Towards Architectural Synthesis of Systems-of-Systems

Marcelo B. Gonçalves[1,2], Flavio Oquendo[2], Elisa Y. Nakagawa[1]

[1]ICMC, University of São Paulo, São Carlos, Brazil
[2]IRISA-UMR CNRS/Université de Bretagne-Sud, Vannes, France

{marcelob,elisa}@icmc.usp.br, flavio.oquendo@irisa.fr

Abstract. *A System-of-Systems (SoS) is the result of constituent systems interoperating to achieve common goals. This emerging class of systems brings new development challenges, especially for the design of their software architectures. Despite architectural synthesis is a relevant activity in the architectural process with impact in the quality of software architectures, current software engineering approaches do not properly address it in the context of SoS. To address this lack, the main contribution of this work is to present SASI (SoS Architectural SynthesIs), a method to support architectural synthesis in SoS software architectures. Besides introducing the main elements of SASI, this paper reports results of a study aimed to assess the feasibility of the proposed method.*

Resumo. *Um Sistema-de-Sistemas resulta de sistemas constituintes interoperando para alcançar objetivos comuns. Essa nova classe de sistemas traz desafios de desenvolvimento, especialmente em seu design arquitetural. A despeito da síntese arquitetural ser uma atividade importante no processo arquitetural, com impacto na qualidade das arquiteturas resultantes, as abordagens atuais não tratam adequadamente essa atividade no contexto de sistema-de-sistemas. A fim de tratar essa lacuna, a contribuição principal deste trabalho é a introdução do SASI, um método para oferecer suporte à síntese arquitetural em arquiteturas de software de sistemas-de-sistemas. Além de introduzir os principais elementos dessa proposta, este artigo reporta resultados de um estudo de viabilidade realizado para o método proposto.*

1. Introduction

In the last years, it has been possible to notice an increasing interest in the research and development of *Systems-of-Systems* (SoS), a class of complex systems that stems from the interaction among distributed, heterogeneous, and independent constituent systems that interoperate to form a larger, more complex system for accomplishing global missions [Maier 1998]. The result of such a collaborative work is said to be more than the sum of the constituent systems as it enables the SoS to offer new functionalities, i.e., an emergent behavior that cannot be provided by these constituents individually [Boardman and Sauser 2006].

SoS have followed the natural trend of complex, large-scale systems becoming more software-dependent, leading to the so-called *software-intensive SoS*[1], i.e., SoS in which software plays an essential role in their design, development, and evolution [Gonçalves et al. 2014]. As for any software-intensive system, *software architectures*

[1]For the sake of simplicity, software-intensive SoS are hereinafter referred as SoS.

have been regarded as a significant element for determining the success of such systems and taming their complexity, while contributing to the achievement of important quality requirements such as interoperability and performance. In the SoS context, software architectures must be able to address the inherent characteristics of SoS, encompass the organization of constituent systems, and deal with a dynamic and evolutionary context.

Within the construction of software architectures, the *architectural synthesis* aims to design architectural solutions in order to meet the requirements upon the system that directly influence their architecture, the so-called architecturally significant requirements [Hofmeister et al. 2007]. Despite the relevance of this step as a driver for the system implementation, existing approaches in the literature do not properly address architectural synthesis in the SoS context, being typically concerned with systems engineering and high-level aspects of SoS architectural development. In addition, it is necessary to reason which are the fundamental elements for the architectural synthesis in the development of SoS software architectures.

Aiming at tackling these issues, this paper presents SASI (SoS Architectural Synthesis), a method for establishing and managing architectural solutions in *acknowledged SoS*, in which goals, resources, and authority are all recognized at SoS level, but the constituent systems retain their independent management and the behavior is not subordinated to the central managed purpose [DoD 2008]. SASI was conceived to provide guidelines on what must be done in the construction of acknowledged SoS software architectures and support how to perform the required activities for architectural synthesis in any application domain. In order to assess the feasibility of our proposal, we conducted an observational study with six participants. The obtained results contributed to improve SASI and to verify its applicability and independence in terms of application domain.

The remainder of this paper is structured as follows. Section 2 introduces the issues related to the architectural synthesis of SoS software architectures. Section 3 presents our proposal to support this activity. Section 4 presents the evaluation concerning our proposal. Finally, Section 5 outlines some conclusions and directions to future work.

2. Architectural Synthesis of SoS

According to the general model of architectural design proposed by Hofmeister et al. [Hofmeister et al. 2007], the major activities on building any software architecture are (i) architectural analysis, (ii) architectural synthesis, and (iii) architectural evaluation. In the architectural analysis, architecturally significant requirements are elicited expressing which problems in the system context the software architecture can solve. In turn, the architectural synthesis encompasses the design and proposition of candidate architectural solutions in order to effectively solve the architecturally significant requirements, moving from the problem to the solution space. Finally, the architectural evaluation aims at evaluating the proposed architectural solutions against the architecturally significant requirements. In order to verify if the design decisions made are the right ones.

Despite the existence of approaches covering the synthesis of software architectures, the class of SoS reaches a complexity threshold in which traditional software engineering is no longer sufficiently adequate [Boehm and Lane 2006]. Due the inherent characteristics of SoS, they software architecture differ from a monolithic systems on several issues, such as the communication involving multiple stakeholders and organi-

zations, evolutionary development, dynamism in an operational environment based on emergent behaviors [DoD 2008]. As a result, SoS software architectures have been constructed through ad-hoc perspectives in which each SoS has its software architecture developed on a particular manner. Therefore, there is a lack of further investigations on the systematic development of SoS software architectures (and more specifically on the architectural synthesis) in order to improve the available solutions to support the architectural development of these architectures.

Due the independence of constituent systems, SoS software architectures are inherently dynamic. Moreover, emergent behaviors result from the collaborative work of constituent systems and these systems are not totally subordinated to SoS interests. Therefore, the architectural solutions must consider the relevance of self-organization concerns and prediction of both desired and undesired emergent behaviors. In general, desirable behaviors come from architectural solutions and must be maximized since they foster the accomplishment of SoS missions. On the other hand, undesirable behaviors must be minimized because they may negatively affect the accomplishment of SoS missions and/or important quality attributes such as performance, security, and reliability.

3. SASI: A Method for Architectural Synthesis of SoS

SASI, our method to support architectural synthesis of acknowledged SoS, is structured upon the OMG's Essence Standard[2], which comprises a language for method authoring and a kernel designed to be a reference for software development projects [Jacobson et al. 2013]. With the Essence Language, it is possible to instantiate processes by using *kernels* and *practices*: kernels provide conceptual grounding and guidelines to "what must be done" whereas practices provide elements such as specific activities and work products determining the "how to do". Our previous work [Gonçalves et al. 2015] proposed the SoS SA Kernel, a kernel that adheres to the Essence Kernel for determining the "what must be done" in the construction of acknowledged SoS software architectures, independently from application domains, work products, and organizational contexts. We have taken advantage of SoS SA Kernel when conceiving SASI in order to support project teams on the instantiation or improvement of their own development processes targeting acknowledged SoS.

The main elements borrowed from the Essence Language and employed in the representation of SASI are *alphas*, *work products*, *activity spaces*, and *activities*. Alphas determine the "things to work with" of methods and have a set of progression states verified through checklists. Changes in these states indicate work progress and can help development teams to understand their own way of working. A work product is an artifact that concretely represents an alpha, e.g., a document or a code slice. Activity Spaces determine the "what must be done" in development projects and activities define approaches to accomplish activity spaces by providing guidelines on how to work with alphas when following a given method. Figure 1 shows the main workflow of SASI for architectural synthesis. Next subsections detail the main elements of the method.

3.1. SASI Work Products

In our previous work [Gonçalves et al. 2015], several alphas were established in order to encompass all the "things to work with" when architecting acknowledged SoS. Follow-

[2]Essence Standard is avaiable at <http://www.omg.org/spec/Essence/>

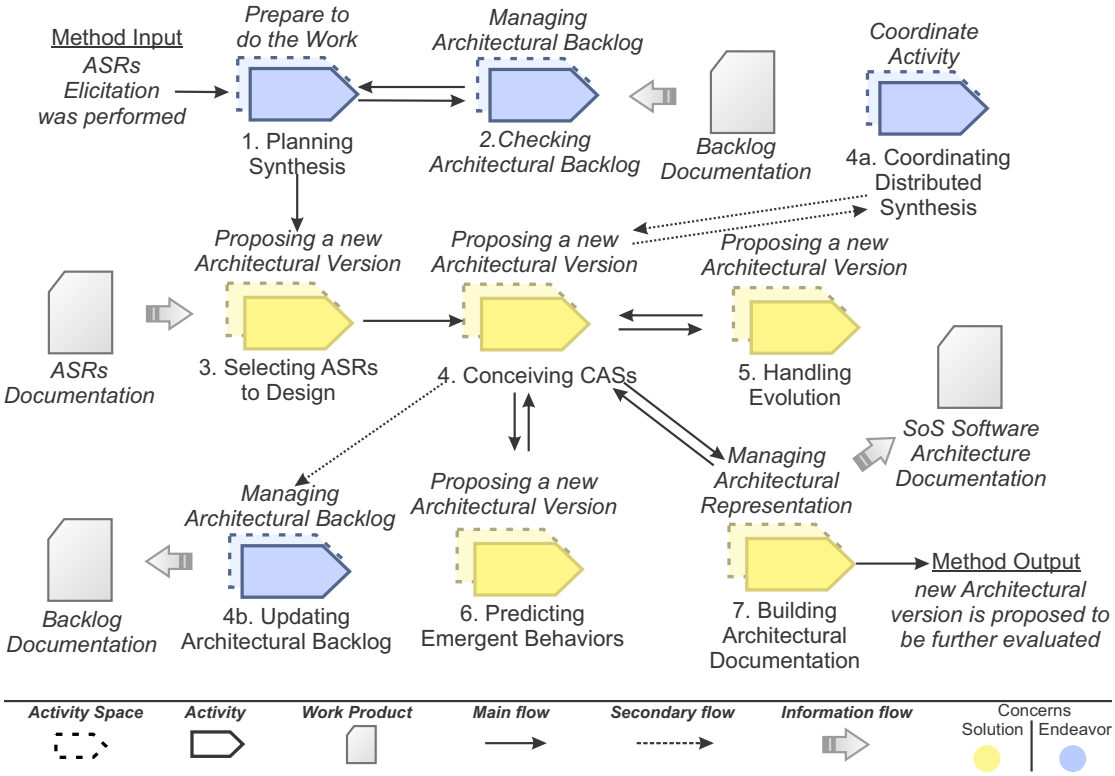

Figure 1. SASI workflow.

ing, we describe the work products introduced by SASI that might express these alphas considering only the perspective of architectural synthesis.

Architectural Backlog Documentation. The Architectural Backlog encompasses any relevant matter related to the architecture and it can be updated at any time in the life cycle. In this context, the Architectural Backlog documentation must comprise the knowledge to guide the architectural development, including issues such as the prioritization of architectural requirements, feedback of problems found during the architectural evaluation, and registration of new ideas for viability analysis in further iterations.

Architecturally Significant Requirements (ASRs) Documentation. An ASR is any functional or non-functional requirement relevant for the SoS software architecture and therefore drives the architectural design. In the SoS context, ASRs are often related to quality attributes, missions, constraints, and requirements derived from environmental conditions. As a recommended work product, the ASRs documentation is obtained after agreement through different stakeholders to be further handled by the architecture, including, for example, a quality model created to provide specific metrics and constraints for relevant quality attributes in SoS such as interoperability, security, and performance. Furthermore, despite we do not expect all ASRs to be known a priori, the ASRs elicitation is a precondition to perform architectural synthesis.

SoS Software Architecture Documentation. The architectural documentation is proposed to explicity express, support evaluation, and convey an SoS software architecture. Therefore, it must provide elements required to adequately express the architecture

of the system, such as architectural viewpoints and views, or architectural prototypes. Moreover, it can be made with different representation techniques (informal, semi-formal, and formal) and must cover the different views (e.g., structural and behavioral) to provide a better understanding of the system architecture to both stakeholders and developers.

3.2. SASI Activities

Following, we present the activities defined in SASI.

1. Planning Synthesis. This activity deals with the establishment/update of planning issues concerning the execution of synthesis activities. It includes scheduling time and resources, e.g., the establishment and management of the team and its way of working with stakeholders and other project teams, as well as the review of previous iterations in order to maintain the architectural design as expected.

2. Checking Architectural Backlog. This activity refers to the strategic checking of the Architectural Backlog, which can be used as information source to support architectural activities.

3. Selecting ASRs to Design. This activity is about the decision and establishment of what must be designed in each synthesis iteration based on the set of ASRs to be handled. Several factors can influence the decision about the how many ASRs will be included, such as the diversity of application domains, the architects expertise in these domains, and the teams size.

4. Conceiving Candidate Architectural Solutions (CASs). In this activity, a set of CASs is proposed to encompass the ASRs under design, thus meeting a set of ASRs and establishing a new architectural version to be further evaluated. The CASs can be created based on several different sources, such as a background on building similar architectures, frameworks, design checklists, domain decomposition, reference architectures, and architectural patterns [Bass et al. 2012].

5. Coordinating Distributed Synthesis. SoS development typically involves different organizations performing a collaborative, distributed development of the software architecture. This activity space must encompass the required support for such a collaborative work through heterogeneous teams. In this sense, it encompasses the management of the distributed work to be performed as expected, e.g., negotiation of authority levels and communication strategies.

6. Updating Architectural Backlog. This activity space encompasses registering and maintaining inter-dependencies, trade-offs, changes, traceability links, as well as any other relevant information regarding the performed synthesis.

7. Handling Evolution. Evolvability is the ability to easily accommodate future changes. This attribute is highly required in SoS scenarios since the architecture is constantly evolving. In this context, this activity determines the investigation and establishment of strategies to maintain the SoS evolvability. Since CASs must be established by considering the evolvability concern, this activity dialogues with the *Conceiving CASs* influencing in the proposed architectural version.

8. Predicting Emergent Behaviors: Emergent behaviors of an SoS are not simply a sum of parts (i.e., constituent systems and their capabilities) and an effort must be di-

rected trying to predict it also considering the identification of both desired and undesired behaviors. Despite the management of these behaviors includes other phases of architectural construction (i.e., architectural analysis and evaluation), it is possible to conduct efforts in the architectural synthesis on how the proposed CASs can promote desired behaviors and also the analysis of how behaviors from previous iterations interfere in these CASs.

9. Building Architectural Documentation. The proposed CASs must be reflected in assets that allow the further understanding and evaluation of what was proposed. In this activity, the SoS software architecture is described according to the development context of each SoS, thus considering different interests, viewpoints, and particular environments. Moreover, it must provide a documentation that reflects the current SoS software architecture and also a way to convey the SoS software architecture that reaches all interested stakeholders and developers.

4. Evaluation

Aiming to assess the feasibility of applying SASI to generate method instances for SoS in any application domain, we conducted a study with six graduate students from the University of São Paulo during the Fall 2015 semester. A secondary goal was to receive feedback on the format and contents of the SASI description, which was used to make enhancements in SASI. In this context, the study has concentrated on the identification of SASI elements to conceive a method instance for a particular SoS context and the ability of understanding such elements on any SoS application domain.

The materials used during this study consisted of (i) the initial version of SASI description in the Essence Language, (ii) descriptions of two SoS inn different application domains, and (iii) a form to be filled by the subjects. Participants first received a training on SASI and then they were grouped into two teams. Each team received a different system description and each subject tried to compose a method instance by identifying which SASI elements (activities, alphas and work products) would be required to establish a method for architectural synthesis considering the provided SoS description. Moreover, the subjects had to justify why not included a given activity and to point out the alpha states before and after the architectural synthesis.

In order to verify if the method instances were as expected, they were compared to templates including all the adequate SASI elements on each SoS description. Figure 2 summarizes results obtained with the observational study. The quantitative data from this study showed some positive results, we observed that the instances generated by the subjects achieved a high degree of conformance (i.e., all evaluation averages reached at least 65% of conformance) to the templates and the conformance reached by the two teams was similar. In this context, we concluded that it was possible to generate instances of architectural synthesis methods on specific development contexts by using SASI, in spite of its generality regarding application domains.

Qualitative data also provided us with some lessons to enhance SASI. Subjects were questioned about their personal usage experience with SASI and asked for providing additional comments and enhancement suggestions. The main suggestions provided by the experts and further incorporated into SASI were, (i) more details about the relationships among activities and (ii) offering a template for the Architectural Backlog and

ASRs documentation as means of enhancing the guidelines concerning which information/asset should be registered/updated. In general, we consider that the results of this study represent a good indicative that SASI can be an adequate, comprehensive method for supporting architectural synthesis in acknowledged SoS software architectures.

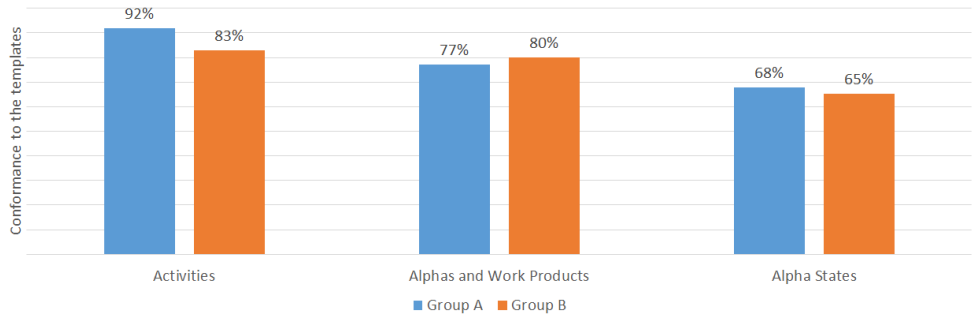

Figure 2. Conformace with templates.

4.1. Threats to Validity

The conducted study and its results may have been affected by some threats to empirical validity. Following, we briefly discuss some of these limitations.

Internal Validity. Internal validity is mainly concerned with unknown factors that may influence the results. To increase the validity of our study regarding this concern, we carefully designed, piloted, and iteratively refined the form and documentation provided to the subjects. Additionally, we made the participation voluntary and anonymous.

External Validity. External validity is related to claims for the generality of the presented results. In this context, we believe that the number of participants can be accepted since our main goal was to observe the results of using SASI and gain insights and suggestions for improving it.

Construct Validity. Construct validity focuses on the correct interpretation and measurement of the perceptions, i.e., the relationship between concepts and theories behind the study and what is actually measured and affected. We attempted to mitigate most of bias coming from the subjects by structuring a significant part of the form to drive the use of SASI in a particular SoS context. Furthermore, since enhancement suggestions could yield different interpretations, answers to these questions and consequent improvements were discussed among the researchers.

5. Conclusions

Despite the growing presence of SoS in several application domains and societal sectors, there are no standard processes or consensual practices regarding the construction of SoS software architectures. For this reason, important investigations still remain not addressed, such as the architectural synthesis on SoS software architectures. To fulfill this gap, the main contribution of this work is to propose a method to support this activity in acknowledged SoS context. We evaluated the feasibility of SASI in an study which shown good results on generating specific method instances with SASI.

Taking in account the relevance of adequate synthesis for architectural construction in SoS scenarios, future work will encompass a deeper investigation on the application of SASI in industry. In this context, we also intend to investigate and propose

methods for the other major activities of architectural design, i.e., architectural analysis and architectural evaluation.

References

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in practice*. Addison-Wesley, USA, 3rd edition.

Boardman, J. and Sauser, B. (2006). System of systems – The meaning of *of*. In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Piscataway, NJ, USA. IEEE.

Boehm, B. and Lane, J. (2006). 21st century processes for acquiring 21st century software-intensive systems of systems. *Journal of Defense Software Engineering*, 19(5):4–9.

DoD (2008). *Systems Engineering Guide for Systems of Systems*. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, Washington, DC, USA. Version 1.0.

Gonçalves, M. B., Cavalcante, E., Batista, T., Oquendo, F., and Nakagawa, E. Y. (2014). Towards a conceptual model for software-intensive system-of-systems. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 1605–1610, Washington, DC, USA. IEEE Computer Society.

Gonçalves, M. B., Oquendo, F., and Nakagawa, E. Y. (2015). A meta-process to construct SoS software architectures. In *Proceedings of the 30th ACM/SIGAPP Symposium on Applied Computing*, pages 1411–1416, New York, NY, USA. ACM.

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.

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., and Lidman, S. (2013). *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley.

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