SIMVA-SoS: Simulation-based Verification and Analysis for System-of-Systems

Sumin Park, Yong-jun Shin, Sangwon Hyun, Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{smpark, yjshin, swhyun, bae}@se.kaist.ac.kr

***Abstract*—A System-of-Systems (SoS) is a large and complex system containing multiple Constituent Systems (CS) to achieve common goals. To verify the common goals in an SoS, one of the common approaches is a simulation-based verification. In this study, we develop the simulation-based verification and analysis for system of systems (SIMVA-SoS) tool. SIMVA-SoS consists of interactive simulation and statistical verification. The interactive simulation follows discrete-time and multi-agent simulation structures and enables continuous evolutionary representation. The statistical verification runs the simulation repeatedly and uses the simulation results to statistically determine the satisfaction of verification properties. In the case study, a stimulus was injected into the interactive simulator at runtime to examine the expressiveness of continuous evolution, and statistical verification results were analyzed. The SoS representation through interactive simulation and solving the state explosion problem through statistical verification confirmed the possibility of the simulation-based verification of the SoS.**

***Index Terms*—System of systems, constituent system, interactive simulation, statistical verification**

I. Introduction

A System-of-Systems (SoS) is a large and complex system containing multiple Constituent Systems (CSs) to achieve common goals [1]. Achieving a common goal with multiple CSs is more difficult than achieving a goal in one single system, because it needs complex interactions between CSs. SoS is used in many domains including energy management systems, traffic control systems, and mass casualty incident response systems. These are all safety-critical systems that can cause enormous damage to human resources and the environment when the systems fail. Thus, a systematic verification technique for SoS in various domains is required.

One of the approaches to verify common goals in an SoS is simulation-based verification [2], [3]. The main reasons for choosing a simulation-based verification approach are (1) it is infeasible and impractical to conduct an experiment on a real SoS, (2) the simulation gives insight into the unique characteristics of the SoS and helps in visualization, and (3) it can alleviate the state explosion problem [4] which is one of the main problems in verification.

For the simulation-based verification of an SoS, simulation techniques to suit the characteristics of an SoS are needed. An SoS has five characteristics: managerial and operational independence, geographic distribution, evolutionary development, and emergent behaviors [1]. The first three are related to a CS and the other two, to an SoS. To reflect these characteristics in a simulator, we applied a discrete-time and multi-agent simulation techniques. The discrete-time simulation structure reflects evolutionary development [5], and a multi-agent simulation structure is used to reflect the CS's characteristics [6]. In this study, we define this simulator as an interactive simulator. Most of the existing SoS simulator including our previous work, lack the ability to allow dynamic changes during simulation as all inputs are required to be defined before execution. However, an interactive simulator can reflect evolutionary development features, by allowing users to inject new scenario events during the simulation. Finally, the emergent behavior can be shown during the simulation.

To verify the achievement of common goals, typical verification method such as model checking can be applied [7], [8]. However, owing to the size and complexity of an SoS, the verification efficiency is affected by the state explosion problem. The state explosion problem occurs as the size of the system state space increases exponentially as the number of state variables in the system increases. A statistical model checking (SMC) technique is proposed to improve the efficiency of an exhaustive model checking technique, and it is one of the effective ways to avoid the state explosion problem in verification. It simulates the model several times and statistically verifies whether the verification properties are satisfied based on the accumulated simulation results. Against this background, the properties can be verified through statistical verification to verify the achievement of the common goals.

This study makes the following contributions:

- We propose an architecture and workflow for interactive simulation of SoS that allows users to modify the simulation model and scenario during the simulation to represent the evolutionary development.
- We applied the statistical verification technique to SoS goal verification using the interactive simulation by specifying an SoS goal as a verification property.

The rest of this paper is structured as follows. In Section II, we describe the SMC statistical verification technique.

978-1-7281-8050-2/20/$31.00 ©2020 IEEE

In Section III we review and analyze the SoS simulation and SoS verification. Section IV presents the architecture of the developed simulation-based verification and analysis for system-of-systems (SIMVA-SoS) tool, and explains its simulation and verification. Section V presents a case study of SIMVA-SoS through MCI response SoS scenarios. Section VI, discusses the threat and validity. Finally, Section VII presents the conclusions of this study.

II. Background: Statistical Model Checking

For large and complex systems such as an SoS, exhaustive verification methods such as traditional model checking are inefficient in terms of its verification time and memory [9]. The state explosion problem makes it difficult to apply model checking to a large and complex system. Statistical model checking (SMC) is one of the statistical verification techniques that could be an alternative to traditional model checking [9]. SMC improves the verification efficiency by inferring quantitative verification results of given model's property satisfaction from multiple simulation results of the model. Rather than exploring all possible states of the model, SMC repeats the simulation until a reliable verification result is obtained based on the statistical analysis of the simulation results. In addition, SMC can be applied to black-box or non-deterministic systems if we can acquire the simulation results of the system. SoS models may have non-deterministic behaviors or be based on incomplete or abstracted information because of the characteristics of an SoS; therefore, we can enjoy the benefits of SMC by applying it to SoS verification. Among many SMC algorithms, we use the sequential probability ratio test (SPRT) in SIMVA-SoS, as explained in Section IV.

III. Related Work

A. Simulation of SoS

SoS engineers have utilized simulations to effectively analyze an SoS whose behavior or structure is extremely complex depending on the combination of constituents [10]–[16]. Baldwin *et al.* modeled and simulated interacting constituents' common goal achievement and SoS characteristics including autonomy, belonging, connectivity, and emergence [10]. Zeigler *et al.* [11] and Mittal *et al.* [12] used simulations to observe and predict the emergent behavior of an SoS. In addition, Manzano *et al.* evaluated the dynamic reconfiguration of the SoS architecture through simulation for anticipating the consequences of architecture changes at runtime [13]. Simulations are widely used to see the achievement of the SoS mission [14], [15]. However, an SoS may encounter unpredictable changes in itself or in the environment at runtime. Nonetheless, most existing simulation-based approaches, including our previous work [16], are unable to handle dynamic changes into the SoS because their simulation inputs, such as SoS models and scenarios, are supposed to be fully given before simulation execution. To extend our previous work, in this study, we propose an interactive simulation of an SoS that allows the injection of stimuli in SoS simulation models and scenarios during simulation execution for analyzing the SoS with various changes at runtime.

B. Verification of SoS

Not only observing the SoS simulation but also systematically verifying SoS models are effective ways to verify the properties that the SoS should satisfy [1]. Calinescu *et al.* verified the utility of SoS policies for multi-objective optimization [17]. Michael *et al.* verified the quality of the SoS architecture to meet (non-)functional requirements [18]. Sindiy *et al.* proposed a methodology for SoS verification and applied it to a space exploration SoS, with the decision-making spaces consisting of the verification process and activities [19]. Lim *et al.* [7] and Lomuscio *et al.* [8] verified goal achievement by the cooperation of multiple agents with their own goals and capabilities. Exhaustive verification of an SoS model, such as the model checking that was widely used in the above-mentioned studies, is powerful; however, it suffers from the execution complexity caused by the state explosion problem [9]. Our research group has proposed SMC approaches, known as an efficient alternative to the exhaustive model checking techniques, that use SoS simulation results to minimize the verification cost [2], [3]. In this study, we applied SMC to interactive SoS simulation. Further, we have made the implemented tool available to the SoS research community.

IV. SIMVA-SoS

This section discusses the architecture and structure of SIMVA-SoS[1]. Fig. 1 shows the whole architecture of SIMVA-SoS. SIMVA-SoS can be divided into two modules: SIMVA-SoS simulator and SIMVA-SoS verifier. The SIMVA-SoS simulator has three inputs: SoS model, default scenario, and stimulus. The first two inputs must be defined before running the simulation, and the last input can be defined before or during the simulation. The SIMVA-SoS simulator has a user interface for interactive simulation. Interactive simulation allows users to modify the model or scenario through the injection of a stimulus during the simulation. If the model or scenario is changed through the interactive simulator, these changes are reflected in the simulation. At the end of the simulation, the execution results, that is, the logs of the simulation, are saved in a storage device. After the simulation, the verification process is executed. For the verification, a verification property must be defined in advance. We defined abstract verification property classes for various types of verification properties that can be used in diverse domains. During verification, the statistical verification algorithm runs and the verification result is produced.

A. Interactive Simulator for SoS

To reflect the characteristics, we implement the interactive simulator based on the structure of discrete-time and multi-agent simulation.

A discrete-time simulation structure is used to reflect evolutionary development in the simulator. The interactive simulator

[1]https://github.com/sumin0407/SoS-simulation-engine

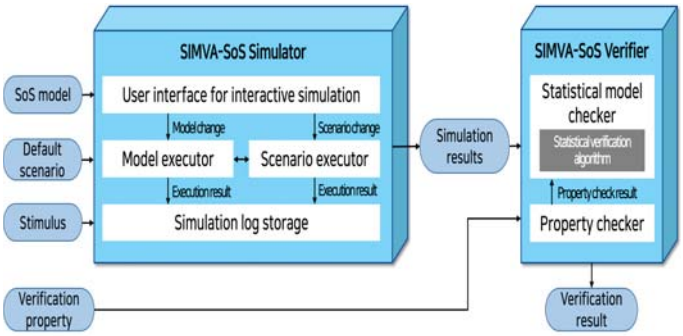

Fig. 1. Architecture of SIMVA-SoS

enables users to inject new scenario events during the simulation. The discrete-time simulation structure, which updates all system states in all discrete times, allows the user to inject new scenario events during the simulation. We applied discrete-time simulation to the SoS simulation, resulting in a structure that updates the state of every CS at every tick.

An SoS contains many individual components with different behavior; therefore, it is better to focus on the activated component of the system [6]. A multi-agent simulation structure is used to represent a complex system consisting of several components with many different kinds of behavior. By focusing on the agents, it reduces the modeling complexity of the system by abstraction. Therefore, a multi-agent simulation structure is appropriate for reflecting characteristics such as managerial independence, operational independence, and emergent behavior. For these reasons, we implement an interactive simulator based on a discrete-time and multi-agent simulation structure. Fig. 2 shows the workflow of the interactive simulator from the user's perspective and the simulation engine's perspective.

In the user's perspective in Fig. 2, the user sets up the simulation scenario, simulation policy, and simulation configuration before running the simulation. After running the simulation, the user can interrupt the simulation at any time during the simulation. When the user interrupts the simulation, the simulation stops while maintaining the current simulation state. Then, the user can inject a new scenario event. The user can resume the simulation after injecting scenario events. When the simulation is resumed, the simulation continues from the point where the simulation was stopped. The user can observe the injected scenario event and progress of the simulation through the GUI. After the simulation is finished, the user can analyze the simulation results.

In the simulation engine's perspective in Fig. 2, the simulation scenario, policy, and configuration are reflected into the simulation. When the simulation runs, scenario events that have been defined by the user are updated in the simulation engine. Further, scenario events that are injected by the user during the simulation are updated in the simulation engine when the user resumes the simulation. After updating the scenario events, the simulation engine increments the simulation time by one tick. Here, a tick is the logical execution time of the simulation. The simulation engine executes scenario events that need to be executed through logic time. For example, suppose we have a scenario event where we add one firefighter

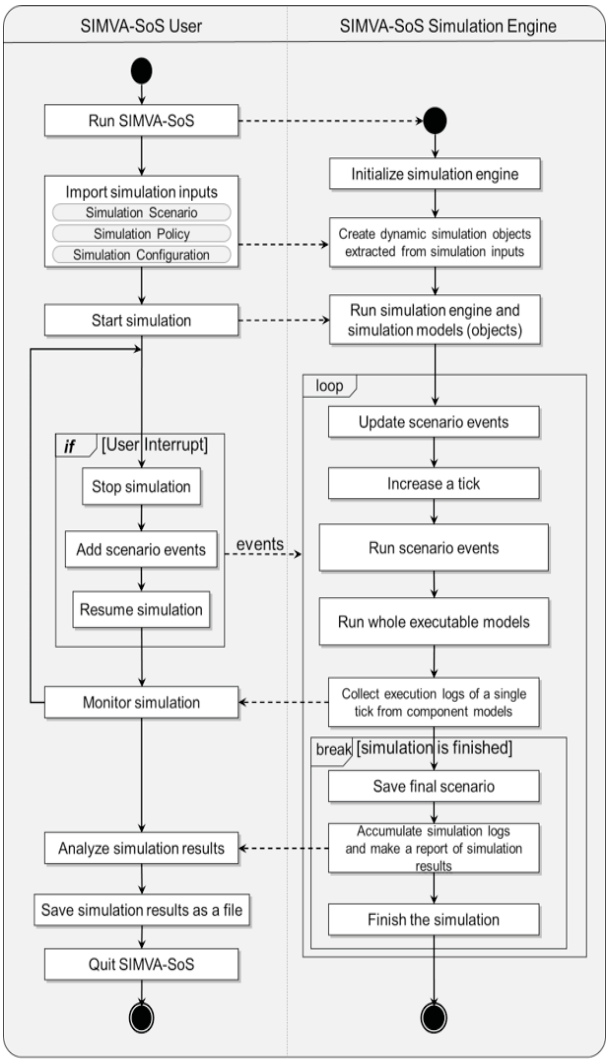

Fig. 2. Workflow of interactive simulation

at 30 ticks. If the logical time of the simulation increases from 29 ticks to 30 ticks, the scenario event is triggered at this point. In other words, one firefighter is added to the simulation. The process of this loop repeats until the simulation is finished. When the simulation is finished, the simulation engine stores all scenario events applied to the simulation. In addition, simulation logs and result reports are generated for the simulation.

B. Statistical Verification for SoS

An SoS has non-deterministic characteristics such as an evolutionary development and emergent behavior that produce non-deterministic simulation results even for the same scenarios. One of the ways to verify non-deterministic systems is using SMC. In SMC, hypothesis testing is used to determine the satisfaction of verification properties. Several statistical verification techniques use hypothesis testing, including Gauss-CI, Chernoff-CI, SPRT, and Azuma.

For the statistical verification of the SoS, the results of repeated simulations are used as sample data, and the probability of satisfying the verification property in the SoS is

TABLE I
TYPES OF VERIFICATION PROPERTY

Verification Properties	Description	Example (MCIR-SoS)
Absence	Globally, it is never the case that P holds with a probability $<$ p	Rescue - Transfer $<$ 0 % should not occur
Universality	Globally, it is always the case that P holds with a probability $\geq$ p	0% $\leq$ Rescue $\leq$ 100% must always be satisfied
Existence	Globally, P holds eventually with a probability $\geq$ p	Rescue $>$ X% should occur
Bounded Existence	Globally, P holds in specific time t with a probability $\geq$ p	CS should be dispatched within 10 ticks
Transient State Probability	Globally, P holds after t ticks with a probability $\geq$ p	Rescue $\geq$ X% for (T-t) time is greater than Y%
Steady State Probability	Globally, P holds in the long run with a probability $\geq$ p	Rescue $\geq$ X% for a long time is greater than Y%
Minimum Duration	Globally, state X holds at least Y duration with a probability $\geq$ p	Activity rate of CS is at least X% for at least Y time
Maximum Duration	Globally, state X holds at most Y duration with a probability $\geq$ p	Rescue = 0% duration for maximum time Y
Recurrence	Globally, state X should occur repeatedly at Y time intervals with a probability $\geq$ p	Repeat the X state within Y hours
Precedence	Globally, state X must occur before state Y with a probability $\geq$ p	Transfer must occur before Rescue
Response	Globally, if state X occurs, it must occur before state Y with a probability $\geq$ p	First aid occurs prior to patient transfer
Until	Globally, state X must be maintained until state Y occurs with a probability $\geq$ p	If all the patients are not rescued, continue the rescue

estimated through statistical analysis. We use SPRT because it is an efficient verification technique based on hypothesis testing [20]. SPRT automatically calculates the number of sample data required for verification under statistical judgement and provides high verification speed by minimizing the number of samples. The SPRT algorithm is shown below in Algorithm 1. The verification property and verification configuration such as type 1 error, type 2 error, and threshold must be given as inputs. The verifier checks the condition with the number of true, number of samples, and verification configuration to run more simulation for statistical verification. If the condition is true, it runs the simulation and store the result (trace). The result is checked through the trace and verification property, and the number of true is incremented by one if the result is true. Then, the number of samples is incremented by one. This process is repeated until the condition is false. Finally, the algorithm presents the verification result.

Algorithm 1 SPRT Algorithm for Verification

```
Input: verification property & verification configuration
while Condition(numOfTrue, numOfSample, config)
    trace = simulation.run()
    if property.check(trace, property) == true then
        numOfTrue + 1
    end if
    numOfSample + 1
end while
result = satisfied(numOfSample, numOfTrue, config)
return result;
```

The SIMVA-SoS verifier supports verification for 12 verification properties. Table 1 provides a detailed description of the verification properties. The probability value p in the table is the result of the statistical verification. For example, if we want to verify an example of the *Existence* property, 'Rescue $\geq$ 50% of patients', the verification result gives the probability p that the property will be met.

V. CASE STUDY

A. Research Question

Some simulations are partially dynamic; however, most simulators do not allow the injection of new stimuli during the simulation. Most simulators that can inject a new stimulus while the simulation is in progress do not provide verification techniques. The purpose of the case study is to verify that interactive simulation and verification are working properly. We consider the following research questions:

- RQ1: Does the interactive simulator allow modifying the scenarios at runtime?
- RQ2: Does the verification function provide proper verification results based on the simulation results?

RQ1 is intended to examine whether the interactive simulator can express the evolutionary development characteristic. Specifically, it checks whether the scenario can be modified at simulation runtime. RQ2 examines whether the verification through simulation results works normally. In the case study, several scenarios are used to identify changes in the verification results.

B. MCIR-SoS Scenario

A Mass Casualty Incident (MCI) is an incident that overwhelms the ability of emergency services to provide appropriate services to patients, due to the number of victims and the severity of their injuries [21]. We follow the MCI Response System-of-Systems (MCIR-SoS) scenario [16] for the case study. In short, the MCIR-SoS scenario consists of four CSs: firefighter, safe zone, ambulance, and hospital. Further, emergency manager gives commands to the CSs. Fig. 3 shows a schematic diagram of MCIR-SoS scenario.

In our previous work *et al.* [16], we defined eight types of stimuli and five types of injection techniques for MCIR-SoS.

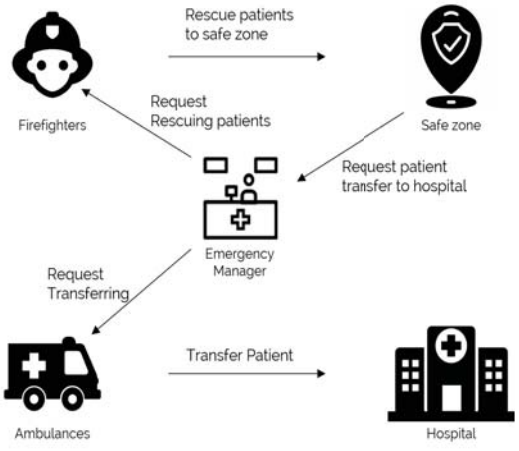

Fig. 3. Schematic diagram of MCIR-SoS scenario

TABLE II
MCIR-SoS Scenarios

Scenario types	Speed	Sight range	Communi-cation	Add CS	Remove CS
Baseline	↓ ↓	↓ ↓	Delay	↑ ↑	
Scenario 1	↓	↓	Normality	↑ ↑ ↑	
Scenario 2	↓ ↓ ↓	↓ ↓ ↓	Loss	↑	↓

The stimuli were defined by analyzing the MCI reports and identified issues. In addition, five types of injection techniques were proposed for dynamic simulation. We use these stimuli and injection techniques for injecting new scenarios at runtime through interactive simulations. In this study, to examine RQ1 and RQ2, three MCIR-SoS scenarios were used. Table II shows how the stimuli were applied to the three scenarios. In scenarios 1 and 2, additional stimuli are applied to the baseline scenario. The larger the number of arrows, the larger the number of stimuli are injected. The configuration such as initial number of CSs and total simulation time of the simulation is the same in all three scenarios.

C. Results and Analysis

1) RQ1: Does the interactive simulator allow modifying the scenarios at runtime?: Evolutionary development, which represents the continuous evolution of the SoS, is one of the most important characteristics of the SoS. To represent the SoS through simulation, the simulation should be able to express evolutionary development. In other words, the user should be able to inject new scenario events at runtime. Five different types of stimuli were injected into the simulation to answer RQ1. All stimuli were applied to firefighters or ambulances. Fig 4 shows the simulation results for three scenarios with various stimuli injected at runtime.

The left-hand side of the figure represents the changes in the number of firefighters and ambulances in each scenario. The line graph shows the number of firefighters and the bar graph shows the number of ambulances. The number of firefighters and ambulances is represented by the left- and right-hand-side y-axes, respectively. After 300 ticks, the number of firefighters and ambulances changed because CS addition and removal stimuli were injected in scenario 1 and 2.

Communication stimuli were applied from 200 to 400 ticks. In the baseline scenario, the communication was delayed at around 75 ticks, and in scenario 2, communication loss occurred. In scenario 1, no communication problem occurred. The right-hand side of the figure shows the number of communications received for each tick; these features are well represented.

Fig 4 indicates that the various stimuli injected at runtime are well applied into the simulation. In conclusion, the user can inject new scenario events at runtime through the interactive simulation, and the interactive simulator was able to express evolutionary development.

2) Does the verification function provide proper verification results based on the simulation results?: After the simulation, SIMVA-SoS start the verification using the same scenarios. SIMVA-SoS use the SPRT algorithm for the statistical verification described in algorithm 1. To verify each scenario, the SPRT runs tens of thousands of simulations. For all three scenarios, the existence property is used as the verification property. The existence property performs a verification on the probability that more than 50% of the patients are rescued during the simulation. Fig 5 shows the verification result using the existence verification property. The probability, represented on the y-axis, is the ratio of *numOfTrue* shown in algorithm 1. The probability can be calculated as *numOfTrue / numOfSample*. Verification with SPRT is calculated using the number of samples, number of true, and threshold value. As a result, it returns true if the value of probability is greater than the value of theta, and false if it is small.

The verification result of the baseline scenario, scenario 1, and scenario 2 shows 54%, 68%, and 32% probabilities of rescuing more than 50% of patients. These results confirm that the probability increases in the order of *scenario 2* < *baseline scenario* < *scenario 1*. We confirmed that the best verification results were obtained in scenario 1 in which the most appropriate stimuli were injected for the rescue. In other words, different verification results are obtained for each scenario, and a good scenario shows a good verification result. In conclusion, statistical verification through simulation results can be seen to provide proper verification results.

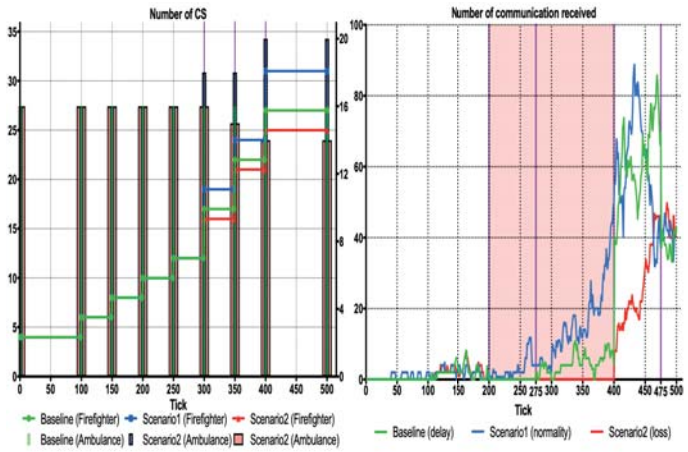

Fig. 4. Result of the interactive simulation

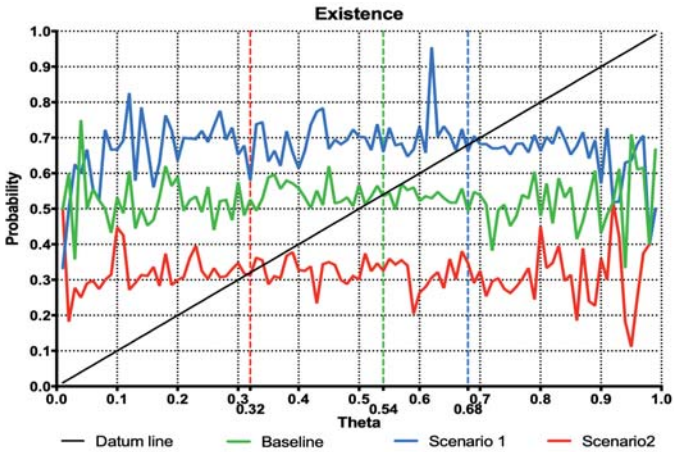

Fig. 5. Result of verification

VI. Threats to Validity

The main threat to validity is the applied scenarios. The MCI scenario we applied extracted information from the actual MCI report; however, it was impossible to obtain all the information from the MCI report. Further, it is not easy to accurately match the logical time of the simulation with the real-time of the real world. However, this study focuses on whether the SoS characteristics can be expressed through simulation and whether the common goal of SoS can be verified through the simulation-based statistical verification. In addition, the case study results confirmed the possibility of a simulation-based verification technique.

VII. Conclusion

This study develops SIMVA-SoS, a tool for the simulation based verification and analysis for system of systems. SIMVA-SoS consists of two parts: simulation and verification. SIMVA-SoS supports interactive simulations that enable injecting dynamic changes into the SoS. This enables modifying the SoS model and scenario at runtime. In addition, SIMVA-SoS has an SMC engine for the efficient verification of a non-deterministic SoS model. It provides quantitative verification results based on the statistical analysis of simulation results without the state explosion problem. We showed the applicability of our tool to the MCIR-SoS case study. A user could inject a stimulus to the MCIR-SoS during simulation execution to represent unexpected changes and quantitative verification results of the goal satisfaction of the SoS were obtained. SIMVA-SoS is an open source tool; therefore, the SoS engineering community can use it as intended or extend it.

The input model of SIMVA-SoS should be implemented manually based on the SIMVA-SoS abstract classes. In future work, we aim to develop a graphical SoS modeling tool that will support the automatic generation of an SoS simulation model for SIMVA-SoS. The simulator of SIMVA-SoS currently shows evolving through the change in the number of instances. We will improve the SIMVA-SoS simulator to apply various changes such as adding new CS and changing interactions. SIMVA-SoS will be applied to other SoS domains, such as traffic control or energy management SoS.

Acknowledgements

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] 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.

[2] D. Seo, D. Shin, Y.-M. Baek, J. Song, W. Yun, J. Kim, E. Jee, and D.-H. 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)*. IEEE, 2016, pp. 12–18.

[3] J. Song, Y.-M. Baek, M. Jin, E. Jee, and D.-H. Bae, "Sos gap slicer: Slicing sos goal and prism models for change-responsive verification of sos," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 546–551.

[4] D. A. Stuart, M. Brockmeyer, A. K. Mok, and F. Jahanian, "Simulation-verification: Biting at the state explosion problem," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 599–617, 2001.

[5] S. Robinson, *Simulation: the practice of model development and use*. Wiley Chichester, 2004, vol. 50.

[6] A. Drogoul, J. Ferber, and C. Cambier, "Multi-agent simulation as a tool for analysing emergent processes in societies," in *Proceedings of Simulating Societies Symposium*, 1994, pp. 127–142.

[7] Y. J. Lim, G. Hong, D. Shin, E. Jee, and D.-H. Bae, "A runtime verification framework for dynamically adaptive multi-agent systems," in *2016 International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2016, pp. 509–512.

[8] A. Lomuscio, H. Qu, and F. Raimondi, "Mcmas: an open-source model checker for the verification of multi-agent systems," *International Journal on Software Tools for Technology Transfer*, vol. 19, no. 1, pp. 9–30, 2017.

[9] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *International conference on runtime verification*. Springer, 2010, pp. 122–135.

[10] W. C. Baldwin and B. Sauser, "Modeling the characteristics of system of systems," in *2009 IEEE International Conference on System of Systems Engineering (SoSE)*. IEEE, 2009, pp. 1–6.

[11] B. P. Zeigler and A. Muzy, "Some modeling & simulation perspectives on emergence in system-of-systems," 2016.

[12] S. Mittal and J. L. Risco-Martín, "Simulation-based complex adaptive systems," in *Guide to Simulation-Based Disciplines*. Springer, 2017, pp. 127–150.

[13] W. Manzano, V. V. Graciano Neto, and E. Y. Nakagawa, "Dynamic-sos: An approach for the simulation of systems-of-systems dynamic architectures," *The Computer Journal*, 2018.

[14] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, and B. Nuseibeh, "Dragonfly: a tool for simulating self-adaptive drone behaviours," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 107–113.

[15] Z. Shu, Q. Jia, X. Li, and W. Wang, "An ooda loop-based function network modeling and simulation evaluation method for combat system-of-systems," in *Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems*. Springer, 2016, pp. 393–402.

[16] S. Park, Z. M. Belay, and D.-H. Bae, "A simulation-based behavior analysis for mci response system of systems," in *2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES)*. IEEE, 2019, pp. 2–9.

[17] R. Calinescu and M. Kwiatkowska, "Software engineering techniques for the development of systems of systems," in *Monterey Workshop*. Springer, 2008, pp. 59–82.

[18] J. B. Michael, R. Riehle, and M.-T. Shing, "The verification and validation of software architecture for systems of systems," in *2009 IEEE International Conference on System of Systems Engineering (SoSE)*. IEEE, 2009, pp. 1–6.

[19] O. Sindiy, D. DeLaurentis, S. Perl, and R. Rovekamp Jr, "Verification and validation of system-of-systems models for lunar command, control, communication, and information architecture," in *AIAA SPACE 2009 Conference & Exposition*, 2009, p. 6557.

[20] D. Reijsbergen, P.-T. de Boer, W. Scheinhardt, and B. Haverkort, "On hypothesis testing for statistical model checking," *International journal on software tools for technology transfer*, vol. 17, no. 4, pp. 377–395, 2015.

[21] "Mass Casualty Incidents," Rogers Fire Department Standard Operating Procedures, https://rogersar.gov/DocumentCenter/View/2505/SOP-510—Mass-Casaulty-Incidents-PDF, 2013.