleting blocks or interface blocks, e.g., *insert interface*.
   c) *toSer1.3* refers to the tool operations for managing interface execution files, e.g., generating the interface execution files.
3) *Configuration toSer2* refers to tool operations for configuring models, such as parameter and interface settings.

Based on the entities and three operational situations used in the configuration scenarios, the ontology is defined as follows:

$$O\_cs ::= \{O\_sc_a, O\_openM_b, O\_cm_c, O\_closeM_d, O\_pc_e\} \tag{6}$$

where O_cs is a collection of SoTs for creating cosimulation models in the configuration scenario; O_sc_a is the ath SoT to configure tools; O_openM_b is the bth SoT to open models; O_cm_c is the cth SoT to create models; O_closeM_d is the dth SoT to close models; and O_pc_e is the eth SoT to configure parameters.

As shown in Table VII, *apiT* in $O\_toolAPI_d^{(apiT,tT)}$ is defined as *id* illustrated in Table III. The object properties between O_cs and tool APIs are illustrated for supporting cosimulation automation in the configuration scenarios.

3) Execution Scenario: This scenario is defined for triggering the cosimulation executions and managing the simulation results where following *toSers* are defined for two operational situations, as shown in Fig. 1.

TABLE VII
OBJECT PROPERTIES BETWEEN O_CS AND THE RELATED OSLC SERVICES OF THE TOOL APIs (IMPL REFERS TO IMPLEMENT)

$\Rightarrow$	O_OpenM_a	O_CloseM_b	O_Cm_c	O_SC_d	O_PC_e
$O\_toolAPI_d^{(1.2,tT)}$		*impl*			
$O\_toolAPI_d^{(2.2,tT)}$		*impl*			
$O\_toolAPI_d^{(1.1,tT)}$	*impl*		*impl*		
$O\_toolAPI_d^{(2.1,tT)}$	*impl*				
$O\_toolAPI_d^{(2.3,tT)}$			*impl*	*impl*	*impl*
$O\_toolAPI_d^{(8,tT)}$				*impl*	
$O\_toolAPI_d^{(7,tT)}$					*impl*
$O\_toolAPI_d^{(2.5,tT)}$			*impl*		
$O\_toolAPI_d^{(4.1,tT)}$			*impl*		
$O\_toolAPI_d^{(3.1,tT)}$			*impl*		
$O\_toolAPI_d^{(12,tT)}$			*impl*		
$O\_toolAPI_d^{(5.1,tT)}$			*impl*		

1) *Execution toSer* refers to the tool operations for triggering cosimulations.
2) *Execution toSer1* refers to the tool operations for managing the simulation results.

Based on the two operational situations and required entities, ontology for the execution scenario is defined as follows:

$$O\_se ::= \{O\_sm_a, O\_sexe_a^t, O\_tool_c^M, O\_toolAPI_{:}^{(:,c)}, O\_Datafiles_{:}^{:}\} \quad (7)$$

where O_se is a collection of SoTs supporting simulation executions. $O\_sexe_a^t$ is the SoT during the ath simulation execution where *t* refers to two types of SoTs including the following.

1) *Triggering execution* (TE) is an SoT for triggering the *master* tool ($O\_tool_c^M$).
2) *Managing results* is an SoT for generating and reviewing the result files for the cosimulations (such as $O\_Datafiles_{:}^{:}$).

O_sm_a is the ontology of the OSLC services representing the ath simulation model for implementing $O\_sexe_a^t$ in the *master* tool. $O\_tool_c^M$ is the ontology of the OSLC services representing the *master* tool for implementing the simulation models. $O\_toolAPI_{:}^{(:,c)}$ refers to the OSLC services of all the APIs in the master tool c to execute the simulations. $O\_Datafiles_{:}^{:}$ refers to all the data files related to $O\_sexe_a^{TE}$.

4) Error Scenario: During cosimulation, some errors may occur and stop the simulations. When these errors occur, the related OSLC services detecting errors are updated in the WPMS. To formalize them, the ontology for the error scenario is

$$O\_er ::= \{O\_sm_a, O\_error_b^{(t,tool)}, O\_sexe_c^{TE}\} \quad (8)$$

where O_er is a collection of SoTs for error scenarios. O_sm_a is the ontology representing an OSLC service of the ath simulation model. During simulation execution *c* ($O\_sexe_c^{TE}$), the bth error occurs, where *tool* refers to the tool (including *Master* and *Slave*) occurring errors. The *t* in O_error refers to 1) *execution error*: Some errors occur during simulation executions and 2) *interface mismatch error*: Some errors occur during inserting interfaces for creating the cosimulation models.

D. Ontology for WPMS

In the MBSE tool-chain, a WPMS is generated from DSM models to implement the cosimulations for developers automatically. The WPMS is created based on BPM Camunda [34] whose process engine is developed based on business process model and notation (BPMN) [35]. Thus, the ontology representing the WPMS is defined as follows based on the BPMN models:

$$O\_wpms ::= \{O\_process_{:}\} \quad (9)$$

where O_wpms refers to the ontology of the WPMS including all the $O\_process$ concepts. $O\_process_i$ is the ontology of the ith process implemented in the WPMS where several activities are defined

$$O\_process_i ::= \{AWT_a, HWT_b, Start_c, End_d, DecisionGate_e, Sequence_f\} \quad (10)$$

where *AWT* refers to the ath work task implemented automatically in the WPMS; *HWT* refers to the bth work task implemented by humans; *Start* is the cth start node of a process; *End* refers to the dth termination of the process; *DecisionGate* refers to the eth process for making decision; and *Sequence* refers to fth sequence relationship connecting with the previous elements in the process. The *object properties* between such elements are illustrated in Table VIII.

TABLE VIII
RELATIONSHIPS BETWEEN THE SERVICES OF THE WORK TASKS IN A WPMS

$\Rightarrow$	$O\_process_i$	AWT_a	HWT_b	$Start_c$	End_d	$DecisionGate_e$
AWT_a	*include*					
HWT_b	*include*					
$Start_c$	*include*					
End_d	*include*					
$DecisionGate_e$	*include*					
$Sequence_f$		*Startfrom /toEnd*	*Startfrom /toEnd*	*Startfrom*	*/toEnd*	*Startfrom /toEnd*
$O\_graph_a^{HWT}$			*link_to*			
$O\_graph_a^{AWT}$		*link_to*				
$O\_object_b^{Gate}$						*link_to*

VI. CASE STUDY AND EVALUATION

A. Introduction

A case study is proposed for evaluating the completeness and logic of the scenario-based ontology, as shown in Fig. 3. This case study refers to a cosimulation scenario based on MATLAB/Simulink, in which the MATLAB FMI toolbox [36] is used to load the FMUs in Simulink, implemented using our proposed MBSE tool-chain. First, stakeholders build the DSM models to formalize the cosimulation process, system artifacts, and information for cosimulation operations. The DSM models and technical resources, such as Simulink models for cosimulation, are then transformed into a WPMS with OSLC entities including OSLC services and service providers. Based on the scenario-based ontology, each work task in the WPMS is accessed to the related OSLC service provider linked with the required OSLC services representing the development information and technical resources. Through the WPMS, stakeholders control and manipulate the technical resources to implement cosimulation automatically.

In Table III [39], the meta-models of the DSM models are developed, based on the GOPPRR approach in the previous work [27]. Meta-models of the process pattern are developed for constructing the process and subprocess graphs based on the BPMN. which include main entities as following. 1) *Process Graph*, which is a graph for illustrating the cosimulation processes. 2) *Start Object*, which is a process start node. 3) *Work task*, which is a work task in the *process* referring to the *Activity* in BPMN. There are two types of work tasks: *The HWT*, which is a work task implemented by developers, and the *AWT*, which is a work task implemented automatically. 4) *Decision gate*, which is a *Gateway* in the BPMN, referring one node forking and the merging of paths depending on the expressed conditions. 5) *End*, which is a process termination node.

In order to formalize the information pattern related to the cosimulations, three other graphs are proposed, as shown in Table III [39], which are as follows.

1) The *WorkTask* graph refers to a collection of *Requirement*, *Component*, *Tool*, *FMU*, *ComponentFromModel*, and *Modelstructrue* entities and their relationships. It is used to represent the related information in each work task. Such meta-models are extended from the requirement diagram of the SysML [37].
2) *ModelStructure* graph refers to a collection of cosimulation components and their connectors.
3) *V&V* refers to a process that describes the cosimulation parameter settings and executions, including the *task* that refers to a simulation, *parameter configurations* (*pc*), *simulation configurations* (*sc*), *simulation results*, *simulation review*, *start*, and *end* nodes. The *Associate* and *Sequence* relationships are used to present connections between them.

In the case study, DSM models are built to formalize one cosimulation process using such meta-models, which includes following four working tasks and one decision gate, as shown in Fig. 3 (process pattern). 1) *WorkTask1* defines the requirements of the cosimulation, as shown in Fig. 3 (WorkTask1). The requirements for the cosimulation, building models, and *V&V*, and their relationships are formalized. 2) *WorkTask2* defines one cosimulation model structure and its related requirement and function, as shown in Fig. 3 (WorkTask2). The model structure *Object* is decomposed into a model structure *Graph* including two MCs and their relationships, as shown in Fig. 4(a) (model structure). It represents two components in one Simulink model (*sm1*). 3) *WorkTask3* defines one MC and one FMU for constructing the cosimulation model. Moreover, the related requirements and functions are also defined, as shown in Fig. 3 (WorkTask3). The MC and FMU are defined and mirrored to the two components in the model structure [see Fig. 4(a) (model structure)]. 4) *WorkTask4* defines the *V&V* and related requirement, as shown in Fig. 3 (WorkTask4). The *V&V* block is decomposed into a *V&V Graph*, which represents a set of task *Objects* referring to the simulation executions in a sequence. Each task *Object* is associated with the parameter settings, simulation configurations, and results.

Then, the DSM models are transformed to ontology representing these DSM models. Then, using the WPMS compiler and OSLC adapters, the ontology and technical resources are transformed into the WPMS, OSLC services, and services providers (see Fig. 1 [39]). The technical resources include MATLAB/Simulink, MATLAB FMI toolbox, related tool APIs, Simulink components in the Simulink model, and FMUs. After the WPMS is generated, stakeholders login into their own accounts and access the OSLC services of the DSM models for reviewing development information. Moreover, they create simulation models based on the model structure *Graph* through OSLC services referring to the SoTs during *WorkTask4*. Such SoTs are linked to the OSLC services of the DSM models to capture the required information and of the technical resources to implement tool operations. During creating models, the MATLAB FMI tool box and Simulink are implemented to load the FMU and add the related components and connections in Simulink. Then, OSLC services for the parameter and simulation configurations are then accessed to implement the Simulink model configurations. Furthermore, the OSLC services for the simulation execution are used to control the simulation executions, after which the results are reviewed in the OSLC services linked to the WPMS. Finally, the OSLC services of such simulation results are loaded by the design gate in WPMS aiming to decide whether the process is terminated. When the results are satisfied by the decision conditions, the process is ended. Otherwise, the process goes back to *Start*.

B. Evaluation Methods

In the case study, we use Protégé to develop the ontology based on the OWL. In order to verify the completeness and logic of the ontology, SPARQL is adopted to evaluate whether the ontology can describe the OSLC services generated by the DSM models completely. Moreover, SQWRL is adopted to evaluate whether the ontology can represent the logic for supporting cosimulation automation.

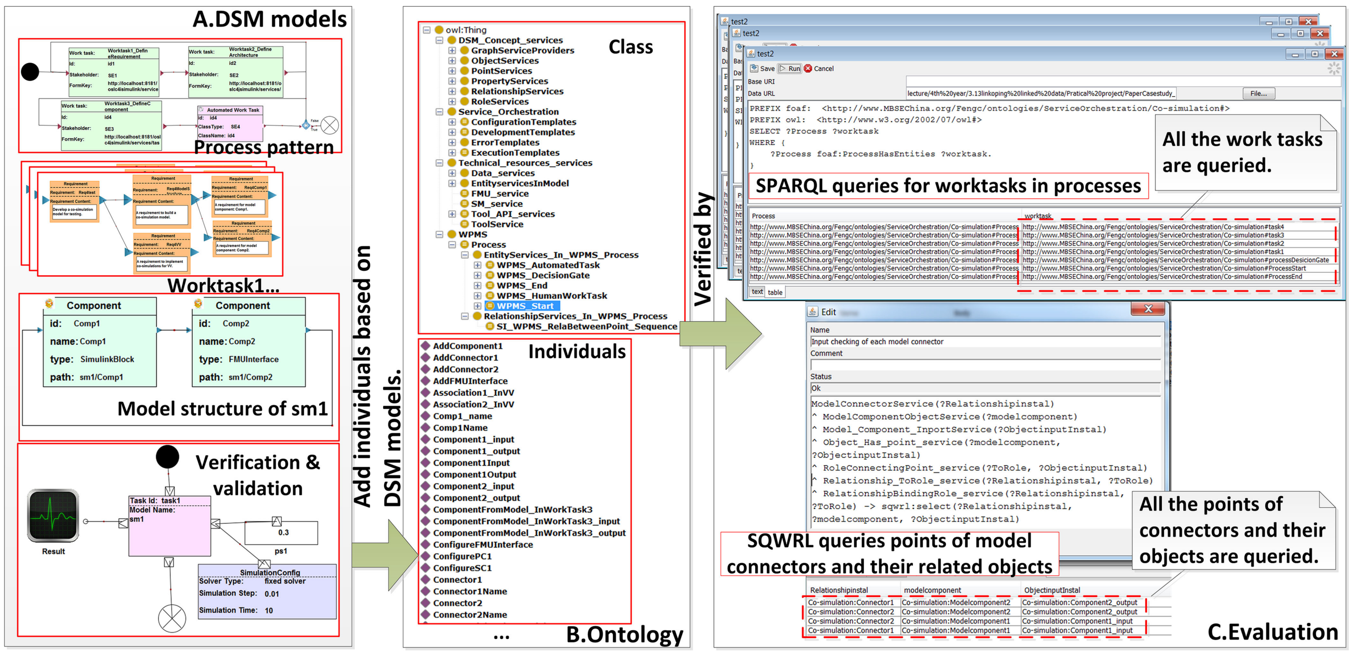
Individuals of the OWL models are then defined in Protégé, including the following: 1) OSLC service providers and services generated from the DSM models, as shown in Fig. 3; 2) OSLC services of SoTs; 3) OSLC services of the required technical resources; and 4) the generated WPMS; then, Twinkle [2] and Protégé are used for implementing the SPARQL and SQWRL queries.

1) Evaluation Based on SPARQL Queries: SPARQL is adopted for evaluating the completeness of the ontology to describe the OSLC services of the DSM models. In the case study, the completeness evaluation includes the following: 1) querying the complete information (OSLC services of the *Object/Relationships/Property*) in the OSLC service providers of the model structure, process, work task, and verification and validation *Graphs* in the DSM models; 2) querying the complete information in the WPMS (*Worktask* as an example); 3) querying the OSLC services representing the *Points* and *Roles* in the OSLC services of the MCs and connectors in the model structure *Graph*; 4) querying the OSLC services of the properties in the OSLC services of the *Objects* and *Relationships*.

Algorithm 1 is an SPARQL algorithm proposed for completeness evaluation.

2) Evaluation Based on SQWRL Queries: SQWRL queries are adopted to verify whether the ontology captures the logic for implementing the cosimulation automatically. The logic includes the following two parts. 1) Logic describing the OSLC services related to the DSM models: first, *model structure* refers to the logic of the OSLC services describing MCs connected to each other; second, *process sequence* refers to the logic describing the sequence orders of the DSM model concepts, such as the sequence order in the process. 2) Logic supporting tool operations: *creating model* is adopted as an example for evaluation.

Algorithm 2 is an SQWRL algorithm adopted for verifying the logic of the DSM models. $O\_object_{:}^{modelcomponent}$, $O\_relationship_{:}^{Relationshipinstal}$, and $O\_point_{:}^{ObjectinputInstal}$ are defined first that refer to the individuals representing the OSLC services of the *Object*, *Relationship*, and *Point* concepts in DSM models. Furthermore, the individuals are queried using


A.DSM models
Process pattern
Worktask1...
Component
Model structure of sm1
Verification & validation
Result
Add individuals based on DSM models.
Class
owl:Thing
DSM_Concept_services
Service_Orchestration
Technical_resources_services
WPMS
Individuals
B.Ontology
Verified by
All the work tasks are queried.
SPARQL queries for worktasks in processes
All the points of model connectors and their related objects are queried.
SQWRL queries points of model connectors and their related objects
C.Evaluation

```
PREFIX foaf:  <http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT ?Process ?worktask
WHERE {
    ?Process foaf:ProcessHasEntities ?worktask.
}
```

Process	worktask
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#task4
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#task3
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#task2
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#task1
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#processDesicionGate
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#ProcessStart
http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#Process	http://www.MBSEChina.org/Fengc/ontologies/ServiceOrchestration/Co-simulation#ProcessEnd

Name: Input checking of each model connector
Comment:
Status: Ok

```
ModelConnectorService(?Relationshipinstal)
^ ModelComponentObjectService(?modelcomponent)
^ Model_Component_InportService(?ObjectinputInstal)
^ Object_Has_point_service(?modelcomponent,
?ObjectinputInstal)
^ RoleConnectingPoint_service(?ToRole, ?ObjectinputInstal)
^ Relationship_ToRole_service(?Relationshipinstal, ?ToRole)
^ RelationshipBindingRole_service(?Relationshipinstal,
?ToRole) -> sqwrl:select(?Relationshipinstal,
?modelcomponent, ?ObjectinputInstal)
```

Relationshipinstal	modelcomponent	ObjectinputInstal
Co-simulation:Connector1	Co-simulation:Modelcomponent2	Co-simulation:Component2_output
Co-simulation:Connector2	Co-simulation:Modelcomponent2	Co-simulation:Component2_output
Co-simulation:Connector2	Co-simulation:Modelcomponent1	Co-simulation:Component1_input
Co-simulation:Connector1	Co-simulation:Modelcomponent1	Co-simulation:Component1_input

Fig. 4. (a) DSM models to formalize the cosimulation development and configurations. (b) OWL models of the DSM models. (c) Evaluation using SPARQL and SQWRL.

the several object properties listed in the following, representing the OSLC services related to them.

1) *Object_Has_point_service* refers to one OSLC service representing a relationship, where one *Object* has several *Points*.
2) *Relationship_ToRole_service* refers to one OSLC service representing a relationship, where one *Relationship* has one *Role* as its end.
3) *RoleConnectingPoint_service* refers to one OSLC service representing a relationship, where one *Role* is connected to one *Point*.

Moreover, SQWRL is adopted to evaluate the logic supporting the tool operations, as shown in Algorithm 3. First, the OSLC services of opening (closing) tools are queried. Then, the OSLC services of the tool operations related to the models are queried, such as opening model. Furthermore, the OSLC services supporting tool operations, which are related to the MCs, FI, and model connectors, are queried to evaluate whether the related OSLC services are queried with the required information. For example, OSLC services of *adding model component* needs information about the related components and models.

C. Summary and Limitations

From the results, we found that all the metrics mentioned in Table I were satisfied. Through the SPARQL query, the contents related to the DSM model concepts were found to be described completely as shown in Fig. 4(c), which are 1) the OSLC services of the *Objects*/*Relationships* in the service providers of each work task; 2) the Worktask in the WPMS; 3) the OSLC services of the *Points* in the OSLC services of *Objects*/*Relationships*; 4) the OSLC services of the *Properties* of the related *Graphs*, *Objects*, *Relationships*, *Roles*, and *Points*.

The results from the SWRQL queries were utilized for evaluating the logic of the DSM models and tool operations supporting simulation automation. The logic of the DSM models refers to the descriptions on how the *Points* in the *Objects* connect with each other through the *Relationships*. Two main metrics were verified through Algorithm 2, which are as follows.

1) The *Points* connected by model connectors in the *Model structure Graphs* were queried. Based on the queried inputs and outputs, the logic of the linking of the model connectors with the *Objects* is represented.
2) The logic representing the linking of the *Process Sequence Relationships* with the *Objects* in the *Process Graphs* was queried. Through the results, the *Objects* were queried in association with each *sequence*, illustrating the sequence orders.

In Algorithm 3, the tool operations for creating models were queried as one example in shown in Fig. 4(c). From the results, we determined that all the information required for each tool operation is depicted. From the previous evaluation, we can infer that the scenario-based ontology is designed to satisfy the completeness and logic for supporting cosimulation automation for the case study in our MBSE tool-chain. However, there are several limitations in this paper. 1) The proposed OWL-based approach is used to provide a framework aiming to develop ontology for service-orchestrations in the MBSE tool-chain. The approach is expected to design ontology for general cosimulation setups for different scenarios. Moreover, formal verifications for the functionalities of ontology will be developed using contract theory in the future [38]. 2) The designed scenario-based ontology is limited to an MBSE tool-chain developed in the previous work [39]. Other scenarios for cosimulations are not included and will be extended in the future. For example, model exchange methods except for the FMI are not included. 3) The current decision-making template is defined as an OSLC-service linking to the OSLC service of decision gate *Objects*. In future, the decision-making template will be linked with the OSLC service provider of the decision gate *Graph*.

VII. Conclusion

In this paper, a scenario-based ontology was presented for service orchestration supporting automated cosimulation in an MBSE tool-chain. The ontology is basic of service orchestrations for linkings between a WPMS, OSLC entities representing the DSM models, technical resources, and SoTs. Through a case study, SPARQL and SQWRL were used to evaluate the completeness and logic of the ontology. The results demonstrated that stakeholders manipulated the tools to implement the cosimulations automatically, through the WPMS in the MBSE tool-chain. Moreover, the ontology serves as a potential specification for realizing simulation automation in general cosimulation platforms. In future, we intend to develop ontologies for other complex scenarios in order to support more complicated design automation.

References

[1] S. Durbha, R. King, and N. Younan, "An information semantics approach for knowledge management and interoperability for the global earth observation system of systems," *IEEE Syst. J.*, vol. 2, no. 3, pp. 358–365, Sep. 2008.

[2] N. Kumar and S. Kumar, "Querying RDF and OWL data source using SPARQL," in *Proc. 4th Int. Conf. Comput., Commun., Netw. Technol.*, Jul. 2013, pp. 1–6.

[3] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member Submission*, vol. 21, pp. 1–157, 2004.

[4] L. Jinzhi, C. De-Jiu, G. Didem, and T. Martin, "An investigation of functionalities of future tool-chain for aerospace industry," in *Proc. INCOSE Int. Symp.*, 2017, pp. 1408–1422.

[5] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: State of the art," Univ. Antwerp, Antwerp, Belgium, Tech. Rep. 1702.00686, Feb. 2017.

[6] Modelica Association Project "FMI," "Functional mock-up interface for model exchange and co-simulation," Modelica Association, Tech. Rep., 2013.

[7] J. Lu, M. Törngren, D.-J. Chen, and J. Wang, "A tool integration language to formalize co-simulation tool-chains for cyber-physical system (CPS)," in *Software Engineering and Formal Methods*. Berlin, Germany: Springer, 2018, pp. 391–405.

[8] J. Lu, J. Ding, F. Zhou, and X. Gong, "Research of tool-coupling based electro-hydraulic system development method," in *Proc. 6th Int. Asia Conf. Ind. Eng. Manage. Innov.*, 2016, pp. 213–224.

[9] J. Fitzgerald, P. G. Larsen, and M. Verhoef, *Collaborative Design for Embedded Systems*, J. Fitzgerald, P. G. Larsen, and M. Verhoef, Eds. Berlin, Germany: Springer, 2014.

[10] B. Chandrasekaran, J. Josephson, and V. Benjamins, "What are ontologies, and why do we need them?" *IEEE Intell. Syst.*, vol. 14, no. 1, pp. 20–26, Jan. 1999.

[11] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins, and M. Ingham, "An ontology for state analysis: Formalizing the mapping to SysML," in *Proc. IEEE Aerosp. Conf.*, Mar. 2012, pp. 1–16.

[12] C. Hennig, A. Viehl, B. Kämpgen, and H. Eisenmann, *Ontology-Based Design of Space Systems* (Lecture Notes in Computer Science), vol. 9982, P. Groth *et al.*, Eds. Cham, Switzerland: Springer, 2016.

[13] J. Miller and G. Baramidze, "Ontologies for modeling and simulation: An initial framework," 2004. [Online]. Available: http://cobweb.cs.uga.edu/~jam/jsim/DeMO/paper/journal/final/demojournal-5.pdf

[14] P. T. Grogan and O. L. de Weck, "The ISoS modeling framework for infrastructure systems simulation," *IEEE Syst. J.*, vol. 9, no. 4, pp. 1139–1150, Dec. 2015.

[15] J.-B. Soyez, G. Morvan, R. Merzouki, and D. Dupont, "Multilevel agent-based modeling of system of systems," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2084–2095, Dec. 2017.

[16] P. Benjamin, M. Patki, and R. Mayer, "Using ontologies for simulation modeling," in *Proc. Winter Simul. Conf.*, Dec. 2006, pp. 1151–1159.

[17] G. Sirin, C. J. J. Paredis, B. Yannou, E. Coatanea, and E. Landel, "A model identity card to support simulation model development process in a collaborative multidisciplinary design environment," *IEEE Syst. J.*, vol. 9, no. 4, pp. 1151–1162, Dec. 2015.

[18] Z. Ming, G. Wang, Y. Yan, J. D. Santo, J. K. Allen, and F. Mistree, "An ontology for reusable and executable decision templates," *J. Comput. Inf. Sci. Eng.*, vol. 17, no. 3, 2017, Art. no. 031008.

[19] G. A. Silver, J. A. Miller, M. Hybinette, G. Baramidze, and W. S. York, "DeMO: An ontology for discrete-event modeling and simulation," *SIMULATION*, vol. 87, no. 9, pp. 747–773, Sep. 2011.

[20] C. Gomes *et al.*, "Semantic adaptation for FMI co-simulation with hierarchical simulators," *SIMULATION*, vol. 95, pp. 241–269, Apr. 2018.

[21] F. Zeyda, J. Ouy, S. Foster, and A. Cavalcanti, *Formalising Cosimulation Models* (Lecture Notes in Computer Science), vol. 10729, A. Cerone and M. Roveri, Eds. Cham, Switzerland: Springer, 2018.

[22] J. Lu, D. Gürdür, D.-J. Chen, J. Wang, and M. Törngren, "Empirical-evolution of frameworks supporting co-simulation tool-chain development," in *Trends and Advances in Information Systems and Technologies* (Advances in Intelligent Systems and Computing), vol. 745. Berlin, Germany: Springer, 2018, pp. 813–828.

[23] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "An integrated workbench for model-based engineering of service compositions," *IEEE Trans. Serv. Comput.*, vol. 3, no. 2, pp. 131–144, Apr.–Jun. 2010.

[24] T. Labidi, A. Mtibaa, W. Gaaloul, and F. Gargouri, "Ontology-based SLA negotiation and re-negotiation for cloud computing," in *Proc. IEEE 26th Int. Conf. Enabling Technol.: Infrastructure Collaborative Enterprises*, Jun. 2017, pp. 36–41.

[25] M. K. S. Biehl, "A modeling language for the description and development of tool chains for embedded systems," Ph.D. dissertation, Dept. Mach. Des., KTH Roy. Inst. Technol., Stockholm, Sweden, 2013.

[26] Y. Wu, F. He, D. Zhang, and X. Li, "Service-oriented feature-based data exchange for cloud-based design and manufacturing," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 341–353, Mar./Apr. 2016.

[27] J. Lu, D. Chen, J. Wang, and M. Torngren, "Towards a service-oriented framework for MBSE tool-chain development," in *Proc. 13th Annu. Conf. Syst. Syst. Eng.*, 2018, pp. 568–575.

[28] S. Kelly, K. Lyytinen, and M. Rossi, "MetaEdit+ a fully configurable multi-user and multi-tool case and came environment," in *Advanced Information Systems Engineering*. Berlin, Germany: Springer, 1996, pp. 1–21.

[29] H. Kern, A. Hummel, and S. Kühne, "Towards a comparative analysis of meta-metamodels," in *Proc. Compilation Co-Located Workshops DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, VMIL'11*, 2011, vol. 1, pp. 7–12.

[30] R. R. Smullyan, *First-Order Logic*, vol. 43. Berlin, Germany: Springer Science & Business Media, 2012.

[31] H. W. Lawson, *A Journey Through the Systems Landscape*. London, U.K.: College Publications, 2010.

[32] M. Simulink and M. A. Natick, *The Mathworks*. Natick, MA, USA: MathWorks, 1993.

[33] W. Hongwei, L. Jinzhi, W. Guoxin, and M. Changfeng, "Ontology supporting model-based systems engineering based on a GOPPRR approach," in *New Knowledge in Information Systems and Technologies*. WorldCIST'19 (Advances in Intelligent Systems and Computing 930). Cham, Switzerland: Springer, 2019.

[34] A. Fernandez, "Camunda BPM platform loan assessment process lab," Queensland Univ. Technol., Brisbane, QLD, Australia, Tech. Rep. 8725853, 2013.

[35] M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz, "BPMN conformance in open source engines," in *Proc. IEEE Symp. Serv.-Oriented Syst. Eng.* Mar. 2015, pp. 21–30.

[36] A. Modelon, "FMI toolbox for Matlab," 2014. [Online]. Available: http://www.modelon.com/products/fmi-toolbox-for-matlab

[37] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. San Mateo, CA, USA: Morgan Kaufmann, 2014.

[38] J. Westman and M. Nyberg, "Extending contract theory with safety integrity levels," in *Proc. IEEE 16th Int. Symp. High Assur. Syst. Eng.*, Jan. 2015, pp. 85–92.

[39] J. Lu, J. Wang, D. Chen, J. Wang, and M. Torngren, "A service-oriented tool-chain for model-based systems engineering of aero-engines," *IEEE Access*, vol. 6, pp. 50443–50458, 2018.