Towards Advances on Software Architecture Design of Constituents for Systems-of-Systems: Enabling Operational Independence

Paulo Gabriel Teixeira[1] **(Master student), Valdemar Vicente Graciano Neto (Supervisor)**[1]

[1] GoIn-Sight: **Go**iás **In**formation **S**ystems and Software eng**i**neerin**g** Researc**h** **T**eam
Mestrado Acadêmico em Ciência da Computação
Programa de Pós-Graduação em Ciência da Computação
Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – CEP 74001-970 – Goiânia – GO – Brazil
Admission: 03/2019 – Qualification: 06/2020 – Expected period of defense: 02/2021

{paulogabriel@inf.ufg.br, valdemarneto@ufg.br}

Abstract. *Over the years, knowledge on how to engineer software-intensive system-of-systems (SoS) has been expanded and advanced. However, challenges remain. Constituent Systems (CSs) are required to instantaneously connect themselves to a SoS while still preserving their operational independence. Moreover, SoS CS is subject to a sort of heterogeneities that makes it difficult to make decisions outside predefined frameworks, environments, and hierarchical command-control structures. Hence, many of the systems currently available are not prepared to be part of an SoS, i.e., they can not maintain their operational independence despite their participation in one or more SoS. Based on this context, the main goal of this research is to exploit how to design software architecture for systems that are intended to become part of a SoS in the future. To achieve this goal, we chose the urban mobility SoS domain and then executed a software architecture design process to design software architecture for an autonomous car as CS of this SoS. Later, we evaluated it through simulation. Preliminary results reveal that our proposal complies with the requirements raised during the architectural design process and can enable a system to be a constituent of a SoS while still preserving its operational independence.*

Keywords: *systems-of-systems, constituent systems, software architecture.*

Events related: *SBCARS, SBES*

We thank CNPq for supporting this research under the grant number 130337/2019-6 (first author). The authors also thank RT Systems Inc. (RTSync), a USA company, for a MS4ME tool free license to conduct our academic studies.

1. Introduction

System-of-Systems (SoS) is an alliance of multiple systems, known as Constituent Systems (CSs), interoperating to achieve a purpose not generally achievable by the individual systems acting independently (Maier 1998). CS is an individual system that interoperate with other individual systems to form a SoS, and have some specific characteristics, including Operational Independence (O.I.). This one comprises the ability of the system to exist outside of SoS, and is related to the ability of a CS to pursue an individual purpose previously established, and it also concerns the ability to maintain its independence and operate for unique purposes, even though it is part of a SoS (Maier 1998). Moreover, the CSs are usually developed separately to accomplish their specific purposes. They can for instance operate independently and are primarily managed to accomplish their separate purposes. A smart city, which is a remarkable example of SoS, is planned to involve CSs that interoperate to (i) leverage the citizen's experience, (ii) manage important concerns (e.g., sustainable power distribution, power billing and economy, and integration of public services such as health and emergency response systems), and (iii) improve city services, including touristic information and city-wide wifi connection (Mendes et al. 2018). Although CSs are part of the smart city SoS, they still preserve their purposes and operation and, consequently, independence.

SoS potentially result from an alliance of new and legacy systems, and these systems are not necessarily designed to form a SoS. These systems are heterogeneous (e.g., in terms of technology and operation), come from multiple domains, are self-contained, have different contexts, behave concurrently, and were independent before being integrated into the SoS (Firesmith 2010). A SoS can consist basically of any type of system, and some have limitations such as hardware, technologies, and processing power. They also have their unique capabilities, problems, supporters, users, budget, schedule, and interface requirements (Carlock and Fenton 2001). Hence, many of these systems are not prepared to maintain their O.I. despite their participation in one or more SoS (Maciel et al. 2017).

According to Maier (Maier 1998), O.I. is a requirement for SoS existence. Other types of complex systems also exist in several domains as healthcare, transportation, energy, and defense, and contexts such as corporations, cities, and government (ISO/IEC/IEEE:15288 2019). However, since they are integrated to interoperate, they use to be exclusively dedicated to that complex system, which narrows the extension and variety of functionalities delivered by that whole system. Moreover, CS should cope with O.I. so that it is capable of acting inside and outside the SoS context; otherwise, motivations for a CS to contribute to a SoS would be lost and all the associated benefits that could take place would not happen, as well. In this context, we need to advance the state of the art on the architectural design of software systems to enable them to regulate their O.I. degree and how it can contribute to the SoS while preserving its independence.

The main objective of this project is to exploit how to design software architecture for systems that are intended to become part of a SoS in the future while specifically satisfying the OI requirement. As a result, we expect to (i) provide derived requirements that need to be addressed to cope with O.I. requirement; (ii) provide a software architecture proposal of a CS that meets the O.I. requirement; and (iii) perform an in-vitro simulation-based experiment to evaluate the software architectural proposal.

The remainder of the paper is organized as follows: Section 2 introduces the foun-

dations and related work. Section 3 details the methodology to conduct the research, the current state of work, and the expected contributions. Section 4 brings the final remarks.

2. Background

A single system is a collection of components organized to accomplish specific purposes. In turn, a *software-intensive system* is a system in which software impacts its entire life cycle, from conception to maintenance and evolution (ISO/IEC/IEEE:42010 2011). SoS has been inherently software-intensive (Boehm 2006). The O.I. was assigned as an essential characteristic to SoS by Maier (Maier 1998). But over the years of SoS research evolution O.I. has also been called as independence (Nielsen et al. 2015) or autonomy (Boardman and Sauser 2006). The O.I. is a major characteristic of CS. It implies that a given CS offers a range of behaviors, some to comply with the needs of SoS and others for its purposes. Besides, the relationship and dependencies between these behaviors and capabilities are not always visible to the SoS engineer (Nielsen et al. 2015).

Also, it is important to note that each CS can have different O.I. degree, i.e., how much the CS is available for the SoS. This degree can be defined by the CS user or by the system itself, and the SoS may or may not have prior knowledge of that degree, depending on the type of SoS (directed, collaborative or virtual) (Nielsen et al. 2015). Moreover, this O.I. degree can vary according to the SoS domain and the mission for which the CS is being requested. For example, in an Urban Mobility SoS (UMSoS), an autonomous car as CS can contribute in several ways, such as: (i) provide data on road conditions and (ii) collaborative rides/car sharing. In the first scenario in which the CS is required to provide data on road conditions, the O.I. degree can indicate (i) the amount of data; or (ii) the number of sensors; or (iii) for how long the autonomous car can lend a certain sensor. This degree cannot impair its functioning and, therefore, does not endanger the user's life. In turn, in a collaborative ride scenario, O.I. can assume how much the user of that car is willing to deliver time or distance for a given ride.

Figure 1 illustrates how the O.I. assumes the value of *how much time the user of the autonomous car CS is willing to deliver to collaborate with SoS and accept rides*. In scenario 1, the autonomous car CS was requested by UMSoS to give a ride to a person who was on its initial route and was going to the same location as the other. Hence, there would be no change in travel time. In scenario 2, for the autonomous car CS meet the ride request, there would be a change in its initial route and consequently an increase in travel time. In this situation, if the user of the CS autonomous car has an appointment and, therefore, cannot be late, he may have defined an O.I. degree less than the necessary to meet the SoS ride request. Therefore, the CS will not be able to contribute to the SoS since it must also fulfill its objective, which is to take its user within the necessary time to the destination. In other words, O.I. degree in the collaborative ride scenario represents the time that the user is willing to allow for an increase in travel time and contribute to the SoS by offering collaborative rides that fit within that established degree.

Moreover, an autonomous car is a software-intensive CS that inherently have a software architecture. According to Bass et al., software architecture is composed of elements, connections, or relations among them and usually some other aspects, such as configuration; surrounding environment, constraints or semantics; analyses or properties; or rationale, requirements, or stakeholders needs (Bass et al. 2003). To design software architecture, an architectural design process is usually used, i.e., a well-defined set of steps

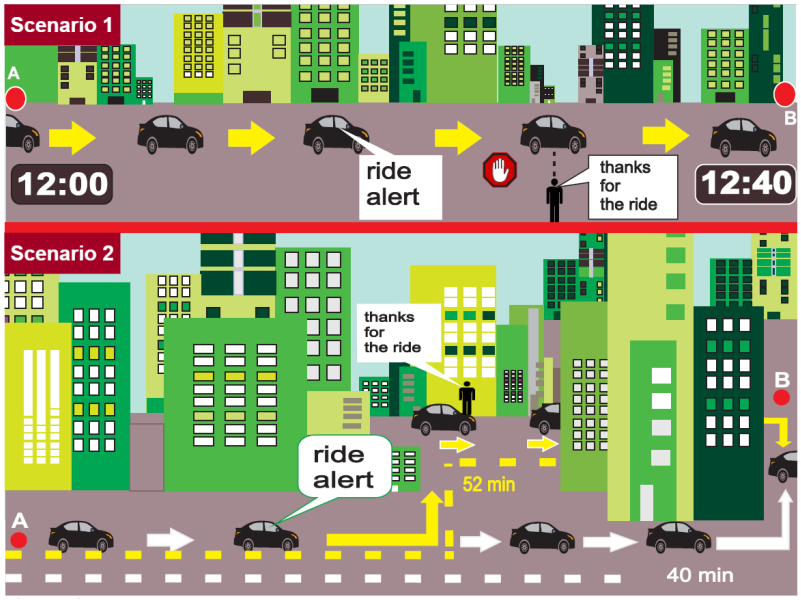

Figure 1. Collaborative ride scenarios.

for the realization of the architectural project. A well-accepted model of the software architecture design process consists of three steps (Hofmeister et al. 2007): (i) determine architectural requirements; (ii) architecture design; and (iii) evaluation.

There are several methods available to evaluate a software architecture (Dobrica and Niemele 2002). One of these methods is the use of simulations, which is the imitation of the operation of a process, or real-world system over time. It involves the generation of an artificial history of the system and the observation of its log to draw inferences about the operational characteristics of the actual system being represented (Banks 1999). There are several types of simulation found in the literature, such as Agent-Based Simulation, Discrete-Event, etc. (França and Travassos 2013). In this research, we will use the Discrete Event System Specification (DEVS) to evaluate our architecture. DEVS is a formalism for simulating a discrete event system based on a hierarchy of atomic, coupled, and hierarchical models, which represent different levels of complexity (Zeigler et al. 2013).

Related Work. A recent systematic mapping showed that there are several studies on SoS software architectures (Cadavid et al. 2020). Scarce studies deal specifically with CS software architecture design, and there are not abundant studies directly related to the CS and their needs. Pelliccione et al. (Pelliccione and et al. 2016) focus on how to architect a car as a CS of a future transportation system. However, the main contribution is the definition of a specific viewpoint of the Volvo Cars architecture framework and does not specifically address the requirement for operational independence. Axelsson (Axelsson 2019) proposes a description of certain common substructures of the SoS, and the corresponding states within a CS. However, the author's approach focuses on the SoS substructure and the states of the CS, but it does not consider the substructure of CS. Salado (Salado 2015; Salado 2016) has presented the concept of abandonment and exile. In particular, abandonment has been defined as the capability of a system within a SoS

to freely decide to leave the SoS. Exile has been defined as the capability of a SoS to voluntarily expel one of its CS. However, despite presenting important aspects directly related to the constituents, no architectural proposals are presented. Yokell (Yokell 2018) assesses the operational relationships between the CS within a SoS, and the managerial relationships between the organizations that own them. Although the study is directly related to the CS of a SoS, no concepts related to the software architecture for CS are presented. Next section details the methodology and preliminary results.

3. Scientific Methodology and Preliminary Results

We present below the steps planned for this research. The steps are aligned with Neto and Travassos guidelines (Dias-Neto et al. 2010). Figure 2 presents the status of the research according to the established steps: as clearer as more concluded that step.

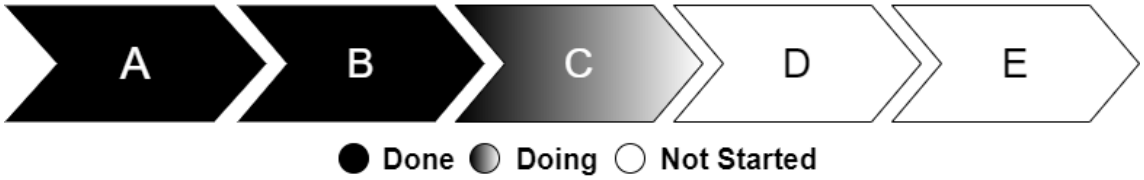

Figure 2. Current State of Work.

Ad-hoc Literature Review (A): This step aims to obtain the main concepts of an area, in a non-systematic approach. Fundamental and also more recent studies of the SoS domain were analyzed to identify (i) the gaps in the literature, (ii) CS characteristics and needs, and (iii) the background to support this project.

Systematic Mapping (B): This step consisted of an evidence-based procedure that aims to identify scientific evidence in an area. Analyzing the studies addressing software architecture in SoS domain (there are already secondary studies addressing this context in SoS (Guessi et al. 2015; Cadavid et al. 2020)), it was possible to notice that most of the studies are focused on SoS as a whole and not on the individual constituents.

Evaluation of collected evidence (C): This step aims at evaluating the identified evidence with the research community. Based on the literature findings, we developed an initial proposal for software architecture for SoS constituents. This proposal was submitted to a relevant vehicle in the area and it was well accepted (Teixeira et al. 2020).

Extended Software Architecture (D): This step aims to extend our initial proposal for a CS software architecture. We have verified some possibilities for improvement in our initial software architecture proposal, and it is still at a higher level. We want to cover views, quality attributes, and architectural styles that a CS may need. To design this software architecture, the process developed by Hofmeister (Hofmeister et al. 2007) will be used together with the reference architecture for SoS constituent systems that are being developed by another member of the same research group (GOInsight) (Batista and Graciano Neto 2020) to which this master's project belongs.

Software Architecture Evaluation (E): The software architecture developed in step D will be evaluated using an experiment based simulation in Discrete Event System Specification (DEVS). We will use the Software Architecture Evaluation Model (SAEM) proposed by Bogado et al. (Bogado et al. 2017) and the guidelines prescribed by de França

et al. (de França and Travassos 2016). Next, we will also prepare a survey (questionnaire with experts) to evaluate the results obtained.

Expected Contribution. The main contribution is the architectural design for a CS to meet one of the main requirements for a system to be a CS of a SoS: operational independence. Other contributions include (i) requirements a SoS CS should cope with; (ii) documentation for a CS software architecture and its views; and (iii) evaluation of CS architecture based on DEVS simulation. We hope that from the conception of this architecture and the reported results, it will be possible to advance the state of the art of software architectures for SoS constituents, leading to other domains also being explored.

4. Final Remarks

This paper presented the results of an ongoing master's research project. We expect this research contributes to the SoS state of the art, especially towards the requirement of O.I. for CSs. This master's research project aims to contribute by (i) providing derived requirements that need to be addressed to cope with O.I. requirement; (ii) providing a software architecture proposal of a CS that meets the O.I. requirement; and (iii) performing an *in-vitro* simulation-based experiment to evaluate the software architecture conceived.

References

[Axelsson 2019] Axelsson, J. (2019). A refined terminology on system-of-systems substructure and constituent system states. In *SoSE*, pages 31–36, Anchorage, Alaska, USA. IEEE.

[Banks 1999] Banks, J. (1999). Introduction to simulation. In *WSC'99*, volume 1, pages 7–13, Phoenix, AZ, USA. IEEE.

[Bass et al. 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

[Batista and Graciano Neto 2020] Batista, P. and Graciano Neto, V. V. (2020). Rumo a uma arquitetura de referência para constituintes de sistemas de sistemas de informação. *WTDSI*, pages 1–6.

[Boardman and Sauser 2006] Boardman, J. and Sauser, B. (2006). System of systems-the meaning of of. In *SoSE*, pages 6–pp, Los Angeles, CA, USA. IEEE.

[Boehm 2006] Boehm, B. (2006). A view of 20th and 21st century software engineering. In *ICSE*, pages 12–29, New York, NY, USA. ACM.

[Bogado et al. 2017] Bogado, V., Gonnet, S., and Leone, H. (2017). Devs-based methodological framework for multi-quality attribute evaluation using software architectures. In *CLEI*, pages 1–10, Córdoba, Argentina. IEEE.

[Cadavid et al. 2020] Cadavid, H., Andrikopoulos, V., and Avgeriou, P. (2020). Architecting systems of systems: A tertiary study. *IST*, 118:106–202.

[Carlock and Fenton 2001] Carlock, P. G. and Fenton, R. E. (2001). System of systems enterprise systems engineering for information-intensive organizations. *Systems engineering*, 4(4):242–261.

[de França and Travassos 2016] de França, B. B. N. and Travassos, G. H. (2016). Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering*, 21(3):1302–1345.

[Dias-Neto et al. 2010] Dias-Neto, A. C., Spinola, R., and Travassos, G. H. (2010). Developing software technologies through experimentation: experiences from the battlefield. In *CIbSE*, Cuenca, Ecuador.

[Dobrica and Niemele 2002] Dobrica, L. and Niemele, E. (2002). A survey on software architecture analysis methods. *IEEE Trans. Softw. Eng.*, 28(7):638–653.

[Firesmith 2010] Firesmith, D. (2010). Profiling systems using the defining characteristics of systems of systems (sos). Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[França and Travassos 2013] França, B. B. N. d. and Travassos, G. H. (2013). Are we prepared for simulation based studies in software engineering yet? *CLEI electronic journal*, 16(1):9–9.

[Guessi et al. 2015] Guessi, M., Graciano Neto, V. V., Bianchi, T., Felizardo, K. R., Oquendo, F., and Nakagawa, E. Y. (2015). A systematic literature review on the description of software architectures for systems of systems. In *SAC*, pages 1433–1440, Salamanca, Spain. ACM.

[Hofmeister et al. 2007] Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., and America, P. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126.

[ISO/IEC/IEEE:15288 2019] ISO/IEC/IEEE:15288 (2019). Systems and software engineering – guidelines for the context of system of systems. pages 1–68.

[ISO/IEC/IEEE:42010 2011] ISO/IEC/IEEE:42010 (2011). Iso/iec/ieee systems and software engineering – architecture description. pages 1–46.

[Maciel et al. 2017] Maciel, R. S. P., David, J. M. N., Claro, D. B., and Braga, R. (2017). Full interoperability: Challenges and opportunities for future information systems. *Grand Research Challenges in Information Systems in Brazil 2016*, 2026:107–116.

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

[Mendes et al. 2018] Mendes, A., Loss, S., Cavalcante, E., Lopes, F., and Batista, T. (2018). Mandala: an agent-based platform to support interoperability in systems-of-systems. In *6th SESoS*, pages 21–28. ACM.

[Nielsen et al. 2015] Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., and Peleska, J. (2015). Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. *ACM Comput. Surv.*, 48(2):18:1–18:41.

[Pelliccione and et al. 2016] Pelliccione, P. and et al. (2016). Architecting cars as constituents of a system of systems. In *SiSoS@ECSA*, page 5. ACM.

[Salado 2015] Salado, A. (2015). Abandonment: A natural consequence of autonomy and belonging in systems-of-systems. In *SoSE*, pages 352–357. IEEE.

[Salado 2016] Salado, A. (2016). Exile: A natural consequence of autonomy and belonging in systems-of-systems. In *SysCon*, pages 1–5. IEEE.

[Teixeira et al. 2020] Teixeira, P. G., Lebtag, B. G. A., dos Santos, R. P., Fernandes, J., Mohsin, A., Kassab, M., and Graciano Neto, V. V. (2020). Constituent system design: A software architecture approach. In *ICSA Companion*, pages 218–225. IEEE.

[Yokell 2018] Yokell, M. (2018). Overview of system of systems (sos) managerial and operational affinity: Assessing and improving relationships within systems of systems. In *SoSE*, pages 438–443. IEEE.

[Zeigler et al. 2013] Zeigler, B. P., Sarjoughian, H. S., Duboz, R., and Soulié, J.-C. (2013). *Guide to modeling and simulation of systems of systems*. Springer.