A Service-oriented Approach Supporting Model Integration in Model-based Systems Engineering

Conference Paper · April 2023

DOI: 10.1109/SysCon53073.2023.10131078

5 authors, including:

Guoxin Wang
Beijing Institute of Technology
126 PUBLICATIONS **1,516** CITATIONS
SEE PROFILE

Shouxuan Wu
Beijing Institute of Technology
5 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Lu Jinzhi
Beihang University
127 PUBLICATIONS **1,326** CITATIONS
SEE PROFILE

Yan Yan
Shanghai Dianji University
156 PUBLICATIONS **1,759** CITATIONS
SEE PROFILE

A Service-oriented Approach Supporting Model Integration in Model-based Systems Engineering

Rui Chen
School of Mechnical Engineering
Beijing Institute of Technology
Beijing, China

Guoxin Wang
School of Mechnical Engineering
Beijing Institute of Technology
Beijing, China

Shouxuan Wu
School of Mechnical Engineering
Beijing Institute of Technology
Beijing, China

Jinzhi Lu*[1,2]
[1]*School of Aeronautic Science and Engineering,*
Beihang University, Beijing,
China
[2]*EPFL SCI-STI-DK, Station 9,*
CH1015 Lausanne,
Switzerland
jinzhi.lu@epfl.ch

Yan Yan
School of Mechnical Engineering
Beijing Institute of Technology
Beijing, China

2023 IEEE International Systems Conference (SysCon) | 978-1-6654-3994-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/SysCon53073.2023.10131078

***Abstract*—When using Model-Based Systems Engineering (MBSE) to develop complex systems, models using different syntax and semantics are typically implemented in a heterogeneous environment which leads to difficulties to realize data integrations across the entire lifecycle. Specifically, seamless exchanges between models of different modeling tools are needed to support system lifecycle activities such as requirement analysis, function analysis, verification and validation. This article illustrates a service-oriented approach to support model integration for model-based systems engineering, especially between architecture design and system verification. First, a set of semantic mapping rules between architecture models and simulation models based on Open Service of Lifecycle Collaboration (OSLC) are proposed to support the formalization of technical resources (models, data, APIs). Then OSLC adapters are developed to transform models, data and APIs into web-based services. The services are deployed by a service discovering plug-in within a specific modeling tool for model information exchange. The approach is illustrated by a case study on KARMA architecture model and Modelica simulation model for a six-degree-of-freedom robot (RobotR3) system. We evaluate the availability and efficiency of this method from both qualitative and quantitative perspectives. The results show that our approach is effective in model and data integration.**

***Index Terms*—Model integration, Model-Based Systems Engineering, Open Service for Lifecycle Collaboration, Modelica**

I. Introduction

Model-based Systems Engineering (MBSE) is an emerging technology that supports model-based development of complex systems. MBSE makes use of formal models to support system requirements, design, analysis, verification and validation activities involved in the entire system life cycle [1]. To support MBSE implementation, IEEE 1220 [2] defines the Systems Engineering Process, which decomposes into eight subprocesses, including requirements analysis, requirements validation, functional analysis, functional verification, synthesis, design verification, etc. Modeling languages (such as SysML) are proposed to construct system models from different perspectives to support requirements definition, analysis of system architecture and design. After system architectural models are constructed, specific analysis and modeling tools will be used for system verification. As a common approach for system verification, simulation can be applied in several stages of system design, such as scheme analysis in conceptual design stage and parameter optimization in detailed design stage [3]. Simulation models can be built by physical modeling languages (such as Modelica [4], etc.) to verify whether the design architecture satisfies the validated requirements baseline, thus speeding up the design iteration and improving the efficiency of system development.

However, due to the heterogeneity between system architecture models and simulation models, there are some problems to realize data integrations of the whole system architecture modeling and simulation process. Firstly, system architecture models and simulation models are developed with different modeling languages which have heterogeneous syntax and semantics. The meanings of model's elements and relationships defined in different rules are diverse from each other. This leads to challenges in exchanging data and knowledge among different stakeholders. Second, specific tools used to construct models have different data protocols and development mechanisms. These tools come from different vendors, use different development frameworks, have different data standards and interfaces, and are not compatible with each other. Developing point-to-point interfaces for tool interoperability is limited by the ability of tools to provide external APIs, thus having poor reusability and scalability. Thirdly, there is a lack of implementation to enable model semantics integration and tool interoperability mentioned above, making it difficult to achieve seamless end-to-end integration of models.

This paper proposes a service-oriented approach supporting model integration in MBSE, especially for system architecture and simulation. The service-oriented approach refers to the use of Open Service for Lifecycle Collaboration (OSLC) [5] ser-

vices, which is an integration specification that facilitates the expression of development information and technical resources (models, data, and tools) into unified formats accessed through URLs [6]. In our approach, we use the OSLC core model to transform KARMA [7] architecture model, Modelica [4] simulation model and APIs for operating the model into unified descriptions. Mapping rules between them are established according to OSLC Core specification to eliminate semantic heterogeneity. Secondly, OSLC adapters are developed to transform these heterogeneous models and APIs into online OSLC services through the mapping rules. Model topology as well as properties are visualized within the adapters. Such unified services with model visualization support information exchange between tools and better comprehension among different stakeholders. Thirdly, a plug-in for querying OSLC services of Modelica models within a specific architecture modeling tool MetaGraph [1] is developed. By linking OSLC services, information of Modelica models in OpenModelica [2] can be accessed without switching the architecture modeling tool MetaGraph, which enables seamless end-to-end integration of models.

The rest of the article is organized as follows. We discuss the related work in Section 2. In Section 3, we propose a service-oriented approach supporting model integration for model-based systems engineering, especially for KARMA architecture model and Modelica simulation model. A case study of a six-degree-of-freedom robot system named RobotR3 is adopted to verify the approach in Section 4. Finally, we offer the conclusions in Section 5.

II. Related Work

In this section, we discuss several methods to support model integration in existing research, including model transformation, middleware and domain standards.

Model transformation can be considered a basic method for addressing challenges of seamless integration of model integration in MBSE. Guo et al. [8] designed code-generation language syntax to realize the integration between MBSE and Simulink models. Johnson et al. [9] created meta models for Modelica based on the extension mechanism of SysML to support the verification of system model. Kapos et al. [10] proposed a declarative approach, based on the query/view/transformation-relations (QVT-R) standard, for the transformation of SysML models to executable simulation models. However, these model transformation approaches are considered highly customized and hard to write a transformation [11]. Insufficient abstraction from meta-models and platforms or missing repositories of reusable artefacts leads to poor reusability and maintainability [12]. Lack of tool support is still one of the major problems, according to studies by Staron and Mohaghegi [13].

[1]MetaGraph is a multi-architecture modeling tool developed by Z.K.F.C: http://www.zkhoneycomb.com/.

[2]OpenModelica is an open-source Modelica-based modeling and simulation environment developed by OSMC: https://openmodelica.org/

Middleware is widely used to facilitate information exchanges between different MBSE tools. For example, extensible XML Metadata Interchange (XMI) is used to support data exchanges between SysML models across multiple tools [14]. Moreover, ontological methods are considered novel middleware. MBSE ontologies have been developed to formalize the domain-specific concepts and their relationship between different languages and solve the problem of semantic heterogeneity. Chen et al. [15] developed transformation rules between the design model and ontology model, and formalized behaviour requirements using the reasoning for verification in the system design stage. Lu et al. [16] proposed an ontology based upon graphs, objects, points, properties, roles, and relationships with extensions (GOPPRR-E) providing metamodels that support the various MBSE formalisms. Moreover, knowledge graph models are developed to support unified model representations and further implement ontological data integration based on GOPPRR-E throughout the entire lifecycle. However, most of the ontological approaches focused on domain-specific problems instead of data interoperability across the entire MBSE lifecycle.

There are many standards in specific domains for exchanges of models and data, such as STEP, QIF, AP233, Express and so on. The aim of them is to provide a neutral mechanism and implementation which are independent of any specific system and can completely describe the product data information. Kwon et al [17] proposed an approach to fuse as-designed data represented in STEP and as-inspected data represented in QIF in a standards-based digital thread based on ontology with knowledge graphs. Zadeh et al [18] explored the synergy between STEP and OSLC to integrate product manufacturing data and heterogeneous IT tools in Production Engineering. However, a single standard can't be universally used across domains, e.g. STEP is typically used in the mechanical field to exchange data between CAX systems, but the exchanged data is difficult to be applied in the IT field such as for software architecture.

Through the above analysis, in this paper, we use OSLC core specification to establish mapping rules between OSLC concepts and technical resources (models, data, APIs) to resolve the problem of model heterogeneity from the semantic level. Converting technical resources into online OSLC services enables dynamical data exchanges and developing a plug-in within the modeling tool for linking model and OSLC services enables tool interoperability across domains.

III. A Service-oriented Approach Supporting Model Integration in Model-based Systems Engineering

To realize the model integration in MBSE, typically the integration of KARMA architecture model and Modelica simulation model in this paper, we have proposed a service-oriented approach as shown in Figure 1. First, to solve the problem of heterogeneous syntax and semantics of models, we use the OSLC core model to transform KARMA and Modelica models as well as model APIs into a unified description.

Based on OSLC core specification, mapping rules between OSLC core model, KARMA model, Modelica model, and APIs are established. Then based on the mapping rules established above, OSLC adapters are developed to parse KARMA and Modelica models and generate online OSLC services. Moreover, model topology and properties will be visualized within the adapter. Thirdly, a plug-in within MetaGraph for querying OSLC services of Modelica models is developed. The main role of this plugin is to extend the functionalities of MetaGraph to establish links between KARMA models and external OSLC services generated from Modelica models, and to achieve end-to-end model integration.

A. OSLC Semantic Mapping for Model Integration

As shown in Figure 1. A, in order to transform KARMA model, Modelica model and APIs for operating model into unified semantics, the model elements and structure are first analyzed. Then the mapping rules between OSLC core model, KARMA model, Modelica model and APIs are established according to OSLC core specification.

The OSLC core model includes the *ServiceProviderCatalog*, *ServiceProvider*, *Service*, and *Resource*, as shown in Figure 1. A. ①.

a). *ServicProviderCatalog*: as a container to discover *ServiceProvider*.
b). *ServiceProvider*: used to provide one or more *Services*.
c). *Service*: a set of capabilities to manage *Resources*.
d). *Resource*: a *Resource* is the smallest unit managed by the *Service*, which has properties and can be linked to other Resources.

As shown in Figure 1. A. ②, the multi-architecture modeling language KARMA based on GOPPRR-E meta-modeling method [7] in MetaGraph is used to create system architecture models. The GOPPRR-E meta-meta-models include the *Graph*, *Object*, *Point*, *Property*, *Relationship*, *Role*, *Extension*.

a). *Graph*: a collection of *Objects* and their *Relationships* and is represented as a single window (an integrated concept of a class diagram and package in the unified modeling language (UML)). The *Graph* is a visual diagram on the top level or lower level decomposed or explored by one *Object*.
b). *Object*: one entity in the *Graph* (for example, one class concept in UML).
c). *Point*: one port in the *Object*.
d). *Relationship*: one connection between the different *Points* of the *Object*.
e). *Role*: used to define the connection rules pertaining to the relevant *Relationship*. For example, one *Relationship* may have two *Roles*. Each one connects with one *Point* in the *Object*. The *Relationship* is then connected with these *Points* in the *Object*.
f). *Property*: one attribute of the five other meta–meta models.
g). *Extension*: the additional constraints used to construct meta-models, such as a connector for binding one *Point* on the *Object* and one *Role* on one side of the *Relationship*.

As shown in Figure 1. A. ③, the multi-domain modeling language Modelica is used to create simulation models. The elements of Modelica model include the *Package* model, System model (abbreviated as *Model*), *Subsystem* model, *Component/Block*, *Connector*, *Connection*, *Parameter*, etc.

a). *Package*: a *Package* can be stored as a directory by creating a directory with the same name as the *Package*.
b). *Model*: a collection of the *Subsystem*, *Component*, *Connector* and *Connection*.
c). *Subsystem*: a *Subsystem* is composed of the *Component* or other *Subsystem*, includes the *Connector*.
d). *Component*: a *Component* is used to encapsulate equations into a reusable form, including the *Connector*.
e). *Connector*:an interface to the *Component/Subsystem*, includes the *Parameter*.
f). *Connection*: a connection between *Connectors* on the *Subsystem/Component*.
g). *Parameter*: one attribute of the other compositions.

Taking the Modelica model as an example, the mapping rules between elements of the Modelica model and concepts in the OSLC core model are established as follows.

1. A *Package* is considered as a *ServicProviderCatalog*. It contains models as a directory.
2. A *Model* in a *Package* is mapped to the *ServiceProvider*. It can provide *Services* for querying subsystem *Resources* and component *Resources*.
3. The APIs for creating, retrieving, updating, and deleting the compositions of Modelica models are considered as *Services*. Such as the APIs for viewing the topology of a model or modifying a parameter of a component are mapped to *Services*.
4. The model, subsystem and component are considered as *Resources* from different granularities, where the *Resource* of the model can link to the *Resource* of subsystem or component.

B. OSLC Adapter Development

After establishing the semantic mapping rules described above, OSLC adapters are developed to transform the heterogeneous models and APIs into online OSLC services. Taking the development of Modelica adapter as an example, as shown in Figure 1. B. First, model structures are analyzed and the key information is extracted using the data parser. Then the extracted information is used to generate Resource Description Framework (RDF) and Uniform Resource Identifiers (URIs) of OSLC services. Moreover, model topology and properties are visualized via a web UI.

1) Data Parsing: As shown in Figure 1. B. ①, in this paper, a data parser is constructed based on another tool for language recognition (ANTLR) using a predefined g4 file, which defines lexical and syntactic rules according to Modelica grammar. The textual Modelica model is analyzed lexically and syntactically by the parser and then an abstract

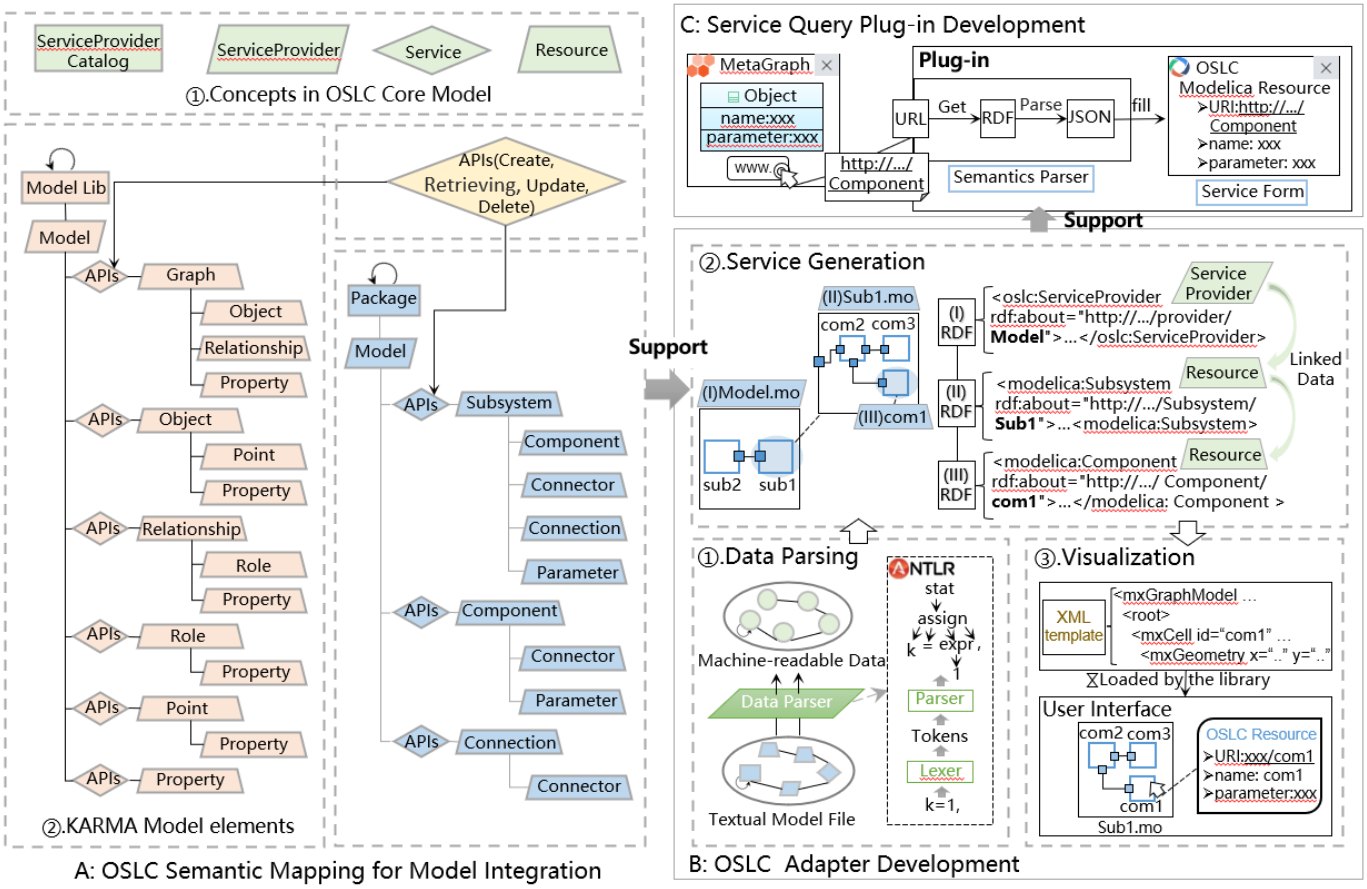

Fig. 1. The Service-oriented Approach for KARMA and Modelica Model Integration.

syntax tree is generated. By traversing the nodes of the abstract syntax tree, the model information is extracted. For example, the textual content "k=1," in a .mo file is converted to a parse tree through built-in lexer and parser. The data parsing method based on ANTLR can transform text model file into machine-readable data.

2) OSLC Services Generation: After the model information is extracted by the data parser, according to the mapping rules mentioned in Section 3.1, the extracted information will be filled into a corresponding OSLC Resource Description Framework (RDF) semantic framework, and generate URIs. RDF is a W3C standard which uses a "subject-predicate-object" triple to describe semantic information and uses URI to name each of the three parts of the triple. Because all OSLC specifications are built upon RDF, OSLC services are expected to provide and accept RDF representations. OSLC enables integration at the data level via links between model elements defined as RDF resources. Each of the resources is identified by an HTTP request URI. As shown in Figure 1.B.②, a simulation model Model.mo is mapped as a *ServiceProvider*, and its RDF semantic framework with an HTTP URI is generated. Based on RDF and linked data principle, the *ServiceProvider* Model.mo links to the *Resource* of the subsystem model Sub1.mo. Furthermore, the *Resource* of Sub1.mo links to the related *Resources* of other model elements, such as the component com1. Relationships between model elements can be discovered through OSLC services.

3) Visualization Generation: During model integrations at the data level, different stakeholders have difficulties understanding the structure and semantics of the models. Therefore, after parsing the model data and generating OSLC services, we visualised model topology and attributes on the web end, thus enhancing the uniform understanding of different stakeholders and improving integration efficiency. Web user interfaces for visualizing model topology and properties are developed in the adapter, as shown in Figure 1. B. ③. After generating OSLC services, the RDF semantics information generated from Modelica model elements will be extracted and filled into a specific XML template. The XML template is used to record information such as names, classes, parameters and geometric positions of system models, subsystems, components and connectors, etc. By loading the XML template through a specific JS-based library, the model topology and properties can be visualized on the user interface.

4) Service Query Plug-in Development: In the architecture modeling tool MetaGraph, a plug-in is developed to link KARMA model elements and OSLC services generated from Modelica model elements. The plug-in includes a semantic

parser and a service form. As shown in Figure 1. C, a KARMA model element (e.g., an object) is associated with a Modelica model element (e.g., a component) through a URL generated by the OSLC Modelica adapter. Based on the input URL, the semantic parser gets the RDF information by sending an HTTP Get request to the adapter and parsing it into JSON data. The service form loads the parsed data and visualizes it. Then the Modelica model information can be accessed within the architecture tool through this plug-in thus realizing seamless end-to-end integration of the models.

IV. Case Study

To verify the proposed approach in Section 3, a case study of the RobotR3 system is developed.

A. Scenario Definition

We take the model of RobotR3 system, which is a six-degree robot system from Mechanics.MultiBody sublibrary of Modelica Standard Library as an example. A user defines a path by start and end angle of every axis and expects that all axes are moving as fast as possible under the given restrictions of maximum joint speeds and maximum joint accelerations.

After requirements analysis, system architecture models of RobotR3 are developed by system engineers in MetaGraph, as shown in Figure 2.①. These architecture models formalize the system's physical characteristics. Based on the architecture models, Modelica models are developed by simulation engineers for system-level verification in OpenModelica, as shown in Figure 2.②. When simulation results don't meet requirements, parameters in simulation models need to be manually adjusted. After the adjustment, system engineers need to be notified and modify corresponding properties in architecture models to complete iteration of the system design.

B. Model Integration for System Architecture and Simulation

As shown in Figure 2.②. (a), the absolute angular acceleration of flange of one of the axes was found to be greater than the maximum reference acceleration of all joints during simulation. According to the analysis, it is due to the unreasonable setting of the proportional coefficient k of the component PI-controller which is used to control the motor speed in the controller subsystem. Therefore, as shown in Figure 2.②. (b), simulation engineers change the value of ks in the Modelica model from 1 to 0.8, assign it to k and rerun the simulation.

The concrete mapping rules and service semantics are shown in Table 1. Based on the mapping rules mentioned in Table 1, an OSLC adapter is developed to generate OSLC services for Modelica models. As shown in Figure 2.③. (a), the package of RobotR3 is considered as a *ServiceProviderCatalog* concept. As shown in Figure 2.③. (b), the Modelica model in the RobotR3 package is considered as a *ServiceProvider* concept, such as AxisType2.mo, which can provide *Services* of query capability for model topology, subsystems, and components in it. The model topology, subsystem and component are separately considered as a *Resource* concept. By clicking the 0th URL in the OSLC service list shown in Figure 2.③. (c), the details of the model AxisType2.mo are shown in Figure 2.③. (d), including the topology, contained subsystems, components and parameter values. The left sidebar shows the package structure of RobotR3 and the right shows the property list of the object (subsystem or component) selected by the mouse. Based on the linked data, the subsystem Controller.mo in the AxisType2.mo can be discovered as shown in Figure 2.③. (d). The details of the subsystem controller.mo are shown in Figure 2.③. (e).

Based on the identified URL and RDF information generated by the adapter, a plug-in within the architecture modeling tool MetaGraph is developed to query OSLC services of Modelica models. As shown in Figure 2.④, system engineers invoke the plug-in on the object named "Parameter" in the architecture model Controller.karma by entering the URI of the Modelica model Controller.mo. The plug-in parses the RDF semantic information located by the entered URL and visualizes it in the service form. After that, system engineers check the form and capture the modified value of proportional coefficient k of the component PI-controller, and return to modify the value from 1 to 0.8. The information exchange between the KARMA architecture model and the Modelica simulation model is realized by OSLC, which improves the iteration of system design.

C. Discussion

1) QUALITATIVE ANALYSIS: In this case study, we adopt OSLC specification to integrate heterogeneous models between different tools, which has several advantages over traditional integration approaches.

Compared to developing point-to-point tool interfaces, which are limited by the ability of tools to provide external APIs. OSLC uses the core model to specify semantic frameworks. Different adapters can be developed by defining corresponding semantic frameworks according to user requirements, which are more scalable and reusable. In addition, compared to data middleware (such as XML and JSON) for static data replication, using OSLC specification, the OSLC adapter translates models, data and tool operations into online OSLC services with identified URLs. Other tools can access the OSLC services through the URLs. Besides, the translated services support bidirectional information exchange which can change dynamically as models are modified, thus reducing data redundancy. Therefore, OSLC specification promotes the interoperability of tools and models in the development process.

Though OSLC supports model integration in the system design of RobotR3, several limitations exist: Firstly, the ability to create models by OSLC service is not realized in this paper, which can be carried out in future work. Secondly, only one adapter was developed in this paper to integrate the Modelica model and data. The same approach can be used to develop multiple adapters for different stages in MBSE to construct digital thread [19] and support data integration in the whole lifecycle.

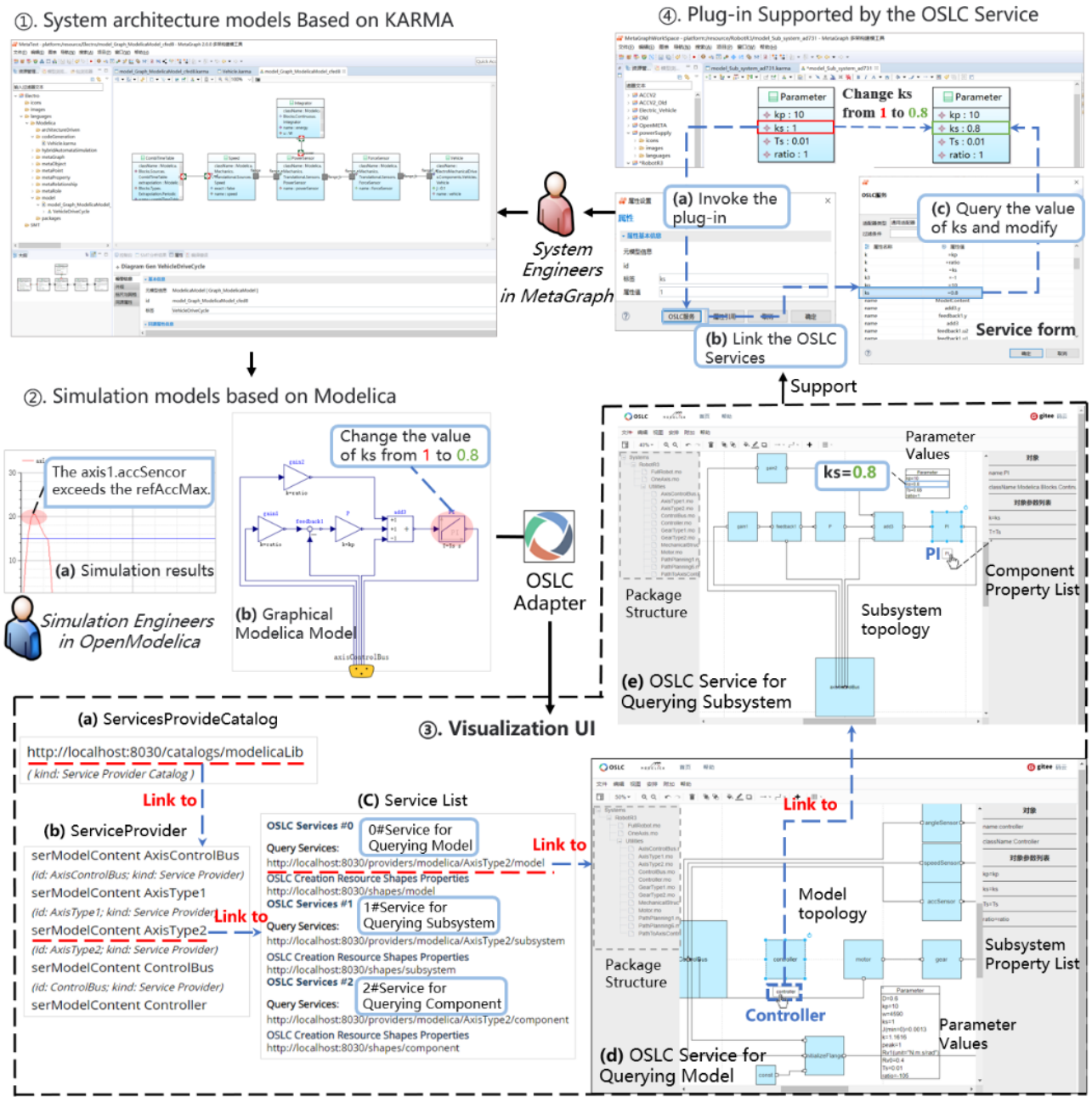

Fig. 2. Integration of KARMA and Modelica models of RobotR3 System.

2) QUANTITATIVE ANALYSIS: The numbers of Modelica model elements translated into online OSLC services in the RobotR3 system are shown in Table 2. There is 1 URI of the *ServiceProviderCatalog*, 12 URIs of the *ServiceProvider*, 12 URIs of the *Service*, and 110 URIs of the *Resource*. The Modelica adapter supports multi-level parsing of models, from system to subsystem to component and finally to properties, which facilitates model integration with small granularity. The identified URIs generated by the adapter can be accessed by other tools thus indicating the interoperability between models in the development process. By linking these OSLC services to other tools, the integration of models can be realized.

V. CONCLUSION

In this paper, we have proposed a service-oriented approach to integrating models in MBSE. The elements and APIs of the KARMA and Modelica model are expressed based on the OSLC core model, and the unified semantics is realized. The OSLC adapter and service query plug-in are developed to realize the exchange of model information across tools. Then, an integration case of the RobotR3 system through OSLC service is constructed to demonstrate that the proposed approach is effective for the integration of models and tools.

TABLE I
THE SERVICE SEMANTICS OF ROBOTR3 SYSTEM

OSLC concepts	**Model/Operation**	**Model/Operation instance**	**Servicesemantics tags**
ServiceProviderCatalog	Package	RobotR3	<oslc:ServiceProviderCatalogrdf:about="http://.../catalogs/RobotR3">... </oslc:ServiceProviderCatalog>
ServiceProvider	Model	FullRobot.mo	<oslc:ServiceProviderrdf:about="http://.../provider/FullRobot">... </oslc:ServiceProvider>
Service	Query	Query a model topology	<oslc:Service>... <oslc:queryCapability>... <oslc:queryBase rdf:resource="http://.../Model/FullRobot">... </oslc:Service>
Resource	Model topology	FullRobot.mo	<modelica:Model rdf:about="http://.../Model/FullRobot">... </modelica:Model>
Resource	Subsystem	Controller.mo	<modelica:Subsystemrdf:about="http://.../Subsystem/Controller">... </modelica:Subsystem>
Resource	Block	PI controller	<modelica:Componentrdf:about="http://.../Component/PIcontroller">... </modelica:Component>

TABLE II
THE METRICS IN THE CASE STUDY OF ROBOTR3 SYSTEM

OSLC concepts	**Modelica concepts**	**Counts**
ServiceProviderCatalog	Package	1
ServiceProvider	Model	12
Service	QueryCapability	12
Resource	Model topology	12
	Subsystem	10
	Component	88

REFERENCES

[1] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 101–111, 2011.

[2] T. Doran, "Ieee 1220: for practical systems engineering," *Computer*, vol. 39, no. 5, pp. 92–94, 2006.

[3] Q. Wang and C. Chatwin, "Key issues and developments in modelling and simulation-based methodologies for manufacturing systems analysis, design and performance evaluation," *The International Journal of Advanced Manufacturing Technology*, vol. 25, no. 11, pp. 1254–1265, 2005.

[4] P. Fritzson and V. Engelson, "Modelica — a unified object-oriented language for system modeling and simulation," in *ECOOP'98 — Object-Oriented Programming*, E. Jul, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 67–90.

[5] D. Johnson and S. Speicher, "Open services for lifecycle collaboration core specification version 2.0," 2013.

[6] J. Lu, J. Wang, D. Chen, J. Wang, and M. Törngren, "A service-oriented tool-chain for model-based systems engineering of aero-engines," *IEEE Access*, vol. 6, pp. 50 443–50 458, 2018.

[7] J. Lu, G. Wang, J. Ma, D. Kiritsis, H. Zhang, and M. Törngren, "General modeling language to support model-based systems engineering formalisms (part 1)," in *INCOSE International Symposium*, vol. 30, no. 1. Wiley Online Library, 2020, pp. 323–338.

[8] J. Guo, G. Wang, J. Lu, J. Ma, and M. Törngren, "General modeling language supporting model transformations of mbse (part 2)," in *INCOSE International Symposium*, vol. 30, no. 1. Wiley Online Library, 2020, pp. 1460–1473.

[9] T. Johnson, A. Kerzhner, C. J. Paredis, and R. Burkhart, "Integrating models and simulations of continuous dynamics into sysml," *Journal of Computing and Information Science in Engineering*, vol. 12, no. 1, 2012.

[10] G.-D. Kapos, A. Tsadimas, C. Kotronis, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "A declarative approach for transforming sysml models to executable simulation models," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 6, pp. 3330–3345, 2019.

[11] S. Götz, M. Tichy, and R. Groner, "Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review," *Software and Systems Modeling*, vol. 20, no. 2, pp. 469–503, 2021.

[12] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger, "Reuse in model-to-model transformation languages: are we there yet?" *Software & Systems Modeling*, vol. 14, no. 2, pp. 537–572, 2015.

[13] D. C. Schmidt, "Model-driven engineering," *Computer-IEEE Computer Society-*, vol. 39, no. 2, p. 25, 2006.

[14] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and simulation modeling using sysml," in *2007 Winter Simulation Conference*. IEEE, 2007, pp. 796–803.

[15] R. Chen, Y. Liu, and X. Ye, "Ontology based behavior verification for complex systems," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 51739. American Society of Mechanical Engineers, 2018, p. V01BT02A038.

[16] J. Lu, J. Ma, X. Zheng, G. Wang, H. Li, and D. Kiritsis, "Design ontology supporting model-based systems engineering formalisms," *IEEE Systems Journal*, pp. 1–12, 2021.

[17] S. Kwon, L. V. Monnier, R. Barbau, and W. Z. Bernstein, "Enriching standards-based digital thread by fusing as-designed and as-inspected data using knowledge graphs," *Advanced Engineering Informatics*, vol. 46, p. 101102, 2020.

[18] N. S. Zadeh, L. Lindberg, J. El-Khoury, and G. Sivard, "Service oriented integration of distributed heterogeneous it systems in production engineering using information standards and linked data," *Modelling and Simulation in Engineering*, vol. 2017, 2017.

[19] S. Wu, J. Lu, Z. Hu, P. Yang, G. Wang, and D. Kiritsis, "Cognitive thread supports system of systems for complex system development," in *2021 16th International Conference of System of Systems Engineering (SoSE)*, 2021, pp. 82–87.