Architectural Strategy to Enhance the Availability Quality Attribute in System-of-Systems Architectures: a Case Study

Alexandre Delecolle, Rodrigo Silva Lima, Valdemar Vicente Graciano Neto, Jérémy Buisson

Architectural Strategy to Enhance the Availability Quality Attribute in System-of-Systems Architectures: a Case Study

Alexandre Delécolle
Département Informatique
Écoles de Saint-Cyr Coëtquidan
Guer, France
alexandre.delecolle@st-cyr.terre-net.defense.gouv.fr

Rodrigo Silva Lima
Universidade do Sudoeste da Bahia
Vitória da Conquista, Brazil
rodrigolima2804@gmail.com

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

Jérémy Buisson
CREC / IRISA
Écoles de Saint-Cyr Coëtquidan
Guer, France
jeremy.buisson@st-cyr.terre-net.defense.gouv.fr

***Abstract*—Independent software systems have been joined together to form alliances known as System-of-Systems (SoS). SoS are composed of constituent systems and deliver behaviors as a result of the combination of their individual functionalities. In both, single systems and SoS, the prediction of quality at design-time is of utmost importance. Among several quality attributes (QA) a SoS should cope with, availability is particularly important. Given the critical nature of several domains supported by SoS, failures and unavailability could lead to risks and losses to the SoS users. SoS are then required to be highly available. However, while the strategies to deal with availability in single systems are well-known, techniques to predict and enhance availability at SoS level while considering its inherent dynamics can be still investigated. The main contribution of this paper is the establishment of an architectural strategy at SoS level to enable a simulation-based measurement of availability at SoS architectural descriptions. Preliminary results reveal that the established strategy can be successfully used for the purpose of measurement of the QA availability in SoS architectures.**

***Index Terms*—Systems-of-Systems, SoS, Quality Attribute, Availability, Software Architecture, Tactic, Strategy, Simulation-Based Study.**

I. Introduction

The complexity of software systems has grown in a high rate. Modern systems are intensely embedded with software to automate their operation and support better precision for their functionalities. Moreover, those so-called *software-intensive systems* have been pressured to form alliances of multiple systems and interoperate to deliver functionalities as a result of the combination of their individual capabilities [2], [7], [12]. This dynamic set of interoperating systems has been called as Systems-of-Systems (SoS[1]) [12], [17], [22].

[1]For sake of simplicity, herein the SoS acronym will be interchangeably used to denote both singular and plural forms.

SoS are required to exhibit quality, i.e., SoS should cope with users needs by delivering their behaviors according to a set of quality attributes (QA). As a matter of consensus, software architectures enhance quality as they materialize a set of design decisions made to address a set of prioritized QA [4], [19]. Among the several QA a SoS should address, *availability* is particularly important. For instance, if a SoS for Emergency Response Management is unavailable at a certain moment, users can be exposed to higher risks and become helpless. However, dealing with availability at SoS level is even more complicated than doing it for isolated systems, due to (i) the operational independence and autonomy of the constituents, i.e., systems operate not only in the context of SoS but also outside and for their own purposes, (ii) constituents can join or leave the SoS at runtime, which turns the SoS architecture to be dynamic and (iii) availability assurance depends on the presence of constituents that provide the required capabilities to deliver a set of expected behaviors. Hence, it is prominently important to assure, still at design-time, that the SoS architecture will be able to deliver the expected behaviors and be available, despite the existence of the above-mentioned complicating factors. The following research question is then raised: *How can we design and specify architectural strategies to address the quality attribute Availability in Systems-of-Systems?*

The main contribution of this paper is the establishment and design of an architectural strategy to deal with availability at SoS architectural design level. In the classic literature, architectural tactics comprise architectural strategies/decisions developed to enhance QA. We reviewed the literature to obtain the architectural tactics used to deal with availability in single systems and adapted one of them to develop an architectural strategy based on redundancy of capabilities and

replaceability to deal with availability at SoS level. We used a new language (named SoSADL, an acronym from System-of-Systems Architecture Description Language [15], [19], [22]) to specify this strategy and this SoS. Moreover, we relied on a model transformation to automatically generate simulation models specified in DEVS to assess the effectivness of the established strategy [16]. We ran the simulation models to observe (i) the expected behaviour and (ii) whether availability was effectively addressed at that SoS. We specified a Smart Parking composed of independent systems that interoperate to achieve a common goal: *sensing the parking and suggesting a position for cars so they could park*. If a sensor fails, an alternative system replaces the sensor so that the expected behaviour remains available. Simulation results reveal that our architectural strategy was well-succeeded to deal with availability at SoS level, achieving the goal in 100% of the times.

The remainder of this paper is structured as follows: Section II presents the foundations; Section III details the scientific methodology we used; Section IV describes the study and evaluation; Section V brings discussion and Section VI concludes the paper with final remarks.

II. Background

A software architecture comprises software elements, the external properties of those elements, and the relationship among them [25]. Architectures are important because they are the backbone of any well-succeeded system [4]. According to Kruchten [10], a software architecture describes not only the software structure, but also the behavioral description. The architecture should document the system to be used by different stakeholders or to be reused in another system. The same rationale is also valid for SoS [1], [22]. Indeed, a SoS also has an associated software architecture, which comprises the constituents, their interaction links, and the shared decisions and rationale that trigger its building [21]. The constituent systems somehow interact with each other and interactions follow architectural rules, which characterizes the constituents interoperability.

SoS are often defined under Maier's criteria [12]. According to the principle of **operational independence of constituents**, the *constituent systems* should respect rules to be part of the SoS, while also operating independently of it. The **managerial independence of constituents** stated that they can be owned by different enterprises, so each system has to be enough independent to be managed and repaired. The **evolutionary development** comprises the ability of the overall SoS to evolve as a result of the evolution of constituents, goals, or architectural arrangements. Constituents are **distributed** and interoperate via some interoperability link. Finally, the **emergent behaviors** are behaviors or functionalities which emerge thanks to the interactions of the constituents.

Several languages exist to support the specification of software and system architectures, such as UML[2] and SysML[3]. In this paper, we rather use SosADL (System-of-Systems Architecture Description Language) [22]. Unlike SysML or UML, which require precise description of the constituents, SosADL is designed for SoS where the SoS architect cannot precisely predict, at design-time, the constituent systems that can be part of the SoS at runtime. In a SoS, a system can be added, removed or changed at runtime, which certainly impacts on the architecture. Formally founded on π-calculus for SoS [23], a novel formalism to describe intercommunicating processes, SosADL supports all these constructions [18]. In short, SosADL describes SoS, which can be expressed as a combination of architectures, systems, and mediators declarations. Mediators are architectural elements concerned with establishing communication between two or more systems. Mediators and systems have gates and behaviors: gates are abstractions that denote interaction endpoints; and behaviors are given as π-calculus processes. Because a SoS is a system, a SoS architecture has gates and behaviors as well, materialized in this case by a coalition. Coalitions are an arrangement of systems (constituents), which are mediated by mediators [24].

One of the goals an architect should achieve is to cope with a specific set of quality attributes (QA). A QA is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders [25] (e.g. the user or the owner of this system). QA are often result of the software architecture design decisions. Design decisions can enable or inhibit some QA [25]. For instance, if we decide to restrict some communications between components in a system, that decision can influence on the interoperability. Architectural tactics are design decisions known to address or reinforce a specific QA [25]. Architectural tactics result from the analysis of several real projects to realize the design decisions taken to deliberately address specific QA. We call our design solutions as *architectural strategies* to denote proposed design decisions, in contrast to *architectural tactics* that describe state of practice (design decisions already taken and prescribed).

A QA can be harder to be achieved in a SoS than in a single system because SoS raises the following specific challenges: 1) For the overall SoS, the QA should be achieved as a result of the guarantee of it for both the individual systems and for the coalition; 2) There is no standardized quality model to evaluate QA in SoS [27]; 3) Sometimes, emergent behaviours can not be readily predicted.

A. Related work

The investigation on availability (and more broadly, reliability) in SoS is not a new trend. Holt *et al.* [8] propose a Model-Based System Engineering (MBSE) approach to reuse existing design solutions and adapt them for SoS. Mokhtarpour and Stracener [14] report the problem of reliability in SoS. Reliability is actually considered a broader QA that involves availability, according to ISO 25010 [9]. The study reports an evaluation of this QA with the capability of the entire SoS to share data. The study shows how to measure or evaluate the QA in SoS. Sanduka and Obermaisser [26] explain how

[2]https://www.uml.org/

[3]http://www.omgsysml.org/

to use MBSE approaches to deal with reliability in SoS. UPDM (Unified Profile for DoDAF and MODAF, which are US Department of Defense and British Ministry of Defense Architecture Frameworks) models are extended along with a process that connects some of their parts and combines them into a methodology to support reliability. Finally, Garro and Tundis [5] deal with the lack of methods to address the evaluation of non-functional requirements especially in SoS. Moreover, they propose, similarly to our investigation, to detail those requirements or QA because of the insufficiency of standardized definitions. The definition of SoS explains why we should adapt existing techniques to make them suitable to that specific domain. This last study is the closest to our work because it proposes new definitions and adaptations for treating QA and requirements in SoS domain. Next section presents the established methodology.

III. Methodology

The investigation was led according to the following steps:

- **Step 1. Literature Review.** As presented in Section II, we analysed several conference papers and journal articles on ***Software architecture and System-of-Systems*** in order to understand how to design a system and what is an architecture in the software engineering domain.
- **Step 2. Project Setting.** Once the gaps were identified at step 1, we decided to contribute to the state-of-the-art by establishing an architectural strategy to deal with QA availability in SoS. We looked for an instance of a SoS (a Smart Parking) that could be specified in order to evaluate the success of our work.
- **Step 3. Solution development - Replaceability.** After prototyping the SoS, we started to develop an architectural strategy to deal with a specific QA we choose: availability. The proposed solution was ***replaceability***: it consists in providing redundant functionalities and constituents so that if a failure occurs, the availability of the SoS is not affected.
- **Step 4. Evaluation.** For evaluation purposes, we decided to use an existing model transformation [20] that automatically generates DEVS simulation models to assess whether the SoS exhibits the expected architecture (structure) as well as the expected behavior. This part also shows the execution of the simulation. By simulating the corresponding code, we could obtain results in order to analyze them and answer our research question.

IV. Study Conduction and Evaluation

We followed the steps described in Section III to perform our investigation. The **Step 1** (Literature Review) is described at Section II.

Step 2. Project Setting (Domain description). The study was conducted in a specific domain of SoS: Smart Parking. Our Smart Parking is composed of the following systems, as conceptually shown in Figure 1: cars, fixed sensors at parking positions (the robot is explained later at Step 3). When a car arrives at the parking, it asks for a position to park to the smart

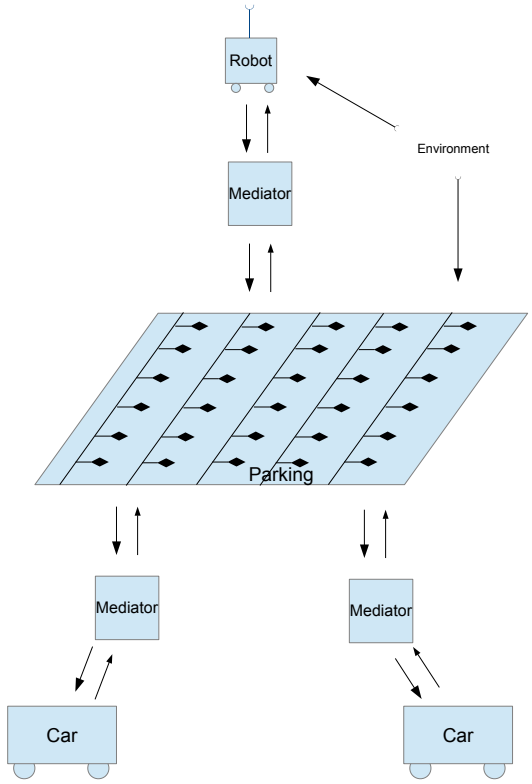

Fig. 1. Architecture of the Smart Parking.

parking system. Sensors at positions are used by the parking to detect what positions are available or not. The parking then responds with the location of one free position to the car.

Step 3. Solution Development. According to the ISO 25010 [6], availability comprises the degree to which a system, product or component is operational and accessible when required for use. But this definition is not really appropriate for SoS. We assume that *SoS availability comprises the degree to which the SoS (as a coalition of its constituents) is capable of delivering a behavior when required*. We measure it as the percentage of the times the SoS delivers the expected behaviour in regard to how many times it is asked to.

To deal with availability, we choose to rely on *Replaceability* principle [11], which states that *the SoS should offer redundant (but not necessarily semantically identical) functionalities in different constituent systems such that, if one fails, there is another one to substitute it as a backup*. In our case, redundancy is obtained by introducting an additional constituent system that provides a mobile sensor functionality. This mobile sensor is an alternative mean for the parking to detect whether a position is available or not. In a real parking, the mobile sensor can be implemented, e.g., by a sensor embedded in a shuttle system, whose route may be adapted depending on the positions to check. In this paper, we consider a robot to elude other missions of that system.

This smart parking is a SoS according to the Maier's criteria. The car and mobile sensor systems are operationally and managerially independent. The car belongs to its driver,

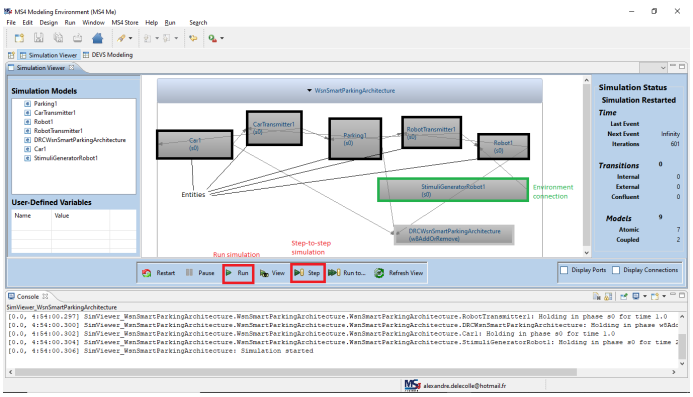

Fig. 2. Presentation of the simulation window on MS4ME.

Datatype	0	1	2	3
Demand	no request	request		
SensorNumber	there is no place and the Parking independently works to provide the answer	there is a place and the Parking provides by its own	there is a place and the Robot provides it	there is no available place and the Robot provides the answer
Place		simulation of a sending place		
Sensor	the place is available	the place is filled		

TABLE I
MEANING OF THE VALUES OF EACH DATATYPE IN THE SMART PARKING.

and obviously fulfills other missions as transportation of its passengers and their goods. As previously explained, the mobile sensor could be provided by a shuttle system, e.g., belonging to the public transportation system and transporting customers. Interactions between the systems yield to the emergent behavior of reduced wander by telling cars where they can park.

Step 4. Evaluation. We modeled our Smart Parking using the SosADL workbench. Then, a model transformation (SosADL2DEVS) was used to automatically transform SosADL models into DEVS models [20]. The resulting DEVS models are *dnl* files and *ses* files interpreted by MS4ME simulator[4]. On that simulator, the *dnl* files are automatically converted to Java files to materialize the simulation execution engine. After that, the *ses* files are used to launch the simulation. The environment is modeled by a Stimuli Generator [19], which is generated along with the DEVS models by the SosADL2DEVS transformation. A video[5] is externally available to explain the process and to present how the simulator helps to show the SoS behaviour being delivered.

After launching the simulation, we could observe the different constituents of the Smart Parking, i.e., the systems and the mediators (see the simulator window in Figure 2). *DRCWSmartParkingArchitecture* is an artifact generated by SosADL2DEVS to perform dynamic reconfiguration, which is out of the scope of this paper. Figure 2, in turn, shows the simulation execution as a step-by-step procedure, so that emergent behaviors, messages exchange (and constituents interoperability) and SoS architecture can be observed.

For improved readability of subsequent paragraphs, we use symbolic constants rather than the values given in Table I.

The simulation of the Smart Parking starts when the Car asks for a place. The Car sends a *Demand* with the value EFFECTIVEDEMAND (value 1). The CarTransmitter receives it and forwards it to the Parking. After that, the Parking has two possibilities to proceed: (i) answering according to the fixed sensors; or (ii) asking to the Robot to look for free positions. According to the result of sensing positions, the Parking answers either NOPLACEPARKING (value 0 - no parking position is available) or PLACEPARKING (value 1 - location of one free parking position).

The latter behaviour is triggered when some fixed sensors fail. In such a case, the Parking can send PLACECHECKED (value 1 - parameterized by the position with failed fixed sensor) to the Robot through RobotTransmitter. By this message, the Parking asks the Robot to change its route such that it checks whether the position is available or occupied. When the Robot passes in front of the position, the Robot senses either AVAILABLE (value 0) or FILLED (value 1) depending on the observed status of the position. In our simulation, the Stimuli Generator establishes and delivers random values (either AVAILABLE or FILLED) for the Robot sensing. Depending on the sensed value, the Robot respectively answers PLACEROBOT (value 2) or NOPLACEROBOT (value 3) to the Parking, which can in turn replies back to the Car, as previously described.

Figure 3 shows one message being sent: Car1 sends a Demand with the value 1 to CarTransmitter1. In the next step, the CarTransmitter1 sends the same message to the Parking1 as we explained. In the step-to-step simulation, we observe messages such as those all the time and it reproduces the actual communications in the SoS.

A. Reporting

Despite the high degree of abstraction, the SoS behavior comprises, basically, a parking that delivers an empty place or notifies when all places are occupied. Figure 4 plots experimental results: it shows the answers of the Smart Parking to the Car demands. Each square represents one demand-answer: when green, the answer is provided by the fixed sensors; when yellow, the Robot is involved. To simplify, our simulation considers that at most one fixed sensor does not work properly. The plot shows that, despite intermittent failure of the fixed sensor, the Smart Parking always answers to the Car: it is 100% available. So the replacement with the Robot is effective.

[4]http://www.ms4systems.com/pages/ms4me.php
[5]https://vimeo.com/377294844

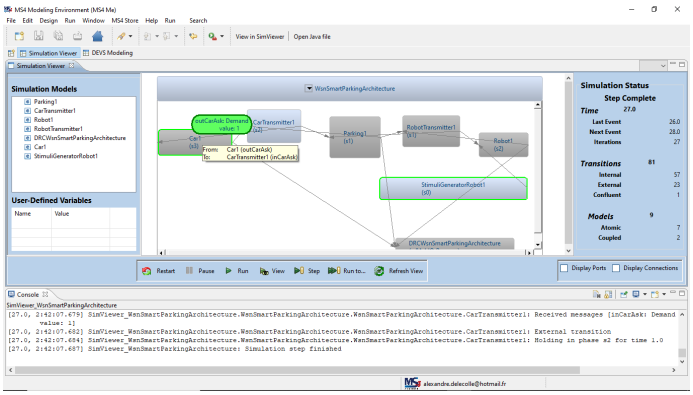

Fig. 3. Example of a message in the simulation step-to-step of the Smart Parking

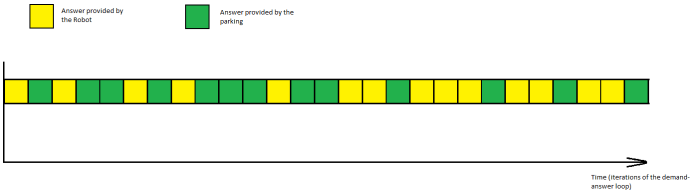

Fig. 4. Graph showing how the SoS answers to a Car ask

After obtaining the results of our study, we can answer the posed question: *Can the SoS provide availability as a result of a design decision?* The answer is Yes, at least for the illustrated scenario, by introducing redundant functionalities in the SoS. In our case study, the Robot implementation of the mobile sensor system would correspond to a directed SoS; while a shuttle system would correspond to an acknowledged SoS. In both cases, the Smart Parking architect actively requests for the mobile sensor functionality to deliberately address the availability QA.

Threats to Validity. Our study relied on a model transformation that generates simulation models from a SoS architectural specification. One threat is related to the correctness of the model transformation. However, this threat is alleviated by the fact that, over the last years, more than 1,300,000 DEVS lines of functional code were automatically and correctly generated by the model transformation [13].

Another threat regards the empirical procedure itself. As a matter of fact, an empirical investigation should establish a varying set of randomness to obtain a reliable result on the effectiveness of the proposed solution. We only investigated one configuration of randomness. Hence, further studies are required to empower evidence obtained in this study.

An additional threat is how realistic are the SoS and its simulation. Having a robot whose sole purpose is checking parking positions is questionable. But, as already discussed, the mobile sensor system may be part of a larger system, such as a shuttle system transporting persons or a delivery system transporing goods. Considering one or the other does not affect the architecture of the Smart Parking SoS, as such missions are the ones operated independently of the SoS.

V. Discussion

Standard practices for dealing with QA are well-established for single systems. However, practices for SoS can be still improved, particularly due to their inherent characteristics. We use a model transformation to automatically generate an executable simulation model in order to assess whether our solution delivers the expected result: providing availability as a result of replaceability. We show that our SoS architectural specification is correct about the expected structure and behavior. This is a contribution to the establishment of reliable methods to support evaluation of SoS software architectures. We contribute to the state-of-the-art of this domain by establishing foundations on how to address the quality attribute availability in a SosADL-described SoS.

In order to proceed with this project, we have to evolve the model transformation to make it suitable to our original and more complete specification. The transformation should be adapted to all the SosADL specification (only a subset of the language grammar is currently addressed). The most important part is the data structures which are needed, and the language does not support them yet.

For the smart parking, we can add some parameters to improve it, regardless of the enhancement of the QA. For example, we could add the coordinates for the parking to send the coordinates of the available place to the car. Moreover, to specify a feasible parking, we could add the payment methods in the SoS specification, besides counting how many times the car was parked there.

The interoperability can be another strategy to enhance availability [3]. Another solution could enhance cars to communicate with other cars, such as in vehicular network approaches. It means that, if a car is parked and it can find an available position closer to it, it can send this place to the other cars. A car can also notify when it leaves a position. This improvement paves the way to dealing with collaborative SoS. In such a case, the design strategy should also advocate incentives to make Cars voluntarily collaborate to the SoS, for instance, by means of differentiated service levels depending on the level of collaboration.

VI. Final Remarks

In this paper, we report an investigation of the literature and the development of a solution to answer the following research question: how can we design and specify architectural strategies to address the quality attribute *availability* in a System-of-Systems? We specified a solution based on replaceability and redundancy to deal with availability at SoS architectures. The original idea was provided by a prior work [11]. We could confirm its validity by specifying the solution and evaluating it through simulation models. In our design (Smart Parking SoS), availability can actually be addressed by replaceability.

We also contributed to the characterization of what could be considered availability at SoS level: previously, availability was not defined for SoS but only for single systems in the standard ISO 25010 [6]. We then proposed a distinct definition,

which is *the degree to which the entire SoS is capable of delivering a behaviour when required*. In our solution, redundancy of functionalities is provided by a Robot. Although the Robot constituent could have other functionalities, in this context its only purpose is to serve the SoS through replaceability. As discussed, it could be part of a larger system, which in turn retains operational and managerial independence, hence satisfying Maier's criteria.

This project also contributed to the evaluation of quality attributes in SoS software architectures. As a matter of fact, each QA requires a different approach and a specific strategy to be addressed. This paper proposes to use redundancy or replaceability to enhance availability. We had to adapt it to make it work at SoS level and in fact exchange special messages for that. This is quite different from all the other ways to perform it for single systems and it is the main contribution of this work. And it opens opportunities for further studies on software quality for SoS.

ACKNOWLEDGEMENTS

The authors thank the institutions involved in the project. Particularly, we thank Écoles de Saint-Cyr Coëtquidan by financially supporting the first author during his internship in Goiânia, Brazil, under the supervision of the third author. We also thank the Federal University of Goiás by providing the environment where the research was conducted. Finally, the authors also thank MS4 Systems Inc. for the license of MS4Me tool granted to the INSIGHT-GO research group (INF/UFG).

REFERENCES

[1] Architecture Capability Team. NATO Architecture Framework, version 4. Technical report, January 2018.

[2] Sabine Buckl, Sascha Krell, and Christian M Schweda. A formal approach to architectural descriptions–refining the iso standard 42010. In *International Workshop on Cooperation and Interoperability, Architecture and Ontology*, pages 77–91. Springer, 2010.

[3] Juliana Fernandes, Francisco Henrique Ferreira, Felipe Cordeiro, Valdemar Vicente Graciano Neto, and Rodrigo Santos. How can interoperability impact on Systems-of-Information Systems characteristics? Providing guidelines for a tradeoff between architectural strategies for interoperability in SoIS and the degree of autonomy of constituents. In *Brazilian Symposium on Information Systems (SBSI 2020)*, pages 1–8, São Bernardo do Campo, Brazil, May 2020. ACM.

[4] Lina Garcés, Flavio Oquendo, and Elisa Yumi Nakagawa. Software mediators as first-class entities of systems-of-systems software architectures. *Journal of the Brazilian Computer Society*, 25(1):8, 2019.

[5] Alfredo Garro and Andrea Tundis. On the reliability analysis of systems and sos: The ramsas method and related extensions. *IEEE Systems Journal*, 9(1):232–241, 2014.

[6] Oleksandr Gordieiev, Vyacheslav Kharchenko, Nataliia Fominykh, and Vladimir Sklyar. Evolution of software quality models in context of the standard iso 25010. In *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland*, pages 223–232. Springer, 2014.

[7] Tihana Galinac Grbac and Goran Mauša. On the distribution of software faults in evolution of complex systems. In *Proceedings of the International Colloquium on Software-intensive Systems-of-Systems at 10th European Conference on Software Architecture*, page 2. ACM, 2016.

[8] Jon Holt, Simon Perry, Richard Payne, Jeremy Bryans, Stefan Hallerstede, and Finn Overgaard Hansen. A model-based approach for requirements engineering for systems of systems. *IEEE Systems Journal*, 9(1):252–262, 2014.

[9] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.

[10] Philippe B Kruchten. The 4+ 1 view model of architecture. *IEEE software*, 12(6):42–50, 1995.

[11] Rodrigo Silva Lima. Simulation-Based Availability Assessment Systems-of-Systems Software Architectures (in portuguese), 2019. Final report - Monograph presented to Universidade do Sudoeste da Bahia.

[12] Mark W Maier. *Architecting principles for systems-of-systems*, volume 1. Wiley Online Library, 1998.

[13] Wallace Alves Esteves Manzano, Valdemar Vicente Graciano Neto, and Elisa Yumi Nakagawa. Simulating systems-of-systems dynamic architectures. *Electronic Journal of Scientific Initiation in Computing*, 17(2), 2019.

[14] Behrokh Mokhtarpour and Jerrell T Stracener. Mission reliability analysis of phased-mission systems-of-systems with data sharing capability. In *2015 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6. IEEE, 2015.

[15] Valdemar Vicente Graciano Neto. Validating emergent behaviours in systems-of-systems through model transformations. In *SRC at MoDELS*, pages 1–6, Saint Malo, France, 2016.

[16] Valdemar Vicente Graciano Neto. *A simulation-driven model-based approach for designing software intensive systems-of-systems architectures*. PhD thesis, 2018.

[17] Valdemar Vicente Graciano Neto, Lina Garcés, Milena Guessi, Carlos Paes, Wallace Manzano, Flavio Oquendo, and Elisa Nakagawa. Asas: An approach to support simulation of smart systems. pages 5777–5786, 2018.

[18] Valdemar Vicente Graciano Neto, Flávio E. A. Horita, Everton Cavalcante, Adair José Rohling, Jamal El Hachem, Daniel Soares Santos, and Elisa Yumi Nakagawa. A study on goals specification for systems-of-information systems: Design principles and a conceptual model. In *Proceedings of the XIV Brazilian Symposium on Information Systems, SBSI 2018, Caxias do Sul, Brazil, June 04-08, 2018*, pages 21:1–21:8, 2018.

[19] Valdemar Vicente Graciano Neto, Carlos Eduardo Barros Paes, Lina Garcés, Milena Guessi, Wallace Manzano, Flavio Oquendo, and Elisa Yumi Nakagawa. Stimuli-sos: a model-based approach to derive stimuli generators for simulations of systems-of-systems software architectures. *Journal of the Brazilian Computer Society*, 23(1):13, 2017.

[20] Valdemar Vicente Graciano Neto, Lina Maria Garcés Rodriguez, Milena Guessi, Carlos Eduardo de Barros Paes, Wallace Manzano, Flávio Oquendo, and Elisa Yumi Nakagawa. ASAS: an approach to support simulation of smart systems. In *51st Hawaii International Conference on System Sciences, HICSS 2018, Hilton Waikoloa Village, Hawaii, USA, January 3-6, 2018*, pages 5777–5786, 2018.

[21] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Comput. Surv.*, 48(2):18:1–18:41, September 2015.

[22] Flavio Oquendo. Formally describing the software architecture of systems-of-systems with sosadl. In *2016 11th system of systems engineering conference (SoSE)*, pages 1–6. IEEE, 2016.

[23] Flavio Oquendo. π-Calculus for SoS: A Foundation for Formally Describing Software-intensive Systems-of-Systems. In *SOSE*, pages 7–12, Kongsberg, Norway, June 2016. IEEE.

[24] Flavio Oquendo and Axel Legay. Formal Architecture Description of Trustworthy Systems-of-Systems with SosADL. *ERCIM News*, (102), 2015.

[25] Jared Reviewer-Herzog. *Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman*, volume 40. ACM, 2015.

[26] Imad Sanduka and Roman Obermaisser. Model-based development of systems-of-systems with reliability requirements. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1531–1538. IEEE, 2015.

[27] Daniel Soares Santos, Brauner R. N. Oliveira, Adolfo Duran, and Elisa Yumi Nakagawa. Reporting an experience on the establishment of a quality model for systems-of-systems. In *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, pages 304–309, 2015.