A Modeling Method for Model-based Analysis and Design of a System-of-Systems

Young-Min Baek, Zelalem Mihret, Yong-Jun Shin, and Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{ymbaek, zelalem, yjshin, bae}@se.kaist.ac.kr

***Abstract*—In recent years, a domain of Systems-of-Systems (SoS) has emerged due to the needs of utilizing collective and collaborative system capabilities. As interest in SoS engineering has grown, this study focuses on the model-based analysis and design of an SoS, and we propose a general-purpose modeling method for the model-based SoS engineering (MBSoSE). Based on requirements that modeling methods of MBSoSE approaches should fulfill, required model types are identified (19 model types) and they are classified on their different modeling purposes and concerns (6 model categories). Model types are meta-modeled using the ADOxx Metamodeling Platform and they are implemented as modeling languages in a tool, called *SIMVA-SoS Modeler*. Using the modeling tool developed, we designed two different SoS cases and their scenarios that can be utilized as inputs of simulation and verification tools. Through the case studies, overall applicability of our modeling method for MBSoSE is evaluated and specific modeling results are provided as base reference models.**

***Keywords*—Software System Modeling, Software Modeling Tool, Model-based Systems-of-Systems Engineering (MBSoSE), Simulation Model**

I. Introduction

In recent years, a number of software-intensive system domains leverage collective capabilities of multiple software, based on emergent behaviors and goal-oriented orchestration [1], [2], [3]. Many networked and distributed systems, for example, have been extended to other specialized system domains, such as Internet-of-Things (IoT) [1], Cyber-Physical Systems (CPS) [2], and Multi-Agent Systems (MAS) [3]. They target to take advantages by employing software-intensive components that have potential capabilities to contribute to achieve a shared and high-level common goal(s). Meanwhile, the quality of those integrated software-intensive systems has been improved and their size and complexity have been getting larger. With these changes, a more complicated system domain has emerged, which is a *System-of-Systems*, called *SoS* for short. The ISO/IEC/IEEE 15288 Annex G defines the SoS as a system type that brings together a set of constituents for a goal-based task that none of the individual constituent can accomplish on its own [4].

Compared to conventional software/systems, an SoS possesses unique combinations of characteristics that make engineering more challenging. The most distinguishing differences on system characteristics are from constituents' darkness [5]. For SoS engineers, comprehensive understanding and control over individual performers are nearly impossible. Also, the constituents not only decide their behaviors, manage resources exclusively by their own, but they can also exist and operate with or without the existence of an SoS. Although many existing distributed and networked systems are comprised of multiple component systems that are capable of performing required functionalities, they are not independent both from each other and from the higher-level system. On the contrary, constituent systems (CSs) of an SoS are highly autonomous and they can be independently designed in terms of operation and management. These characteristics cause uncertain interactions among multiple CSs. In turn, this may lead to various emergent behaviours that SoS can utilize to achieve its goal.

A goal of SoS engineering (SoSE) is to drive the emergent behaviour of multiple CSs in the right direction (i.e., goal achievement) in the midst of uncertainties. According to existing studies [6], [7], however, predicting and engineering of all possible emergence are almost infeasible due to the scale and complexity of an SoS. In addition, applying conventional system/software engineering techniques for SoSE has much difficulties because constituents should be regarded as black-box components in many cases. Model-based software/system engineering, therefore, should be refined, improved, and specialized for systematic SoS engineering.

In this paper, in order to address the scale and complexity issues caused by SoS characteristics, we focus on a model-based SoS engineering (MBSoSE) approach. Among various MBSoSE approaches, this study proposes a general-purpose modeling method for model-based analysis and design of an SoS that is envisioned to deal with engineering issues during the analysis and design phases of SoSE. Our modeling method can serve dual purposes for methodologists and modeling engineers (i.e., system architects/designers). A domain methodologist can develop a domain-specific design method at meta-level by extending our method as a base method, and a practitioner who designs an SoS can perform modeling activities on the modeling method developed by the methodologists. Our modeling method has been developed as a tool called *SIMVA-SoS Modeler (A Modeling Tool for Simulation-based Verification and Analysis of SoS*, and this tool supports general-purpose modeling technique and procedure for various SoS domains.

Our method is not only developed as a technical approach, it is also implemented as an open-sourced modeling tool that conforms to the SoS architecture and the SoS meta-model. Major contributions of this study can be summarized as follows:

- This study proposes a general-purpose modeling method specialized for the model-based SoS engineering (MB-SoSE). By defining 6 model categories and 19 model types developed in the method, our method supports extensions to develop domain-specific SoS modeling methods.
- Our modeling method is implemented as a modeling tool, called *SIMVA-SoS Modeler*. This tool can be used as a means of producing simulation models and specifications that can be used as inputs to the simulation and verification tool in a consistent and systematic way.
- We implemented the modeling method as an open-sourced modeling toolkit with an extensible library, supported by the *Open Model Laboratory (OMiLab)* community. Therefore, methodologists can utilize the library to systematically develop their own domain-specific modeling languages (DSMLs) or methods.

II. Related Work

In this section, model-based engineering approaches that can be utilized for SoS engineering are investigated. Nielsen *et al.* conducted a systematic literature review on model-based SoS engineering (MBSoSE) studies and investigated the approaches in terms of eight dimensions to position an SoS engineering problem and corresponding approaches [7]. Based on the comprehensive analysis by the study, some of the representative modeling methods and analysis methods are introduced in this section.

Modeling Methods and Architectural Frameworks for MBSoSE. The *Unified Modeling Language (UML)* is a standardized general-purpose modeling language in the field of software engineering [8]. It is used for design, specification, and documentation of artifacts created during information system development process. Extended from UML, *SysML (System Modeling Language)* is developed as an object-oriented, general-purpose architecture modeling language for systems engineering applications [9]. In addition to the specification, design, and analysis; SysML also enables the verification and validation of system properties. For these purposes, additional diagrams such as the *parameteric diagram* are introduced in SysML. Both UML and SysML have been considered to be technologies that enable general-purpose Model-Based Software/System Engineering (MBSE) and Model Driven Development (MDD) [10].

Existing architectures for conventional systems are established on the assumption of tightly coupled system components, and has formal foundations that only cope with specification of architectures that are static or dynamic (but limited to the anticipated reconfigurations known at design-time). This has significant contribution to the lack of models and tools that enables SoS engineers to deal with unique SoS characteristics such as the analysis of uncertainties, emergent

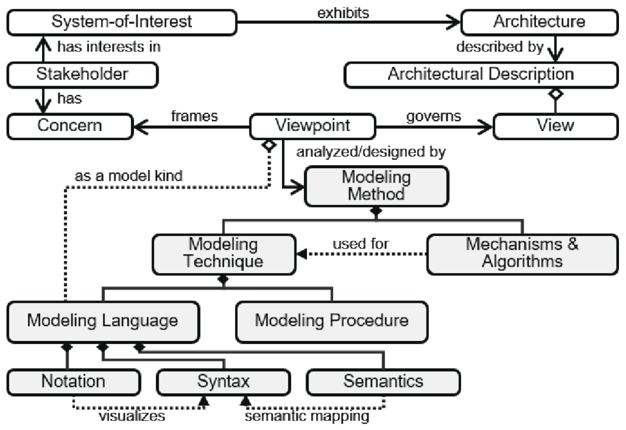

Fig. 1. Components of a Modeling Method and an Architecture

behaviors, and evolutionary development. Another challenge is due to size and complexity of SoS. Existing industrial tools and practices with a modeling language need to be augmented in order to fully express an SoS and its both individual and emergent behaviors under the uncertainties. This leads to sub-optimal design and expensive rework during integration [11].

The following three architectures and modeling frameworks are highly related with SoS or directly developed to deal with unique concerns of SoS. The *TOGAF* (The Open Group Architecture Framework), which is an enterprise architecture, has four perspectives: business, data, application, and technique. It provides architecture and management methods throughout the life cycle of enterprise IT systems based on enterprise requirements [12]. The *DoDAF* (The DoDAF Architecture Framework), which is a system modeling framework that is optimized to analyze and design a large-scale defense system, defines several viewpoints, such as capabilities, data, operational, project, service, standards, and systems [13]. It can be used to facilitate effective decision making through organized and consistent information sharing across boundaries of systems, missions, stakeholders, and departments. The *AMADEOS* (Architecture for Multi-criticality Agile Dependable Evolutionary Open SoS) framework supports to architect an SoS at conceptual, logical, and implementation level based on SysML profiles [14]. According to general SoS engineering concerns, the AMADEOS models are analyzed in different viewpoints, such as structure, dynamicity, evolution, dependability and security, time, multi-criticality, and emergence. Compared to the existing architecture frameworks, our method is more extendable based on meta-models, so domain-specific frameworks can be made on our method and tool.

Model-based Simulation and Verification of SoS. One of our previous studies modeled different types of SoS with probabilistic models and verify them using a statistical model checker [15]. The study found that probabilistic models are not only essential to represent nondeterministic behaviors of black-box constituents but also viably proper to simulate uncertain and emergent behaviors. In addition, since conventional model checking approaches have had difficulties to examine a large set of complicated behaviors, statistical model checking

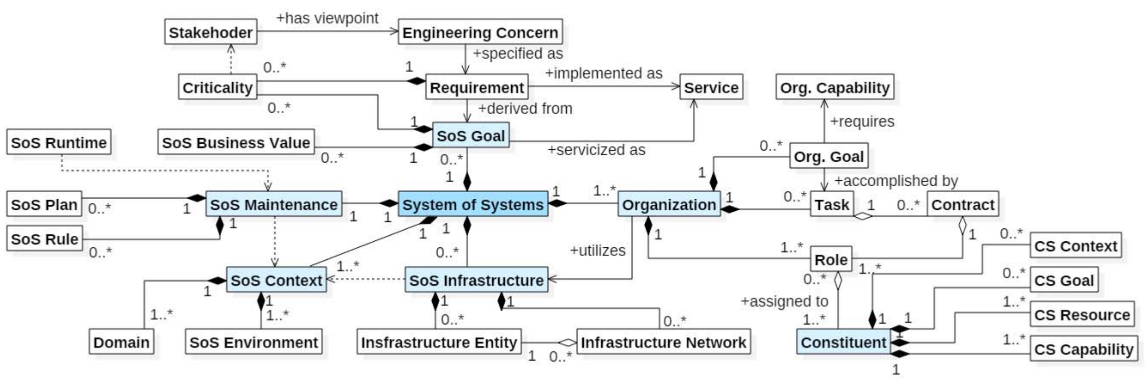

Fig. 2. A Simplified Version of the *Meta-Model for SoS Engineering (M2SoS)*

(SMC) was found to be more feasible and practical for real-world SoS cases. More recently, S. Park *et al.* developed a tool for simulation-based verification and analysis for SoS [16], called *SIMVA-SoS*. The study developed an SMC technique based on the sequential probability ratio test (SPRT), and the tool supports development of probabilistic simulations models and specification of probabilistic scenarios as inputs.

III. Background

In order to newly design a modeling method, a methodologist should know what components need to be developed so that the method fully exploits modeling capabilities. As Figure 1 shows, *stakeholders* have interest in a *system* and management/engineering *concerns* about the system. For more systematic engineering and management of a system-of-interest, the concerns are framed by concrete *viewpoints* that govern one or more *views* handled by describing an *architecture* and *architectural description*.

Based on a specific viewpoint, a modeling method can be invented by developing a modeling technique that consists of a *modeling language(s)* and *modeling procedures*. The modeling language contains three essential constructs: *notation*, *syntax (grammar)*, and *semantics*. The *notation* is concerned with representation of various concepts in the modeling language. In the model-based system/software engineering approach, in most cases graphical symbolisms (i.e., icons) are used for the purpose of visual representation. The *syntax* (and its *syntactic grammar*) defines whether or not the notations (symbols of a modeling language) are correctly composed to a valid form of a language, and it determines how modeling components are put together. On the other hand, the *semantics* is about meanings associated with or intended by the whole model, components, and relations. Based on the notations, syntax, and semantics defined, *semantic mapping* associates meanings embedded in the syntax of the modeling language with the semantic domain. From a modeling engineer's perspective, a modeling method also should provide effective *modeling procedure* to optimally utilize the capabilities of the modeling language towards his or her intended goal. In addition, a modeling method provides *mechanisms and algorithms* for various purposes such as inspection, simulation, visualization, transformation, evaluation, and so on.

A. *Meta-Model for SoS Engineering*

Our base model that we consider to develop a modeling method for MBSoSE is a meta-model whose name is *M2SoS*, which stands for a *Meta-Model for Systems-of-Systems (Engineering)* [17]. As Figure 2 shows[1], the M2SoS provides a holistic high-level view of an SoS, which includes major SoS entities. Using the M2SoS, both SoS engineers and managers can be guided during the whole MBSoSE phases, and they can also evaluate the completeness or correctness of their analysis and design results (e.g., models, artifacts). Another capability of the M2SoS is that SoS engineers can conduct the ontological analysis for various SoS domains. This enables SoS stakeholders (i.e., both SoS-level stakeholders and constituent-level stakeholders) to establish a common knowledge base of their target SoS. If the ontology is developed based on the M2SoS and is stored in a shared project repository of an SoS, commonly used vocabularies can be accessed and utilized for various purposes, such as for communications.

Based on the M2SoS, an SoS architecture can be developed for conceptualizing various SoS viewpoints. Also, required classes of an SoS and corresponding modeling languages can also be developed that conform to the architectural definition of a general SoS. While defining classes, there are several requirements for general-purpose metamodeling of SoS. First, modeling entities should be distinguished in terms of their belongings and affiliations while considering levels and layers defined in the SoS architecture. Second, metamodeling of an SoS should consider goal-oriented engineering (e.g., GORE), since most SoSs have high-level common objectives either explicitly or implicitly. The third requirement mainly focuses on the uncertainties inherent in an SoS. Environmental factors that produce most of the uncertainties should be regarded as one of the first-tier entities [19]. In the next section, these entities of the M2SoS are utilized to develop a high-level conceptual SoS architecture and corresponding model types.

[1]The meta-model of Figure 2 is an abstracted version of the original M2SoS. You may refer to the complete documentation of M2SoS from our website [18].

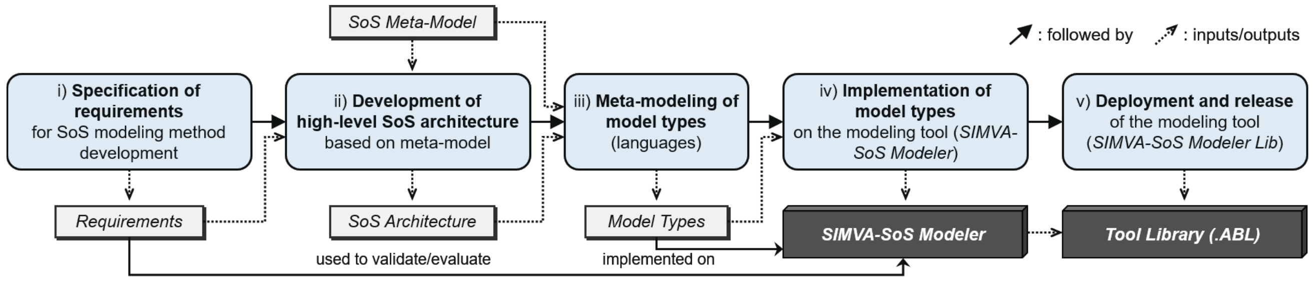
SoS Meta-Model
: followed by
: inputs/outputs
i) Specification of requirements for SoS modeling method development
ii) Development of high-level SoS architecture based on meta-model
iii) Meta-modeling of model types (languages)
iv) Implementation of model types on the modeling tool (SIMVA-SoS Modeler)
v) Deployment and release of the modeling tool (SIMVA-SoS Modeler Lib)
Requirements
SoS Architecture
Model Types
SIMVA-SoS Modeler
Tool Library (.ABL)
used to validate/evaluate
implemented on

Fig. 3. Overall Process to Develop a General-Purpose Modeling Method for SoS Engineering

IV. A Modeling Method for SoSE

A. Overall Approach

This study develops a modeling method and the method is supported by a tool, and overall process to develop the method is illustrated in Figure 3. Step (i) describes requirements specification for developing an SoS modeling method from the perspective of analysis, design, simulation and verification. This step comprehensively analyzes requirements that should be satisfied by generated models for every engineering phase. An output of the first step is a set of requirements to be fulfilled by our modeling method, and the requirements will be used to validate the method afterwards. Based on the collected requirements, Step (ii) develops a high-level SoS architecture and identifies required model types (i.e., model kinds). For the systematic identification, this step defines conceptual levels and layers of the high-level SoS architecture by utilizing the SoS meta-model as inputs. In addition, the model types are classified into 6 model type categories to generally cover SoS concepts/entities and meet the requirements of Step (i). Step (iii) describes actual development of a modeling method by meta-modeling all the model types identified in the previous step. This step produces several model types (i.e., meta-models) according to the categorization of Step (ii), and model types are implemented on the modeling tool on Step (iv) as a tool, called the *SIMVA-SoS Modeler*. As explained in Section III, the tool includes multiple modeling languages consisting of diagrams and procedures to support our modeling technique. The tool, in Step (v), is deployed and released on the open-modeling community as a tool library, which can be extended by any other domain methodologists.

B. Requirements for Developing a Modeling Method

1) Requirements for Analysis and Design of SoS:

a) Identification of SoS entities and boundary: One of the major purposes of using an architecture (and architectural description) is to establish a specific viewpoint by identifying system entities and relationships among entities. The entities of an SoS include software and hardware components, agents, services, human and organization factors, data, and so on. In addition, the boundary of an SoS should be somehow determined for effective MBSoSE. However, the boundary of an SoS is not static in most cases, but it could be ambiguous, fluid, and negotiable, even at runtime. Nonetheless, our modeling method should be able to provide the criteria for what entities are included in and excluded from an SoS, by providing finite set of classes in model types.

b) Support of goal-oriented analysis and design: As a branch of requirements engineering, conventional system/software engineering have conducted Goal-Oriented Requirements Engineering (GORE) to acquire concerns and identify goals, derive corresponding requirements, and capture alternatives and conflicts [20]. An SoS must have a shared common goal either implicitly or explicitly, and the most important objective of SoSE is to drive emergent behaviors to achieve the goal. Therefore, goal-based analysis and design should be supported by a proper modeling method, and the method should be well-accepted and well-established. Based on the common goals of an SoS, engineers can choose an overall engineering approach; it can be top-down or bottom-up. Unlike a conventional monolithic system or software, since an SoS can consist of legacy systems that are already designed without consideration of belonging to a higher-level system (i.e., SoS), the bottom-up approach might be inevitable choice for some cases. The bottom-up approach can be realized by capability-based analysis of constituents, so the analysis of collective capability of multiple constituents should be supported by our modeling method.

c) Support of traceability between models or between modeled objects: One of the principal reasons for using a modeling tool is to ensure the completeness of models built and the traceability among them. Model traceability help engineers and stakeholders understand associations and dependencies among artifacts and entities [21]. Especially, an SoS has much more entities and artifacts that are interconnected with and interdependent on each other, support of the traceability outweighs any other issues for complexity management and communications.

d) Support of domain-specific extensions: Our goal is to develop a general-purpose (i.e., domain-general) modeling method and tool that can be extended to domain-specific SoS modeling methods by actual domain engineers. Since one general-purpose modeling method cannot cover all domain-specific systems, our modeling method has to properly consider domain-specific extensions. Even though all classes are not used (and instantiated) for every domain, model types of our modeling method must be able to be extended by domain-specific methodologists.

e) Support of ontological analysis: Ontology is a formal and explicit representation of knowledge [22], which describe concepts with their categories, properties, and relations with other concepts. Building an ontology can be significantly valuable because the ontology establishes common vocabularies and common understanding. This enables the refined communication among system stakeholders by reducing terminological/conceptual mismatches among them. Besides, ontology is used to capture system's complexity with structured terminologies and taxonomies, and the method for this is called *ontologial analysis*. The analyzed results can support the decision for the system design and development. We introduced the *M2SoS* to support ontological analysis for SoS engineers as discussed in Section III-A [17]. Our modeling method caters features that enables to utilize the M2SoS capabilities.

2) Requirements for Simulation and Verification of SoS:

a) Consideration on object-oriented (OOP) and agent-oriented programming (AOP): Based on the OOP and AOP paradigms, a design and programming paradigm specialized for MBSoSE can also become an extension of existing system development paradigms. Agents of an agent-based system (e.g., Multi-agent System (MAS)) share a lot of similarities with constituents of an SoS [17]. Both system types have a higher-level common goal(s) to be achieved by collective capabilities of autonomous component systems, so called agents and constituents. However, a constituent system can be much more independent from normal objects or agents and SoS engineers may not have full authority to access or control the constituents. For these reasons, a modeling method for MBSoSE should borrows fundamental concepts and ideas from the OOP and AOP and extend them to be more suitable for constituents of an SoS.

b) Representation of temporal and geographical properties: For more realistic simulation-based analysis, a simulation model(s) generated by our modeling method should be able to represent temporal and geographical information. The temporal information primarily includes time, duration of actions and temporal properties of events. The geographical information includes every simulated object's geolocation that positions the object on a logically or physically designed map. The way of simulation of temporal and geographical properties can differ depending on formalisms of simulation models and simulation engine's mechanism. Among various simulation models, our modeling method considers models for the simulation as discrete-event/time simulation (DES), which simulates behaviors of a system as a discrete sequence of events in logical time.

c) Representation of uncertainties: One of the purposes of performing model-based simulation is to observe and analyze possible set of non-deterministic behaviors with respect to a given model. The observed behaviors can be used as a solution recipe to deal with real-world domain specific problems. Non-deterministic behaviors connote the lack of certainty about when a specific behavior will occur in time. It doesn't necessarily imply the lack of knowledge about the behavior. In the case of SoS, however, due to evolutionary development and emergent behaviors characteristics it is possible that some behaviors may prevail following interactions between constituent systems. These inherent uncertainties can pose risks in achieving SoS goals. Nonetheless, such uncertainties are not avoidable, instead can be utilized for achieving the SoS goals. With regard to this, our modeling tool should support the modeling of rules and policies to deal with uncertainty representation and analysis in view of orchestrating the behaviors towards to achieving the SoS goals.

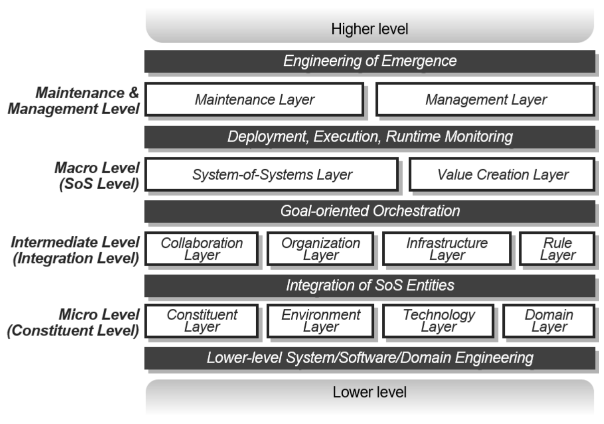

Fig. 4. High-level SoS Architecture Conceptualizing *Levels* and *Layers*

d) Support of dynamic reconfiguration: An SoS and its constituents dynamically change their behaviors and structures, while performing operations. This characteristic of an SoS is called *dynamic reconfiguration* or *(self-)adaptation*, and it is one of major dimensions to position and develop a model-based SoS engineering method [23]. This capability is to undertake both internal and external changes to behaviors, structures (connections), and compositions of an SoS. In order to realize and simulate the reconfiguration capability, input models should be able to be dynamically reconfigured during simulation. Reconfigurable models can be developed by allowing the representation of probabilistic behavior execution, state transitions and structural changes.

e) Capturing and observing emergent behaviors: Emergence of behaviors refers to the synergistic behaviors that cannot be accounted for by a single constituent, and capturing the emergence is a primary reason for simulation-based analysis. In order that simulation models are capable of exhibiting emergent behaviors, autonomy of input constituent models need to be properly represented and simulated. Depending on constituents' autonomy, a simulation engine should let the whole models behave on their own to exhibit the emergence. During the simulation, emergent behaviors can be observed, which can be goal achievement, task accomplishment, or emergent occurrence of accidents/failures. To capture and analyze them, a modeling method should support the specification of scenarios and properties, to guide a simulation and to discover/examine the emergence.

C. High-level SoS Architecture

Compared to general software/system architectures, representing the architecture of an SoS has other challenges. First, an SoS can involve both cyber and physical system components that exhibit highly uncertain and non-deterministic behaviors. In addition, human factors make the architecting more challenging, since SoS engineers should consider not only technical parts but also social or socio-technical aspects. Most SoSs, second, do not have fixed and static system boundaries. The boundary can be dynamically defined according to the engineering concerns, runtime situations, or the evolutionary nature of SoS. This characteristic requires the adaptiveness capability of an SoS, which refers that an SoS is capable of adjusting itself in response to changes both internal and external context. Third, all constituents and component elements are somehow interacting with each other, and SoS engineers should coordinate (i.e., orchestrate) them to accomplish a shared goal(s) or mission(s). In order to effectively integrate the constituent elements, a holistic view should be provided by using proper architectural descriptions. For these reasons, defining a high-level abstract architecture can be a more feasible and effective solution, rather than using lower-level specific approaches (e.g., domain-specific, technology/technique-specific approaches).

In order to provide general-purpose architectural description for various SoSs, our architecture provides a comprehensive view by defining conceptual *levels*, *layers*, and *abstract interfaces*, as depicted in Figure 4. The levels consider managerial and operational contexts, and they are realized by appropriate engineering/management strategies. For each level, one or more layers are defined according to related SoS stakeholders and their engineering concerns, which are analyzed and designed by different model types. Each model type has a specific engineering discipline, thus it means that a single level can compose multiple different disciplines. In our architecture, two types of abstract interfaces are identified, which are *cross-contexts* and *cross-disciplinary*, respectively. The cross-contexts can be defined as interfaces between different levels, and the cross-disciplinary aspects can be considered on the interfaces between different layers. To reflect "open-at-the-top" and "open-at-the-bottom" characteristics of an SoS [24], the architecture does not involve neither specific top-level applications nor fixed bottom-level elements.

D. Definition of Multi-Aspect Model Types

In many general-purpose software/system modeling methods such as *UML* and *SysML*, model kinds are typically classified into three model types: (a) structural models, (b) behavioral models, and (c) interaction models. However, for SoS engineering, internal information of a considerable number of components cannot be fully available for engineering. Not only that, organizations, constituents, infrastructures, and their contexts and environment may have more complicated interconnection with each other. In order to address these concerns, our modeling method classifies model types (MTs) into three categories: *Architectural MT (AMT)*, *Reference MT (RMT)*, and *Verification and Validation MT (VVMT)*. Due to the lack of space, the model types are summarized in Table I detailed definitions and meta-models of each diagram are uploaded on our web page [18].

TABLE I
CLASSIFICATION OF MODEL TYPES

Category	Included Model Types
Architectural Model Types (AMT)	SoS Integration Model (SoSIM) SoS Organization Model (SoSOrgM) SoS Infrastructure Model (SoSInfraM) SoS Environment Model (SoSEnvM) SoS Map Model (SoSMapM)
Reference Model Types (RMT)	
RMT-Goal-based/-oriented Model Types (RMT-GMT)	Goal Decomposition Model (GDM) Requirement Specification Model (RQSM) Rule Specification Model (RLSM) Risk Analysis Model (RAM)
RMT-System Model Types (RMT-SMT)	System Capability Model (SCM) Service Description Model (SDM) Interface Description Model (IDM)
RMT-Operational Model Types (RMT-OMT)	Task Process Model (TPM) State Machine Model (SMM)
RMT-Domain Model Types (RMT-DMT)	Environmental Factor Model (EFM)
Verification & Validation Model Types (VVMT)	Simulation Specification (SimSPEC) Simulation Scenario Specification (SimScnSPEC) Verification Specification (VerifSPEC) Verification Property Specification (VerifPptySPEC)

1) Architectural Model Types (AMTs): The AMTs represent architectural (i.e., structural) viewpoint of an SoS. The main purpose of these model types is to integrate and orchestrate containers, constituents and component entities into an SoS in a model-based manner. To describe SoS structure, organization(s), infrastructure(s), and environment(s) are defined as major container classes, and the *SoS Integration Model* composes them to determine an SoS boundary.

There are four major models to represent architectural aspects of an SoS: *SoS Integration Model*, *SoS Organization Model*, *SoS Infrastructure Model*, and *SoS Environment Model*. The overall interconnection is described and components are integrated by the *SoS Integration Model*, and it has multiple inter-references to other architectural models (organization, infrastructure and environment). Also, for the representation of geographical information, an SoS-level map(s) is defined using the *SoS Map Model*.

2) Reference Model Types (RMTs): The RMTs are basically referenced by the Architectural Model Types to model comprehensive and supplementary information of an SoS, such as system, operations, goals, and domain-specific information. By extending reference model types, multiple and cross-domain SoS engineers can collaboratively conduct domain-specific analysis and design.

a) Goal(-based) Reference Model Types (GMTs): This model type is used to define models that are related to (or based on) SoS-level common goals and objectives. By decomposing and refining SoS goals, SoS engineers can derive and specify requirements, rules, policies, and other factors.

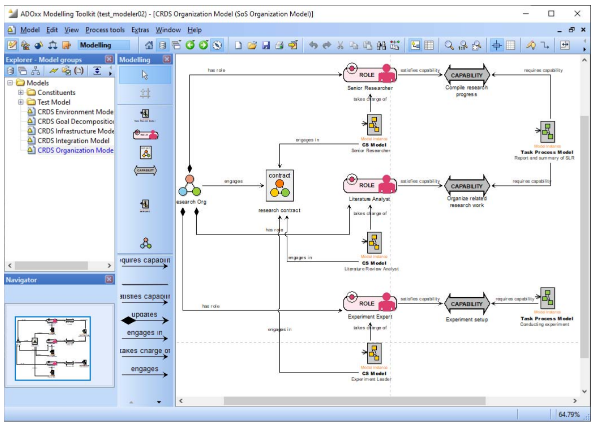

Fig. 5. User Interface of the *SIMVA-SoS Modeler*

b) System Reference Model Types (SMTs): This model type is used to define models that describe system elements and their capabilities (i.e., abilities and capacity of systems), which are utilized/deployed for SoS-level goal achievement. In order to contrive to get collaborative capabilities, interoperability analysis using the SMTs is required.

c) Operational Reference Model Types (OMTs): This model type is used to define models that represent behavioral aspects of SoS entities, such as task (and business) processes, actions, other behaviors. Behaviors of an SoS can be classified into two categories; one is a collective behavior (i.e., collaboration, cooperation) by multiple constituents and the other is an individual behavior performed by a single constituent. The collective behavior is described by the *Task Process Model*, and the individual behavior is modeled using the *State Machine Model* and action specifications only if the internal information can be obtained by an SoS engineer.

d) Domain Reference Model Types (DMTs): This model type is used to define models that describe information of environmental factors, domain, and context. In the current version of our method only includes the *Environmental Factor Model* that represents relations between environmental factors and dynamic changes of the factors for specification and analysis of SoS environment and more detailed information of the model can be found in our previous work [19].

3) Verification and Validation Model Types (VVMTs): The VVMTs specify input artifacts/specifications for execution time, which mainly focuses on V&V phases, such as simulation, testing, and verification (model checking). These specification model types are basically designed for producing inputs of the simulation-based statistical model checking tool, called SIMVA-SoS [16]. For analysis using the SIMVA-SoS, *Simulation Scenario Specification* is needed for intended execution of a simulation, and *Verification Property Specification* is required for (statistical) model checking of the simulation model.

E. Development of a Modeling Tool: SIMVA-SoS Modeler

The *ADOxx Metamodeling Platform* was originally developed and released by the *OMiLAB* of the University of Vienna. Along with other frameworks for modeling tool development (e.g., EMF), the ADOxx framework is known as one of the most innovative meta-modeling tools to model modeling methods. The framework provides extensible functions, libraries and it also supports the whole development of a modeling method (or methodology). This enables efficient development of a GUI-based modeling tool [25]. The ADOxx platform mainly supports metamodeling approaches, and we can easily develop a modeling language by defining classes, relation classes, and their attributes and operations at meta-level.

Our modeling tool is developed based on the ADOxx meta-modeling platform, and it is specialized to address and implement unique concerns of system of systems engineering methods. The *SIMVA-SoS* stands for *SIMulation-based Verification and Analysis for SoS*, and this supports model-based SoS engineering mainly for analysis, design, validation and verification [16]. The SIMVA-SoS can be generally used as a statistical model checking tool, which is performed by repeating multiple times of simulations. In order to simulate an SoS, the SIMVA-SoS requires simulation models, configuration, and optionally a scenario as inputs. The *SIMVA-SoS Modeler* is hereby used for SoS engineers to build a simulation model(s) of an SoS in a systematic way, by utilizing model types we defined in Section IV.

The *SIMVA-SoS Modeler* supports several functionalities for SoS engineers to perform the modeling activities. The current version of our tool supports modeling of 19 model types, and a user (i.e., SoS modeling engineer) can select model types by following a modeling procedure of a chosen approach (e.g., top-down or bottom-up). As shown in Figure 5, the user can perform graphical modeling using modeling interfaces (e.g., toolbox, canvas, dialogs), and the tool checks if a modeling rule or cardinality is violated. After a model is created, it can be exported to a file of the XML- or ADL(ADOxx Model Language)-formatted file, and other external model files also can be imported into our tool. The tool and its details are described on our website [18].

The ADOxx community is based on the open model initiatives, and a main purpose of the initiatives is to enable the development of enterprise reference models collaboratively, and promote sharing of developed software products to the extent that everyone can copy, use, modify and (re)-distribute in an open and public process without restriction[2].

V. Case Study

A. Case Scenario Design

To assess the modeling capability of the *SIMVA-SoS Modeler*, we selected two SoS case systems, which can cover different SoS types (directed (DIR), acknowledged (ACK), collaborative (COL), and virtual (VIR), proposed by Maier's study [26] and Dahmann and Baldwin's study [27]). The SoS type is determined by explicitness of SoS-level goal, management (enforced, encouraged or operational-independent), and ownership/governance (subordinated or

[2]https://www.adoxx.org/live/home

TABLE II
CLASSES AND INSTANCES OF SOS CASES

(a) Cleaning Agents SoS (CASoS)

Classification	Class	Instance
Organization	`Organization`	*RoomCleaningOrg*
		SequentialCleaningTask
		PrallelCleaningTask
Constituents	`CS`	*SweepingRobot*
		MoppingRobot
		SweepingMoppingRobot
Infrastructure	`System`	*WindowControllerSystem*
	`Service`	*DustMeasuringService*
	`Resource`	*Broom*
		Mop
Environment	`Environment Element`	*FloorTiles*
		Window
		OutdoorDust

(b) Incident Response SoS (IRSoS)

Classification	Class	Instance
Organization	`Organization`	*IncidentResponseOrg*
		- IncidentSceneMngOrg
		– CasualtyManagementOrg
		– FirefightingOrg
		- MedicalTreatmentOrg
	`Task`	*IncidentResponseTask*
		- IncidentSceneMngTask
		– CasualtyRescueTask
		– FirefightingTask
		- PatientCareTask
Constituents	`CS`	*CommandCenter*
		FirefighterAgent
		RescueAgent
		PTSAgent (Ambulance)
		MedicalCenter
Infrastructure	`System`	*RadioCommunicationSystem*
		TrafficControlSystem
	`Service`	*WeatherForecastingService*
		OccupancyCheckingService
	`Resource`	*RescueEquipment*
		FirefightingEquipment
		First-aidEquipment
		SurgicalEquipement
		Beds
Environment	`Environment Element`	*WeatherCondition*
		Fire
		Patients

managerial-independent) [15]. The first SoS case is a *cleaning agents SoS* that collaboratively cleans a room by employing autonomous robot agents, and the second case is an *incident response SoS* that performs missions to handle an accident or incident, which may result in casualties.

- [COL-type] *Cleaning Robot Agents SoS (CASoS)*: The CASoS case was developed as a collaborative-type SoS example, and major components are described in Table II-(a). This SoS has an organization of autonomous cleaning robots who co-operate to clean a room. An SoS-level goal is to clean all tiles of a room and we assume actions of mopping a tile should follow sweeping actions for cleaning all tiles. The SoS goal can be achieved in two ways. The first is a sequential task that mopping agents mop all tiles after they finish sweeping the tiles, and the other is a concurrent task that sweeping and mopping agents work together in parallel.
- [DIR/ACK-type] *Incident Response SoS (IRSoS)*: Another SoS example is designed as a directed or acknowledge-type SoS, and components are listed in Table II-(b). When an incident occurs, such as a large fire incident happening at a high rising building, the SoS should respond to the emergency situation that can cause mass casualties. The two major tasks are to rescue patients from the incident area and to extinguish the fire. To achieve the tasks, *Firefighters*, *Ambulances*, and *Hospitals* cooperate with each other according to SoS-level goals, requirements, based on their individual and collaborative capabilities.

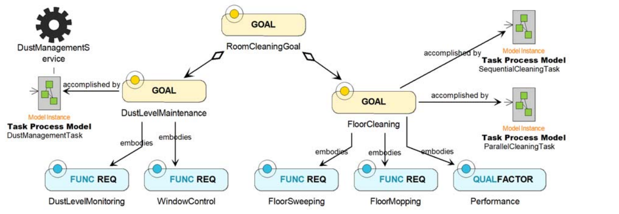

Fig. 6. Goal Decomposition Model of CASoS

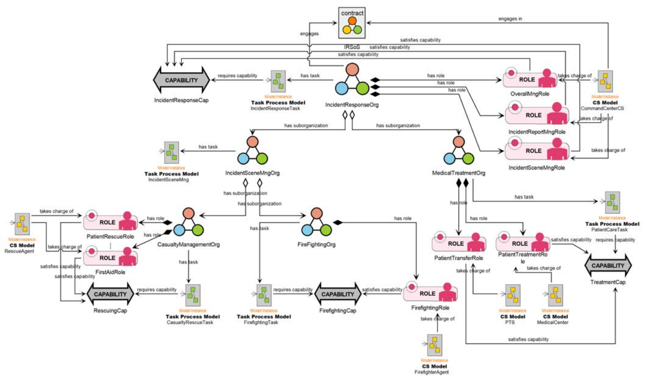

Fig. 7. An SoS Organization Model of IRSoS

B. Modeling Case Scenarios using SIMVA-SoS Modeler

Both SoS cases should explicitly have SoS-level common goal(s) that organizations and constituents pursue. Figure 6 depicts a modeled *Goal Decomposition Model* of the *CASoS* that analyzes a goal tree including the highest-level goal and subgoals. In order to systematically elicit and refine the goals, classes of *Goal*, *Requirement*, *Task*, and required relation-classes (e.g., decomposition, means-ends) are instantiated.

Two case scenarios have different types and structures of organizations; The *CASoS* case exhibits totally collaborative behaviors based on robot agents' autonomous capabilities, but the *IRSoS* case is managed, guided, and governed by specific SoS-level managers as the *SoS Organization Model* of Figure 7 shows. This means that the CASoS case does not need to mandatorily include *Task Process Models*, but the IRSoS does. One of the required tasks of the IRSoS is modeled in Figure 8. The model of *ManageIncidentTask* represents required working flows using activities, and the activities are related to corresponding subtasks, roles, constituents, and resources.

As explained in Section V-A, a modeling method that can be used in general purposes should support the variety of modeling procedures and approaches, such as top-down/bottom-up. In *SIMVA-SoS Modeler*, the procedures can be determined by

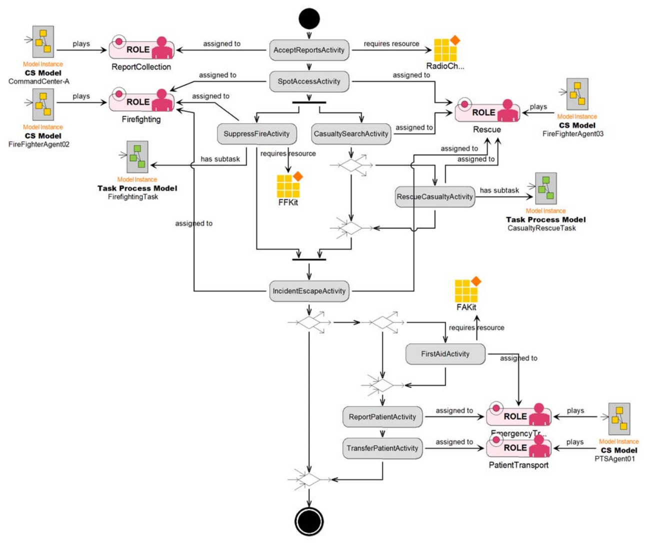

Fig. 8. A Task Process Model of IRSoS: *ManagingIncidentTask*

ordering the modeling activities. For example, we designed the CASoS case in a bottom-up way because unit robots are not originally designed for higher-level collaboration and SoS-level integration. On the contrary, the IRSoS or MCI-response SoS [16] considers goal-based cooperation and interactions when the SoS is designed, thus a top-down approach fits to this case. Other modeling results are uploaded on our web [18].

VI. Evaluation and Discussion

A. Evaluation of Case Study

Our modeling method, its model types and case studies are evaluated based on the requirements listed in Section IV-B.

Validation of requirements for SoS analysis and design. In order to identify SoS entities and boundary, our model types are categorized depending on whether they are inside or outside of an SoS. An *SoS Integration Model* integrates every entities, but the entities in the SoS boundary are mainly modeled by *SoS Organization Model* and *SoS Infrastructure Model*. References to entities outside of the boundary are modeled by *SoS Environment Model*, and additional reference models can be used to describe supplementary information, such as contexts, standards, technologies, and so on. Our modeling method and tool supports the goal-oriented requirements engineering by providing *Goal Decomposition Model*, and a modeling engineer can elicit and refine the goals. According to a chosen approach, a top-down or bottom-up goal modeling can be performed. Based on the integration model and its derived goal model(s), an engineer can effectively trace objects by following their inter-references (model pointers) to other SoS tasks and objects.

In order to support domain-specific extensions and their ontological usages, the modeling tool is first released as a general-purpose (i.e., domain-general) modeling library (*SIMVA-SoS Modeler v3.1*). The library can be extended by defining domain-specific classes using the *ADOxx Development Toolkit*, and the extended languages can be considered as domain-specific modeling languages (DSMLs). If modeling results are produced based on the DSML, a set of objects defined in the models can be exported and collected to provide common vocabularies for model-based ontological analysis of an SoS.

Validation of requirements for SoS simulation and verification. Most well-known simulation models have their own formalisms, and we formalized required models by metamodeling classes and relationships for each model. In model types of our modeling method, every class can be converted into objects/agents for both object- and agent-oriented programming and simulation. For better modularization and reuse of classes, we have designed interfaces and abstract SoS classes both in the tool and in Java-based simulation model. The top-level abstract class is **`SimObject`**, and classes inheriting **`SimObject`** can be specialized into other abstract classes, such as **`SimActionableObject`** and **`SimNonActionableObject`** (See detailed definitions of abstract classes in [18]). For system classes, behaviors can be probabilistic and instances of *SimAction* class have temporal properties, such as duration and time constraints.

In the SIMVA-SoS, to support dynamic reconfiguration during the simulation, some of our models allow runtime manipulations by external actors or events. Since one of the major reasons to perform the reconfiguration is changes of object status, our *Task Process Model* can be dynamically determined in terms of role assignment, performers, and task execution paths. Also, each *System Capability Model* holds its own knowledge base to dynamically make decisions to perform actions or state changes by assessing conditions, cost/benefit, and priorities of actions. These dynamic changes and interactions lead to emergence of behaviors, and collaborative goal achievement or occurrence of failures can be observed and analyzed during the simulation of the *SIMVA-SoS*.

B. Discussion

One of the potential threats to our modeling method can be the scalability of SoS models. If the target SoS becomes more complicated, the number of model instances, objects, interconnections can be increased exponentially. Even though the *SIMVA-SoS Modeler* does not set upper limits on numbers of models and their elements, but disordered models may lead to the generation of poor simulation models. When extending our method, an SoS methodologist can prevent spaghetti modeling by specifying rules/constraints, which can be the object and relationship cardinality of classes and relations or conversion rules. The rules can be directly implemented by the methodologist using script-based mechanisms & algorithms (written in the *AdoScript* supported by the ADOxx framework).

The current version of the *SIMVA-SoS Modeler (v3.1)* supports exporting and importing models as XML/ADL-formatted files. Generating simulation models require another effort for manual programming, so designing the simulation model may be at the expense of additional cost for MBSoSE While implementing simulation models on *SIMVA-SoS*. To address this issue, we are now developing a *XML2Java* technique/tool that automatically generates simulation models (e.g., Java code, DEVS model) based on modeling outputs of *SIMVA-SoS Modeler*.

VII. Conclusion

This study focuses on a model-based SoS engineering (MBSoSE) approach by proposing a general-purpose SoS modeling method. For the development of a general-purpose SoS modeling method, we first identify requirements that should be commonly fulfilled by MBSoSE approaches. Based on the requirements, model types and their meta-level classes are developed and modeling activities are supported by a tool, called the *SIMVA-SoS Modeler (A modeling tool for SIMulation-based Verification and Analysis of SoS)*. The model types implemented in our tool conform to the high-level SoS architecture and SoS meta-model. Using the modeling tool, two domain-specific SoS cases are modeled for the validation of requirements. For both cases, evaluation results show that the requirements are fulfilled appropriately and our modeling method can effectively support the representation of an SoS-of-interest for the MBSoSE. Also, the modeling results can be exported as input files for simulation and verification purposes, especially for the *SIMVA-SoS* [16]. In future work, we aim to develop a model-driven approach that automatically generates object/agent-oriented simulation code of modeling outputs made using *SIMVA-SoS Modeler*.

Acknowledgement

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) and Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (2017M3C4A7066212).

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," Future generation computer systems, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] K. Kim, "Challenges and future directions of cyber-physical system software," in 2010 IEEE 34th Annual Computer Software and Applications Conference. IEEE, 2010, pp. 10–13.

[3] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The gaia methodology," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 12, no. 3, pp. 317–370, 2003.

[4] L. Yang, K. Cormican, and M. Yu, "An ontology model for systems engineering derived from iso/iec/ieee 15288: 2015: Systems and software engineering-system life cycle processes," World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 11, pp. 1–7, 2016.

[5] J. P. van Gigch, "General systems theory. by lars skyttner. published by macmillian press, london, 1996, isbn 0 333 61833 5," Systems Research and Behavioral Science, vol. 14, no. 2, pp. 149–150, 1997.

[6] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige, "Large-scale complex it systems," Communications of the ACM, vol. 55, no. 7, pp. 71–77, 2012.

[7] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," ACM Computing Surveys (CSUR), vol. 48, no. 2, pp. 1–41, 2015.

[8] O. A. Specification, "Omg unified modeling language (omg uml), superstructure, v2. 1.2," Object Management Group, vol. 70, 2007.

[9] M. Hause et al., "The sysml modelling language," in Fifteenth European Systems Engineering Conference, vol. 9, 2006, pp. 1–12.

[10] J.-P. Tolvanen and S. Kelly, "Model-driven development challenges and solutions: Experiences with domain-specific modelling in industry," in 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD). IEEE, 2016, pp. 711–719.

[11] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, "Features of cml: A formal modelling language for systems of systems," in 2012 7th International Conference on System of Systems Engineering (SoSE), 2012, pp. 1–6.

[12] A. Josey, "An introduction to the togaf standard, version 9.2," W182, The Open Group, Reading, UK, 2018.

[13] D. of Defense, "Dod architecture framework, version 2.0, volume 1: Introduction, overview, and concepts manager's guide," 2009.

[14] A. Babu, S. Iacob, P. Lollini, and M. Mori, "Amadeos framework and supporting tools," in Cyber-Physical Systems of Systems. Springer, 2016, pp. 128–164.

[15] D. Seo, D. Shin, Y. Baek, J. Song, W. Yun, J. Kim, E. Jee, and D. Bae, "Modeling and verification for different types of system of systems using prism," in 2016 IEEE/ACM 4th International Workshop on Software Engineering for Systems-of-Systems (SESoS), 2016, pp. 12–18.

[16] S. Park, Y.-j. Shin, S. Hyun, and D.-H. Bae, "Simva-sos: Simulation-based verification and analysis for system-of-systems," in 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE). IEEE, 2020, pp. 575–580.

[17] Y. Baek, J. Song, Y. Shin, S. Park, and D. Bae, "A meta-model for representing system-of-systems ontologies," in 2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS), 2018, pp. 1–7.

[18] SIMVA-SoS Modeler, 2020 (accessed July 24, 2020), https://sites.google.com/view/sos-modeler.

[19] Y.-J. Shin, Y.-M. Baek, E. Jee, and D.-H. Bae, "Data-driven environment modeling for adaptive system-of-systems," in Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 2044–2047.

[20] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: A systematic literature map," in 2016 IEEE 24th International Requirements Engineering Conference (RE). IEEE, 2016, pp. 106–115.

[21] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," IBM Syst. J., vol. 45, no. 3, p. 515–526, Jul. 2006. [Online]. Available: https://doi.org/10.1147/sj.453.0515

[22] T. R. Gruber et al., "A translation approach to portable ontology specifications," Knowledge acquisition, vol. 5, no. 2, pp. 199–220, 1993.

[23] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," ACM Comput. Surv., vol. 48, no. 2, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2794381

[24] R. Abbott, "Open at the top; open at the bottom; and continually (but slowly) evolving," in 2006 IEEE/SMC International Conference on System of Systems Engineering, 2006, pp. 6 pp.–.

[25] H. Fill and D. Karagiannis, "On the conceptualisation of modelling methods using the adoxx meta modelling platform," Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model., vol. 8, no. 1, pp. 4–25, 2013. [Online]. Available: https://doi.org/10.18417/emisa.8.1.1

[26] M. W. Maier, "Architecting principles for systems-of-systems," Systems Engineering: The Journal of the International Council on Systems Engineering, vol. 1, no. 4, pp. 267–284, 1998.

[27] J. S. Dahmann and K. J. Baldwin, "Understanding the current state of us defense systems of systems and the implications for systems engineering," in 2008 2nd Annual IEEE Systems Conference, 2008, pp. 1–7.