Model-Based Engineering & Simulation of Software-Intensive Systems-of-Systems

Experience Report and Lessons Learned

Valdemar Vicente Graciano Neto
Universidade Federal de Goiás
Goiânia, Goiás, Brazil
valdemarneto@inf.ufg.br

Wallace Manzano
Universidade de São Paulo
São Carlos, Brazil
wallace.manzano@usp.br

Mohammad Kassab
Pennsylvania State University
Malvern, United States
muk36@psu.edu

Elisa Yumi Nakagawa
ICMC, University of São Paulo
São Carlos, Brazil
elisa@icmc.usp.br

ABSTRACT

Software has been increasingly embedded into systems (e.g., autonomous cars, traffic control systems, power distribution systems) to increase the precision of their functionalities, deliver automation, and make them smarter. Those systems have been combined and formed Systems-of-Systems (SoS) to realize futuristic software applications, such as smart cities. However, the complexity exhibited by them has also increased, claiming for techniques to support their engineering and quality. In this direction, modeling and simulation (M&S) have been established as valuable resources to deal with such complexity, potentially supporting an accurate prediction of SoS software correctness. The main contribution of this paper is reporting the results and advances achieved in the last five years of research on the use of M&S techniques to support Software Engineering of SoS. We report our experience, present learned lessons, and point for important challenges that must still be addressed.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; **Simulator / interpreter**; **Model-driven software engineering**;

KEYWORDS

Model, Modeling, Model-Based, Model-Driven, Simulation, Systems-of-Systems, Lessons Learned, Experience Report.

ACM Reference Format:
Valdemar Vicente Graciano Neto, Wallace Manzano, Mohammad Kassab, and Elisa Yumi Nakagawa. 2018. Model-Based Engineering & Simulation of Software-Intensive Systems-of-Systems: Experience Report and Lessons Learned. In *12th European Conference on Software Architecture: Companion Proceedings (ECSA '18), September 24–28, 2018, Madrid, Spain.* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3241403.3241432

ECSA '18, September 24–28, 2018, Madrid, Spain

ACM ISBN 978-1-4503-6483-6/18/09…$15.00
https://doi.org/10.1145/3241403.3241432

1 INTRODUCTION

Systems-of-Systems (SoS[1]) are complex software-intensive systems constructed as a combination of constituent systems to create innovative and purposeful solutions for a particular domain. SoS are intended to support critical domains, such as emergency and crisis response management systems, smart traffic control, and smart grids. As such, simple failures can cause serious damage and losses. Owing to such criticality, SoS should be analyzed, as early as possible in the development process about the precision of their operation. Moreover, SoS should also be analyzed under a complementary perspective of structure and behavior, enabling an early identification and correction of potential failures.

When we started the research on Software Engineering for SoS, there was an expressive lack of methods and techniques to deal with the such requirements. Our first insight was that models could help on such endeavor. Since a *model* is a selective, reduced, and (hopefully) accurate representation of a system that concisely captures the essential properties for a given set of concerns [52], we could use models to specify and evaluate SoS still at design-time. However, the pure adoption of static models could hamper SoS analysis, as constituents exhibit operational and managerial independence, which leads to a dynamic architecture [47]. Such characteristics required the adoption of a technique to deal with the inherent SoS dynamics. Then, we searched for dynamic/runtime models. However, only simulation models were able to cope with such requirements, despite suffering of a lack of precision in representation of SoS software architectures. Indeed, at that moment, we were not aware of any tool or method that supported, at the same time, both static and dynamic representation of SoS architectures. Hence, we envisioned the gap and advanced on it, achieving results to harmonize two complementary formalisms that separately dealt with the representation of SoS static and dynamic concerns.

The main contribution of this paper is providing an experience report on a robust set of solutions provided for software engineering of SoS, focusing on model-based engineering and simulation approaches for evaluating the accuracy of SoS software architectures. On top of such experience, we also report lessons learned

[1]Herein, the acronym SoS will be interchangeably used to express both singular and plural forms.

from it, and externalize open issues and challenges that must still be addressed in the forthcoming years.

This paper is organized as follows: Section 2 brings the necessary background to understand the challenges we faced and solved. Section 3 contains our experience report. Section 4 presents the lessons we learned from that experience. Section 5 details the challenges and gaps that still must be bridged. Section 6 discusses related works; and finally, Section 7 presents concluding remarks.

2 FOUNDATIONS

SoS are concerned to accomplish missions, which consist of high-level goals that can be splitted in smaller parts to be assigned and solved by the constituents [53]. Constituents in a SoS communicate through mediators, which are connectors that enable the establishment of communication links between two systems, such as an enterprise bus, or a wi-fi hub [59]. Moreover, a SoS is surrounded by an environment, which consists of the real world that provides stimuli to feed the SoS operation.

Software has played an important role on SoS engineering, which motivated research efforts to establish strategies, techniques, and methods to deal with the classic concerns of software engineering regarding SoS [7, 41]. Since SoS is software-intensive, it exhibits a software architecture, which comprises the SoS in its fundamental structure, including its constituents, connections between them, and properties of the constituents and of the environment [44]. Due to the essential characteristics of operational and managerial independence of constituents [36], SoS software architectures are inherently dynamic, since constituents can freely join or leave the SoS structure at any moment [45]. Such ability can be considered even an essential advantage, as it possibly minimizes system disruptions while new or modified constituents are joined into a SoS to substitute the failing ones. However, such characteristic also increases the level of uncertainty about SoS operation, as the absence of a pivotal constituents could cause a system crash.

Models can potentially reduce uncertainty in software engineering through Model-Based Software Engineering (MBSE) [4, 14–16, 39, 51]. By adopting a suitable specification language, models can capture systems structure and behavior. Many languages have been used for specifying SoS architectures [12, 34, 38]. However, general language purposes (such as UML, SysML, and others) have lacked essential constructs to represent all the relevant aspects of a SoS [29]. On the other hand, simulations have represented SoS dynamics [5, 11, 37]. Simulations correspond to an imitation of the operation of a real-world process or system over time, and enable the observation of the effects and outcome of a given set of stimuli against a system to draw inferences about the operation of real-world systems that they represent [3, 56, 61]. Simulations enable such observation because they are often accompanied by animations provided by underlying platforms and tools, which give quick visual feedback to novice modelers and can thus help them identify improper use of modeling constructs [16].

Modeling & Simulation (M&S) is the combination of both techniques. It comprises a paradigm that associates model-based and simulation approaches to capture both SoS structure and behavior, enabling though a prediction, at design-time, of the impact of SoS dynamic architectures on the provided functionalities. On one hand, models can precisely capture the SoS structure. Analogously, simulation models can represent SoS dynamics and enable the visualization of SoS behaviors [2, 8, 13, 16, 28, 56]. As such, M&S promote: (i) a visual and dynamic viewpoint for SoS software architectures, reproducing stimuli the system can receive from a real environment; (ii) prediction of errors, diagnosing them and enabling corrections, and (iii) observation of expected and unexpected emergent behaviors of an SoS [5, 50]. Since SoS are required to exhibit precision and accuracy in their operation, they must be carefully assessed, still at design-time, about their ability to sustain the operation despite their inherent dynamic architectures. Next section details how we cohabited both modeling and simulation in our solutions.

3 EXPERIENCE REPORT

This section reports the advances achieved during the project conduction. We adopted a chronological and sequential order to report the progress of our research, as follows.

State of the art externalization. To support a robust analysis and prediction of SoS operation at design-time, we searched for one or more specification languages that could capture both SoS structure and behavior. Over the past five years, we have conducted literature reviews to acquire a panorama about the *status quo* of available formalisms and notations to support SoS specification [22, 35, 38]. Results revealed that several languages have been adopted in MBSE solutions for SoS engineering. However, none of them were able to cope with the aforementioned requirements, in particular to support the prediction of dynamic architectures impact on the SoS runtime execution. Then, SoSADL emerged. SosADL was an architectural description language (ADL) to bridge the aforementioned gap [45–47]. It was formally founded on π-calculus, while still offering a syntax for specification in high-level of abstraction [47]. In SoSADL, architectures were abstractly defined by means of coalitions, which represent temporary alliances among constituents that collaborate via mediators [47]. Such coalitions can be dynamically formed and changed at runtime. Coalition behaviors document how constituents interoperate to accomplish a given set of missions, whilst mediators represented communication links between two or more constituents [59]. After, we needed a simulation language.

Considering the necessity of representing and simulating SoS dynamics, we surveyed the literature to find some notation that could be appropriate to deal with SoS peculiarities, specially supporting dynamic reconfiguration due to operational independence, multiple and heterogeneous constituents, support to environmental modeling and simulation, representation of dynamics of SoS with an specification of the interoperability between constituents, that is, the data exchanged among them, and a type of time representation to be based upon, we opted by DEVS as the formalism to simulate SoS software architectures [61]. Other simulation formalisms such as Simulink/MATLAB[2], Executable UML [31], or SysML[3] do not support a native representation of SoS with multiple constituents, demanding adaptations and costly implementations. In turn, DEVS

[2] https://www.mathworks.com/products/simulink.html
[3] http://www.omgsysml.org/

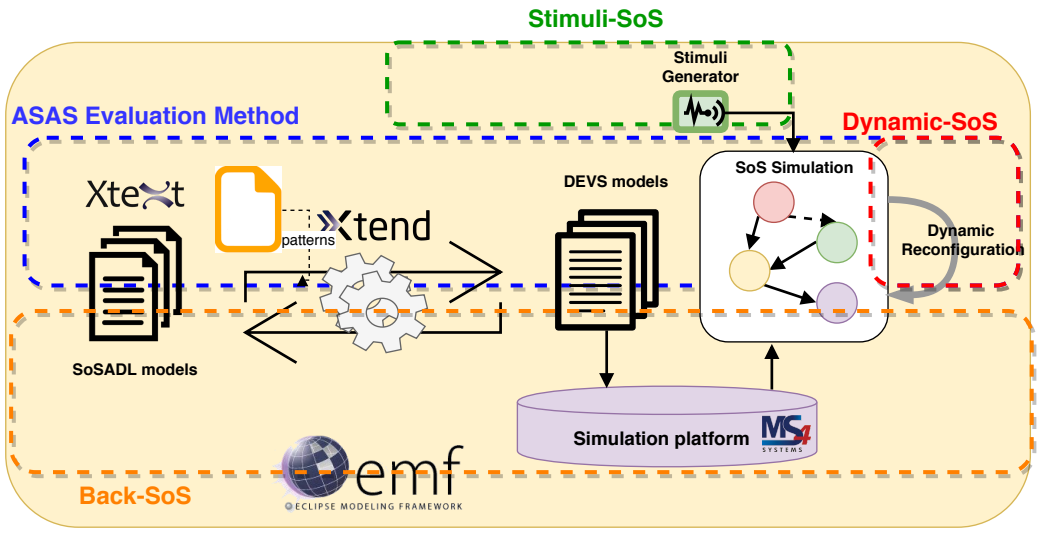

Figure 1: Contributions delivered from our experience [20].

was developed specially to represent SoS. Thus, we chose this language.

DEVS was a modeling formalism for SoS based on the idea of atomic and coupled models [40, 54, 56, 61]. Atomic models represents individual entities in the SoS (for instance, systems), while coupled models represent a combination of atomic models. A variant of DEVS called FD-DEVS and delivered as a natural-language version of DEVS called DEVSNL was adopted. It is supported by a Java library that automatically converts all the DEVSNL code into equivalent Java classes to support simulation execution inside an Eclipse-like modeling Environment named MS4Me[4].

The core of the project was the SosADL2DEVS model transformation, represented in the Figure 1 as a left to right arrow, upon which all advances were built. We detail it, as follows.

SoSADL2DEVS model transformation. Then we decided to combine those two formalisms to achieve the required characteristics of our intended solution. SosADL was the language chosen to describe SoS software architectures [45, 47]. In turn, DEVS formalism was chosen as the modeling language, since it enables to simulate SoS architectures, including the inherent dynamic reconfiguration [61], besides anticipating the architecture response in regards to the inherent forthcoming changes that take place at runtime. We combined both formalisms using a model transformation, i.e., a mapping between the considered models that establishes a traceability link between them, enabling an automatic conversion of one formalism into another one. The establishment of those traceability links consisted of mapping all the concepts of SoSADL language into the concepts of DEVS language. This was not trivial, as SosADL provides a high-level abstraction notation that enables to specify a SoS software architecture in a format that is close to third language generation programming languages, such as Java and Python. In turn, DEVS is a more low-level abstraction language that enables the specification of the individual constituents ports and data types, whilst constituents behaviors are specified as input-output state machines, which events are triggered as data are received or sent out.

Simulation patterns. The mapping between SoSADL and DEVS was quite straightforward when considering constituents systems to atomic models, and SoS architecture to coupled models. However, between the specificities, the automatic description of constituents behaviors was specially challenging, since this consisted of establishing a conversion of constituents behaviors specification into an equivalent state machine. The difficult part was the fact that simulations often rely on dozens of lines of code and some of the simulation specification instructions were conflicting. Besides, the number of lines of code could reach huge dimensions, the systems engineering and DEVS guides did not recommend how we could group simulation instructions to describe constituent behaviors in a non-conflicting way. Then, contributing to the state of the art on software engineering techniques for SoS, we identified, established, and externalized patterns to automate the conception of labeled state machines to represent constituents behaviors in DEVS language [25]. We argue that the provided solutions are really patterns, as they comprise a triple ***{context, recurrent problem, standardized and adapted solution}*** [57]. The context was *Engineering SoS simulations*, the problem was *simulations formalisms offer conflicting instructions to describe constituents behaviors*, and our solution was *grouping non-conflicting instructions to express canonical constructs of constituents behaviors, grounded on input and output events*. A pattern is a reusable abstraction of a solution applicable to a given domain or paradigm. For example, the Factory pattern can be used in any object-oriented language (Java, C ++, etc.). Therefore, the solution we provided is also a pattern, as it can be reused for to automatically generate simulation code to specify constituents behaviors using by DEVS-based formalisms, tools, and platforms, not only that one implemented by MS4Me[5] Other instances include [18] PythonPDEVS [55]; CD++ [58] and

[4]http://ms4systems.com/pages/main.php

[5]Tendeloo and Vangheluwe provide a comprehensive survey of DEVS simulation dialects and tools existent in literature [54].

Virtual Laboratory Environment (VLE) [49], both implemented in C++; DEVS-Ruby [17]; and James II [30], also in Java.

Figure 1 summarizes the contributions built on top of the forward model transformation (SosADL2DEVS)[6], and each part of the contribution is described, as follows.

Stimuli-SoS: Environment representation and stimuli generators. Despite the success of our model transformation to obtain functional and consistent simulation models, a straightforward generation of DEVS code did not guarantee the simulation was executable. This happens because the SoS operation is deeply related to the stimuli received from the environment that triggers the simulation execution. Hence, it was necessary to elaborate a specific entity in the simulation model that could be responsible for delivering the expected stimuli to trigger and drive the SoS operation of the SoS. This need met a current and still actual need for systems engineering, so-called the surrounding *environment representation* problem. The environment comprises the SoS surroundings, such as temperature, wind, water level, and noise; and and/or conditions in which a system operates, such as battery level and geographic position [27]. Environment is local to each system. By the nature of SoS, environments are only partially known at design-time [45]. We then created Stimuli-SoS [27], a solution to automatically produce a set of stimuli generators, i.e., artificial entities to continuously deliver data to feed the SoS simulation and imitate the SoS surrounding environment, automating the stimuli input [1, 48, 60].

Dynamic-SoS: dealing with SoS dynamic architectures. We also implemented in the model transformation the automatic generation of another artificial entity for the simulation: a dynamic reconfiguration controller (DRC). It is linked to all the constituents and mediators in the SoS architecture and can be used to apply the four canonical operations we can perform on a SoS architecture to change it: addition of a new constituent, removal of a constituent, substitution, and reorganization of the entire architecture. The DRC is also automatically generated by the model transformation, and deployed together with the simulation models.

ASAS evaluation method [21]. On top of such SosADL2DEVS model transformation and automatic generation of stimuli generators, we established an evaluation method based on simulations so-named ASAS (A Model-Based Approach to Simulate and Evaluate Software Architectures of Systems-of-systems). We established a method composed of a well-defined set of steps to support the evaluation of SoS architecture behaviors based on a set of pre-defined attributes.

Cases: FMSoS and Space SoS [23, 25]. All the contributions were evaluated using one or two scenarios in case studies. One scenario was a Flood Monitoring SoS (FMSoS) concerned to monitor threat of floods in a urban area. FMSoS was composed of smart sensors, drones, and crowdsourcing systems exchanging information to draw conclusions about possible floods and trigger alarms through gateways. The second scenario was the Brazilian Space SoS, a complex and large-scale system that involves a constellation of six satellites, more than three hundred data collection platforms spread around the Brazilian territories, and ground stations. All these systems were coordinated to deliver important behaviors, in particular, environmental data collection and distribution for meteorology, national security, and environmental condition monitoring, including Amazon rain forest region.

Back-SoS: Architectural Drift and Degradation in SoS [24]. During the conduction of our evaluation studies, as the SoS simulation proceeded, the SoS architectural configuration became significantly different from the preliminary structure specified in SosADL. Then we realized that SoS could suffer from a phenomenon known as architectural degradation, i.e., an increasing inconsistency between the real architecture and the intended one. Then, we established Back-SoS, a reverse engineering approach to recover a SosADL model that could represent the current state of SoS that dramatically changed as simulation progressed due to dynamic reconfiguration.

Systems-of-Information Systems (SoIS) [23, 26]. During the conduction of the Space SoS study, some insights were materialized in findings. We realized that when software-intensive information systems (IS) are involved in a SoS, the SoS missions acquire a strong business nature. As a consequence, such particularities arise a distinct class of SoS, the so-called Systems-of-Information Systems (SoIS). Their missions often involve multiple organizations that own one or more of the constituents involved in the SoIS. Moreover, a clear sequence and interdependency emerge between the parts of the missions individually solved by the constituents. Hence, the mission is structured as an inter-organizational and also flexible (due to SoIS inherent dynamic architecture) business process that must be executed by the entire SoIS. This hypothesis was confirmed during the Space SoS specification and model simulation.

Next section discusses the lessons we learned from such experience.

4 LESSONS LEARNED

The projects conducted earned a set of important learned lessons that we wish to share with the community, as follows.

Co-existence of missions in a SoS. The co-existence of missions must be accordingly planned. A conjecture that we extract (with potential for generalization) is *Mediators could hold the logics to trigger a mission accomplishment* for Flood Monitoring and Space SoS. For the former case, a mediator emerges and enables data transmission in the SoS software architecture when the distance between two sensors is shorter than or equal to 50 meters. For the latter, the approximation of the satellite to the DCP station is examined by the mediators, and when it is shorter than or equal to a specific value, the data transmission is enabled. Hence, in both cases, mediators are pivotal elements for the control of data transmission and triggering of a mission. Besides, the co-existence of missions in a same SoS leads to resource sharing and scheduling. Hence, if a resource (constituent) in a SoS software architecture is part of more than one mission accomplishment, both the scheduling and the context switching must be implemented in the shared resource. Behavior in satellites followed this rule, as they were specified to enable the alternation between missions.

[6] A portfolio of the projects conducted during the last years is available at https://goo.gl/61upD9

Missions Types. We distinguished two types of missions in SoS applications: a data forwarding mission, and business process-oriented mission. The former type is hold by DCP stations in Space SoS, and sensors in Flood Monitoring SoS. In neither cases, constituents were aware of the destination of the data collected and forwarded. On the other hand, we perceive that the data requisition made by the Data Center is business process oriented, i.e., the activities being accomplished are interdependent and there is (i) an order in which they are executed, (ii) a systematic separation of responsibilities, (iii) many roles being played, and (iv) many institutions involved. Moreover, the mission show a request-response nature. As such, we conjecture that those *two types of missions may co-exist* in the Space SoS specified; and *SoS domain comprehends, at least, two types of missions, namely data forwarding and business-process oriented.*. We believe another SoS probably exhibits one, both or more types of missions may co-exist in their architecture, sharing and competing for resources. Strategies must be established for resources scheduling.

Establishment of mapping between architectural and simulation models. This type of transformation for SoS context is quite common [20], since it provides a corresponding animation for a SoS architecture model, making it executable. However, despite such mapping being quite straightforward, some conceptual adaptations were needed. In DEVS, every constituent system is represented in a SoS architecture using the same type of entity: atomic models. Hence, unless the name of the constituent can be used to differentiate its type, precision in architectural representation is lost when a SoS software architecture is modeled (or transformed into) a DEVS simulation model. Then, preserving both models is quite important. SosADL supports modeling of abstract architectures, i.e., a modeling of the *potential types of constituents*, and their *potential links* that can be established among them. However, DEVS does not support abstract architectures. Moreover, It is not possible that a novel *type of constituent*, not predicted at design-time, join the SoS at simulation runtime execution. In SoSADL, this is possible, as long as the constituent matches the restrictions imposed to join the SoS.

Moreover, **environment modeling** is supported in SosADL using a keyword named `environment`, whilst DEVS does not have any specific type of its inherent syntax that represents the surrounding environment, which is an important aspect of any software architecture, including software architectures of SoS [33]. Hence, we needed an adaptation of an atomic model to represent such stimuli generator that materializes the environment representation. Equivalently, **there is no differentiation between the concepts of mediators, or constituents, or stimuli generators**. In DEVS, every major entity of an architecture is an atomic model. Thus, the software architecture is not entirely characterized.

Then, the lesson is "a mapping between architectural and simulation models can lead to semantic sacrifices due to constructs that adapted due to the lack of constructs in the simulation formalism to precisely represent software architecture concerns". However, such limitations are compensated and alleviated by the synergy between SoS both structure and behavior, which are captured by the model transformation and maintained by the combination of both models.

5 OPEN ISSUES AND CHALLENGES

Despite the reported advances, challenges still remain. We discuss below some important gaps that shall still be bridged.

Empirical value of simulations and co-simulation for SoS. Simulation is an imitation of real world processes to predict phenomena on a system, preventing potential malfunction diagnosed at design-time in order to deploy a software without those potential defects [3]. As such, simulations offer a platform to evaluate a SoS software architecture according to a pre-defined set of parameters. However, how to guarantee that the simulation model is correct? França et al. have proposed guidelines to use simulations as source of evidence for empirical studies [10]. However, how to guarantee that a simulation model really reproduces the conditions to which the SoS will be subjected to after deployed? Industrial co-simulation and distributed simulations could be investigated to faithfully reproduce real-world conditions [6]. Moreover, if a simulation is correctly deployed and executable, it means that the model transformation was well-succeeded to create a correspondent and functional operational model possibly equivalent to the specification of the SoS software architecture. And, if such transformation is correct, then it is possible to analyze the SoS software architecture behavior according to what was specified (and this is already an important achievement). However, how to guarantee that the deployed SoS will be identical to that being simulated? These are other problems to be solved, and mechanisms shall be established to guarantee that the findings and conclusions drawn from the simulation models are reliable.

Coverage of simulation models. A simulation represents a system and runs during a limited period of time. How to guarantee that all the possible values it could receive were delivered? Or that all the possible combinations of inputs were tested or that all the possible operational states were checked? Testing techniques for SoS have been investigated [42]. However, simulation-based SoS testing must be advanced to guarantee that a simulation is exhaustive enough to provide the broadest coverage of the space of the operational states the system could assume at runtime. This is a really hard task, since SoS involve multiple independent systems. When considering testing single systems, the number of operational states a software system can assume is already huge. When we multiply it by the number of constituents in a SoS, the uncertainty effect of dynamic architecture, and the number of possible links that can be established between each pair of constituents, this complexity dramatically increase, leading to a combinatorial explosion. Hence, techniques must be created to deal with this new reality.

How to be analytic with simulations? Many types of simulations exist: movement simulation for mobile constituents, agent-based simulations, and systems dynamics. It is possible to observe data exchange, emergent behaviors, an holistic view of the SoS operation, and the feasibility of some coalitions. What else? Which type of conclusions can we safely draw using simulations? This is also related to the feasibility of simulation models, discussed above.

Evaluating quality attributes in SoS software architectures. ASAS was designed to evaluate a SoS architectural from a functional point of view, i.e., ASAS supports evaluation of SoS behaviors. However, how to use simulations to reliably evaluate quality attributes

for SoS software architectures? Each type of quality attribute demands one specific handling. For instance, availability should be examined by the ability of a SoS to deliver its operation and missions accomplishments, despite its inherent dynamics. But how to create an equivalence between simulation processing time and real-world time? Security demands an imitation of invasion situations. Each quality attribute require a particular approach, and this should also be investigated in the forthcoming years.

Simulation platforms scalability. In our studies, we conducted simulations that took more than 30 hours to simulate SoS with a couple hundreds of constituents using powerful processors for that. New York city and São Paulo are intended to be smart cities in a near future. How to deal with the processing charge of simulating smart cities composed by cell phones, autonomous cars, smart buildings, houses, and hospitals, with different granularities, and potentially reaching 50 million constituents, if we consider each one of the 20 million citizens owns one autonomous car and one mobile that can join the smart city SoS? Currently, we do not have computational processing power or simulation techniques to tame such complexity. This should also be investigated.

Architectural degradation for SoS. Changes in SoS are very common. Back-SoS was a methodology created to resynchronize the current state of SoS with the initial model. However, architectures can lead to inconsistent states that can interrupt the SoS operation or even create a degradation of its ability to configure itself. Therefore, advances in architectural degradation in SoS have yet to be made.

6 RELATED WORK

Other valuable studies have been conducted about the adoption of models, model-based techniques, and simulations for SoS. France and Rumpe presented a seminal discussion on the use of model-driven development to deal with complex systems, a class of systems that include SoS [16]. In turn, Nielsen et al. [43] established a parallel between the SoS inherent characteristics and how model-based techniques have been used to cope with them.

Besides, independent literature reviews have communicated the use of M&S techniques for SoS [22, 35, 38]. At the beginning, we identified that the application of model-based methods to SoS engineering was still embryonic and could be further exploited, which motivated us to advance on the contributions reported here. Later, another study corroborated our opinion by concluding that MB approaches were a common systems engineering practice and were adopted in several studies (around 59.38% of included studies in a systematic mapping) for managing systems complexity by enabling engineers to better understand requirements, develop candidate architectures, and verify design decisions early in the development process [35]. Hence, our solutions were quite important, as we dealt with software architecture and engineering concerns in contrast with systems engineering practice, which often deals with SoS under a more holistic perspective.

Graciano Neto provided a comprehensive literature review on the use of M&S techniques for SoS [20], discussing an extensive list of studies that have addressed architecture representation and evaluation of single systems or SoS architectures at systems-level. However, the author emphasizes that those studies have not tackled software concern. Other proposals have also exploited model transformations from a given formalism to DEVS simulation models [9, 19, 32]. However, they have not supported the transformation of SosADL descriptions into DEVS models, or a precise representation of both SoS software structure and behavior in a same approach. Therefore, the advances and lessons reported here, which have been extensively evaluated by international forums and publication vehicles, may be considered genuine advances on software engineering for SoS. Next section brings concluding remarks.

7 FINAL REMARKS

This paper reported results of an experience of five years of research on the application of M&S techniques to software engineering of SoS. We highlight the problems we faced, how we solved them and provided contributions and solutions for the community. Besides, we externalized lessons learned from such experience, and pointed for important gaps and research challenges that still must be bridged. The results reported were achieved as a collaboration between two important research groups on software architecture: ArchWare (IRISA/UBS, France) and SofTware ARchitecture Team (START/ICMC-USP, Brazil). Considering the importance of smart cities to improve life quality in cities in the forthcoming years, research on software engineering for SoS assumes a prominent importance. We hope these results can pave the way for software engineering of SoS research in the next years.

ACKNOWLEDGMENTS

This work was partially supported by Foundation for Research Support of the State of São Paulo – FAPESP (grant number 2017/06195-9). We also thank prof. Dr. Flavio Oquendo for its pioneering on the creation of SosADL language and valuable advising that made possible the achievement of the communicated results.

REFERENCES

[1] Bashir Al-Hashimi. 1995. *The Art of Simulation Using PSpice: Analog and Digital* (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.
[2] W. Clifton Baldwin, Brian Sauser, and Robert Cloutier. 2015. Simulation Approaches for System of Systems: Events-based versus Agent Based Modeling. *Procedia Computer Science* 44, 1 (2015), 363 – 372.
[3] Jerry Banks. 1999. Introduction to Simulation. In *31st WSC*. ACM, Phoenix, Arizona, USA, 7–13.
[4] E. Barbi, G. Cantone, D. Falessi, F. Morciano, M. Rizzuto, V. Sabbatino, and S. Scarrone. 2012. A model-driven approach for configuring and deploying Systems of Systems. In *7th SoSE*. IEEE, Genova, Italy, 214–218.
[5] Jan Bosch. 2000. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. Addison-Wesley, New York, USA.
[6] J. F. Broenink and Y. Ni. 2012. Model-driven robot-software design using integrated models and co-simulation. In *SAMOS*. Samos, Greece, 339–344.
[7] Radu Calinescu and Marta Kwiatkowska. 2010. Software Engineering Techniques for the Development of Systems of Systems. In *Foundations of Computer Software. Future Trends and Techniques for Development*, Christine Choppy and Oleg Sokolsky (Eds.). Lecture Notes in Computer Science, Vol. 6028. Springer Berlin Heidelberg, Berlin, Germany, 59–82.
[8] P. Carle, R. Kervarc, R. Cuisinier, N. Huynh, J. Bedouët, T. Rivière, and E. Noulard. 2012. Simulation of Systems of Systems. *AerospaceLab* 4 (2012), p. 1–10.
[9] Deniz Cetinkaya, Alexander Verbraeck, and Mamadou D. Seck. 2012. Model Transformation from BPMN to DEVS in the MDD4MS Framework. In *8th TMS/DEVS*. SCS, Orlando, USA, Article 28, 6 pages.
[10] Breno Bernard Nicolau de França and Guilherme Horta Travassos. 2016. Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering* 21, 3 (2016), 1302–1345.
[11] Liliana Dobrica and Eila Niemele. 2002. A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering* 28, 7 (July 2002), 638–653.

[12] Elisa Yumi Nakagawa et al. 2017. Software Architecture and Reference Architecture of Software-intensive Systems and Systems-of-systems: Contributions to the State of the Art *(11th ECSA Companion)*. ACM, Canterbury, UK, 4–11.

[13] Katrina Falkner, Claudia Szabo, Vanea Chiprianov, Gavin Puddy, Marianne Rieckmann, Dan Fraser, and Cathlyn Aston. 2016. Model-driven performance prediction of systems of systems. *Software & Systems Modeling* 15, 3 (2016), 1–27.

[14] C. Farcas, E. Farcas, I.H. Krueger, and M. Menarini. 2010. Addressing the Integration Challenge for Avionics and Automotive Systems - From Components to Rich Services. *Proc. IEEE* 98, 4 (2010), 562–583.

[15] N. Fischer and H. Salzwedel. 2011. Overcoming the Generation Gap in aircraft designs with executable specifications. In *31st DASC*. IEEE, Sydney, Australia, 1–10.

[16] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *FOSE 2007*. IEEE, Minneapolis, USA, 37–54.

[17] Romain Franceschini, Paul-Antoine Bisgambiglia, Paul Bisgambiglia, and David Hill. 2014. DEVS-ruby: A Domain Specific Language for DEVS Modeling and Simulation (WIP) *(DEVS '14)*. SCSI, Tampa, Florida, Article 15, 15:1–15:6 pages.

[18] Romain Franceschini, Paul-Antoine Bisgambiglia, Luc Touraille, Paul-Antoine Bisgambiglia, and David R.C. Hill. 2014. A survey of modelling and simulation software frameworks using Discrete Event System Specification. In *ICCSW'14 Imperial College Computing Student Workshop*. London, United Kingdom.

[19] Ariel Gonzalez, Carlos Luna, Marcela Daniele, Roque Cuello, and Marcela Perez. 2015. Towards an automatic model transformation mechanism from UML state machines to DEVS models. *CLEI Electronic Journal* 18, 2 (2015).

[20] Valdemar Vicente Graciano Neto. 2018. *A simulation-driven model-based approach for designing software-intensive systems-of-systems architectures*. Ph.D. Dissertation. ICMC - USP.

[21] Valdemar Vicente Graciano Neto, Lina Garcés, Milena Guessi, Carlos Paes, Wallace Manzano, Flavio Oquendo, and Elisa Yumi Nakagawa. 2018. ASAS: An Approach to Support Simulation of Smart Systems. In *51st HICSS 2018*. IEEE, Big Island, Hawaii, USA, 5777–5786.

[22] Valdemar Vicente Graciano Neto, Milena Guessi, Lucas Bueno R. Oliveira, Flavio Oquendo, and Elisa Yumi Nakagawa. 2014. Investigating the Model-Driven Development for Systems-of-Systems. In *8th ECSA Workshops*. ACM, Vienna, Austria, Article 22, 22:1–22:8 pages.

[23] Valdemar Vicente Graciano Neto, Flavio Horita, Everton Cavalcante, Adair Rohling, Daniel Santos, Jamal El-Hachem, and Elisa Nakagawa. 2018. A Study on Goals Specification for Systems-of-Information Systems: Design Principles and Conceptual Model. In *SBSI 2018*. Caxias do Sul, Brazil.

[24] Valdemar Vicente Graciano Neto, Wallace Manzano, Lina Garcés, Milena Guessi, Brauner Oliveira, Tiago Volpato, and Elisa Yumi Nakagawa. 2018. Back-SoS: Towards a Model-based Approach to Address Architectural Drift in Systems-of-Systems. In *33rd SAC*. ACM, Pau, France, 1461–1463.

[25] Valdemar Vicente Graciano Neto, Wallace Manzano, Adair Rohling, Tiago Volpato, and Elisa Yumi Nakagawa. 2018. Externalizing Patterns for Simulation in Software Engineering of Systems-of-Systems. In *33rd SAC*. ACM, Pau, France, 1687–1694.

[26] Valdemar Vicente Graciano Neto, Flavio Oquendo, and Elisa Yumi Nakagawa. 2017. *Smart Systems-of-Information Systems: Foundations and an Assessment Model for Research Development*. Brazilian Computer Society, Porto Alegre, Brazil, 1–12.

[27] Valdemar Vicente Graciano Neto, Carlos Eduardo Paes, Lina Garcés, Milena Guessi, Flavio Oquendo, and Elisa Yumi Nakagawa. 2017. Stimuli-SoS: A Model-Based Approach to Derive Stimuli Generators in Simulations of Software Architectures of Systems-of-Systems. *Journal of the Brazilian Computer Society* 23, 1 (2017), 13:1–13:22.

[28] Jeff Gray and Bernhard Rumpe. 2016. Models in simulation. *Software & Systems Modeling* 15, 3 (2016), 605–607.

[29] Milena Guessi, Everton Cavalcante, and Lucas B. R. Oliveira. 2015. Characterizing Architecture Description Languages for Software-Intensive Systems-of-Systems. In *3rd SESoS*. IEEE, Florence, Italy, 12–18.

[30] Jan Himmelspach and Adelinde M. Uhrmacher. 2007. Plug'N Simulate *(40th ANSS)*. IEEE, Washington, DC, USA, 137–143.

[31] Jianpeng Hu, Linpeng Huang, Bei Cao, and Xuling Chang. 2014. Executable Modeling Approach to Service Oriented Architecture Using SoaML in Conjunction with Extended DEVSML. In *11th SCC*. IEEE, Anchorage, Alaska, USA, 243–250.

[32] Jianpeng Hu, Linpeng Huang, Bei Cao, and Xuling Chang. 2014. Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable. In *26th SEKE*. Knowledge Systems institution, Vancouver, Canada.

[33] ISO/IEC/IEEE. 2011. ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description. (Dec 2011), 1–46.

[34] John Klein and Hans van Vliet. 2013. A Systematic Review of System-of-systems Architecture Research. In *9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA 2013)*. ACM, Vancouver, Canada, 13–22.

[35] Cristiane A. Lana, Nilton M. Souza, Márcio E. Delamaro, Elisa Y. Nakagawa, Flávio Oquendo, and José C. Maldonado. 2016. Systems-of-Systems Development: Initiatives, Trends, and Challenges. In *42nd CLEI*. IEEE, Valparaiso, Chile, 1–10.

[36] Mark W. Maier. 1998. Architecting principles for systems-of-systems. *Systems Engineering* 1, 4 (1998), 267–284.

[37] J. B. Michael, R. Riehle, and M. T. Shing. 2009. The verification and validation of software architecture for systems of systems. In *4th SoSE*. IEEE, Albuquerque, USA, 1–6.

[38] Milena Guessi et al. 2015. A systematic literature review on the description of software architectures for systems of systems. In *30th SAC*. ACM, Salamanca, Spain, 1433–1440.

[39] S. Mittal and J.L. Risco Martin. 2013. Model-driven systems engineering for netcentric system of systems with DEVS unified process. In *45th Winter Simulation Conference (WSC 2013)*. Society for Modeling and Simulation International, Washington DC, USA, 1140–1151.

[40] Saurabh Mittal, Bernard P. Zeigler, Jose L. Risco Martin, Ferat Sahin, and Mo Jamshidi. 2008. Modeling and Simulation for Systems of Systems Engineering. In *System of Systems Engineering*, Mo Jamshidi (Ed.). John Wiley & Sons, Inc., Washington, USA, 101–149.

[41] Elisa Y. Nakagawa, Marcelo Goncalves, Milena Guessi, Lucas B. R. Oliveira, and Flavio Oquendo. 2013. The State of the Art and Future Perspectives in Systems of Systems Software Architectures. In *1st SESoS*. ACM, Montpellier, France, 13–20.

[42] Vânia Neves, Antonia Bertolino, Guglielmo De Angelis, and Lina Garcés. 2018. Do we need new strategies for testing Systems-of-Systems?. In *6th SESoS at ICSE*. IEEE, Gothenburg, Sweden, 1–8.

[43] C.B. Nielsen, P.G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska. 2015. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *Comput. Surveys* 48, 2 (2015).

[44] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. 2015. Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. *Comput. Surveys* 48, 2, Article 18 (Sept. 2015), 41 pages.

[45] Flavio Oquendo. 2016. Formally Describing the Software Architecture of Systems-of-Systems with SosADL. In *11th SoSE*. IEEE, Kongsberg, Norway, 1–6.

[46] Flavio Oquendo. 2016. π-Calculus for SoS: A Foundation for Formally Describing Software-intensive Systems-of-Systems. In *11th IEEE System of Systems Engineering Conference (SoSE 2016)*. IEEE, Kongsberg, Norway, 1–6.

[47] Flávio Oquendo. 2016. Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems. In *10th ECSA*. Springer, Copenhagen, Denmark, 3–21.

[48] L. Piccolboni and G. Pravadelli. 2014. Simplified stimuli generation for scenario and assertion based verification. In *15th Latin American Test Workshop (LATW 2014)*. IEEE, Fortaleza, Brazil, 1–6.

[49] Gauthier Quesnel, Raphael Duboz, and Eric Ramat. 2009. The Virtual Laboratory Environment - An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory* 17, 4 (2009), 641 – 653.

[50] Daniel Soares Santos, Brauner Oliveira, , Milena Guessi, Flavio Oquendo, Marcio Delamaro, and Elisa Yumi Nakagawa. 2014. Towards the Evaluation of System-of-Systems Software Architectures. In *8th WDES*. SBC, Maceió, Brazil, 53–57.

[51] B. Selic. 2003. The pragmatics of model-driven development. *Software, IEEE* 20, 5 (sept.-oct. 2003), 19 – 25. https://doi.org/10.1109/MS.2003.1231146

[52] Bran Selic. 2012. MDE Basics with a UML Focus. In *Formal Methods for Model-Driven Engineering*, Cortellessa Vittorio Pierantonio Alfonso Bernardo, Marco (Ed.). Springer, Bertinoro, Italy, 1.

[53] E. Silva, E. Cavalcante, and T. Batista. 2017. Refining Missions to Architectures in Software-Intensive Systems-of-Systems. In *IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS 2017)*. IEEE, Buenos Aires, Argentina, 2–8.

[54] Yentl Van Tendeloo and Hans Vangheluwe. 2017. An evaluation of DEVS simulation tools. *Simulation* 93, 2 (2017), 103–121.

[55] Yentl Van Tendeloo and Hans Vangheluwe. 2014. The Modular Architecture of the Python(P)DEVS Simulation Kernel *(DEVS '14)*. SCS International, San Diego, CA, USA, Article 14, 6 pages.

[56] Hans Vangheluwe. 2008. Foundations of modelling and simulation of complex systems. *Electronic Communications of the EASST* 10 (2008).

[57] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. 1995. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley* 49, 120 (1995), 11.

[58] Gabriel Wainer. 2002. CD++: a toolkit to develop DEVS models. *Software - Practice and Experience* 32 (2002), 1261–1306.

[59] G. Wiederhold. 1992. Mediators in the architecture of future information systems. *Computer* 25, 3 (March 1992), 38–49.

[60] S. Yang, R. Wille, D. Grobe, and R. Drechler. 2012. Coverage-Driven Stimuli Generation. In *15th Euromicro Conference on Digital System Design (ECDSD 2012)*. IEEE, Izmir, Turkey, 525–528.

[61] Bernard P. Zeigler, Hessam S. Sarjoughian, Raphal Duboz, and Jean-Christophe Souli. 2012. *Guide to Modeling and Simulation of Systems of Systems*. Springer-Verlag London, London, United Kingdom.