A Meta-Process to Construct Software Architectures for System of Systems

Marcelo Benites Gonçalves[1,2], Flavio Oquendo[2], Elisa Yumi Nakagawa[1]
[1]Dept. of Computer Systems, University of São Paulo - USP, São Carlos, SP, Brazil
[2]IRISA-UMR CNRS/Université de Bretagne-Sud, Vannes, France
{marcelob, elisa}@icmc.usp.br, flavio.oquendo@irisa.fr

ABSTRACT

Nowadays, complex software systems tend to be the result of operationally independent, constituent systems working together, arising a new class of software systems called Systems of Systems (SoS). In another perspective, software architectures are essential to promote the success and quality of software systems, even more on SoS. However, the construction of SoS software architectures is typically ad-hoc without well-defined and standardized architecting approaches. In this context, the main contribution of this paper is the proposal of a "Meta-process for SoS Software Architectures" (SOAR), which supports the authoring of processes to construct SoS software architectures. SOAR is also independent of application domains and it is based on a broad, deep literature review as well as knowledge of experts. In order to evaluate the feasibility of SOAR, we conducted a survey with experts in SoS software architecture. The results of this survey indicate a good acceptance of SOAR among experts that also provided insights for improving SOAR. Our intention is to use SOAR as a framework to support the authoring of architecting processes for SoS and, further, to provide specialized versions including architectural decisions for specific application domains. Therefore, in some extent, we hope to contribute to the development projects of the new, important class of SoS software systems.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*life cycle, software process models*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Management, Standardization

Keywords

SOAR, Architectural Process, Survey, Software Architecture, System of Systems, Architecting

SAC'15, April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695737.

1. INTRODUCTION

Software-intensive systems have increasingly become ubiquitous, larger, and complex with considerable dissemination in various sectors and application domains, such as the avionics, smart cities, healthcare, and Global Earth Observation. In several cases, these systems can be classified as System of Systems (SoS) since they have arisen from the integration of independent Constituent Systems (CSs). Adequately coordinated, these CSs cooperate to provide emergent functionalities, which cannot be provided by any CS working separately. In this perspective, SoS is at the forefront of complex systems, being considered as a solution for a number of complex software systems, in which CSs have been integrated in a large-scale, networked, and dynamic way.

In another perspective, software architectures have been considered the backbone for any successful software-intensive system [8] and have played a fundamental role in determining the system quality. In the system development, decisions made at the architectural level directly enable, facilitate, or interfere with the achievement of business goals as well as functional and quality requirements (e.g., interoperability, performance, portability, and maintainability). In this scenario, the main purpose of processes that support the construction of software architectures is to develop specifications for a software-based system. This is a non-trivial activity, even more for complex scenarios of SoS, which encompass several systems and organizations with different interests and development approaches.

The development of SoS differs from monolithic systems in several issues, such as the dynamic contribution and impact of CSs, which are developed and managed by independent sources [4]. In this context, several challenges emerge on architecting SoS, such as the integration of heterogeneous CSs and the determination of special communication protocols and integration rules [12]. SoS have reached a threshold in which traditional software engineering approaches are no longer applicable [2]. Therefore, despite the existence of processes and techniques for software architectures, new approaches must be investigated in order to encompass the construction of SoS software architectures.

In this paper, we introduce SOAR (Meta-process for SoS Software Architectures), conceived to support the development of SoS software architectures. Moreover, considering the different architectural perspectives encompassed by the different categories of SoS, SOAR was conceived to support *Acknowledged* SoS, which refers to a category in which goals, management, resources, and authority are recognized while

the CSs retain their independent management. In this context, changes in the SoS are dependent upon the negotiation between stakeholders and developers of SoS and CSs.

SOAR combines the Systems Engineering experiences on architecting SoS with Software Engineering knowledge on developing software architectures. Moreover, the establishment of SOAR was based on a broad, deep literature review as well as knowledge of experts in SoS software architectures. SOAR is represented by using the Essence approach[1] that comprises a language and a conceptual kernel to support process authoring on software engineering. Both development and evaluation of SOAR were supported by different groups of experts that provided insights and improvements for SOAR.

The remainder of this paper is organized as follows. Section 2 introduces the issues surrounding the construction of SoS software architectures. Section 3 presents SOAR. Section 4 presents the evaluation concerning our proposal. Finally, Section 5 presents our conclusions and future work.

2. BACKGROUND

SoS is a system that results from other operationally independent systems working together to reach common goals. The most consensual SoS categories are [12]: (1) *Virtual*, in which there is no central management authority and the mechanisms to maintain the whole SoS are not evident; (2) *Collaborative*, in which CSs voluntarily collaborate in order to address common interests and a central authority offers standards to enable the collaboration of the CSs; (3) *Acknowledged*, in which SoS goals, management, resources, and authority are all recognized and changes in the SoS depend on the negotiation between the SoS and its CSs; and (4) *Directed*, in which there is a central authority and CSs are subordinated to the central control and its purposes.

In another context, software architecture represents the structure (or a set of structures) of the system, which comprises software elements, the externally visible properties of these elements, and the relationships among them. Regarding its construction, Hofmeister et al. [6] presented a comparison among five main processes, and highlighted similarities and differences among them. As a result, the authors have proposed a meta-process that comprises the main elements expected to any software architecture development process. This meta-process has been adopted as a relevant reference when instantiating new processes for software architectures.

Particularly for SoS software architectures, the SoS Guide of the American Departament of Defense [12] is a relevant work that reflects the experience of government, industry, and academy about the development of acknowledged SoS. This guide establishes that the SoS architecture is a core element and persistent framework to support the evolution of SoS over time. Other examples of similar approaches include design frameworks [5], process model [3], and architectural simulation and validation techniques [9]. Although these studies on systems engineering give important directions in developing SoS, they do not properly encompass software architectures.

Furthermore, a Systematic Literature Review (SLR) presented in [11] identifies the architecturally relevant features of SoSs and how their software architectures have been represented, evaluated, constructed, and evolved. In the same context, there are other studies that discuss the applicability of already existing architectural solutions, such as Architectural Description Languages (ADLs) [1], and Service Oriented Architecture (SOA) [10]. Although these studies point out the relevance of an adequate development of SoS software architectures, they do not comprehensively address what must be done to construct these architectures. In this context, we have not found any specific approach, expressing common challenges and solutions to support the construction of SoS software architectures.

[1] `http://www.omg.org/spec/Essence/1.0/Beta2/`

3. THE SOAR META-PROCESS

Due to the recent challenges brought by the construction of SoS software architectures, it is important to understand what is expected from a process for constructing them. In this scenario, we argue for a set of requirements that expresses what any process should satisfy in order to adequately support the development of SoS software architectures. Table 1 presents this set of requirements based on the construction challenges identified in a previous SLR [11]. By analyzing the current approaches for software architectures in the light of this set of requirements, we have not found a process capable to adequately encompass the construction of SoS software architectures. In this context, SOAR was conceived to be adherent to these requirements. Therefore, software architects, process managers, and other SoS stakeholders can have a support to instantiate their own processes when constructing SoS software architectures.

Furthermore, aiming at providing an useful, well-defined meta-process, SOAR was established over the OMG's Essence Approach, which comprises the Essence Language and the Essence Kernel. The Essence language provides a common mean for authoring processes from different perspectives (e.g., meta-processes, practices, and instances of process). Furthermore, since the Essence language allows an incremental evolution of processes, development teams can experiment and evolve their way of working to be as adequate as possible in the context of each project. Therewith, Essence language was chosen because it is compatible with the evolutionary development required by SoS.

In the Essence Language, the process authoring can be performed by using kernels and practices: kernels provide conceptual grounding (i.e., the "what must be done"), and practices provide the specific activities, work products, and workflow to be performed (i.e., the "how to do"). In this perspective, the Essence Kernel itself was conceived to provide a generic common basis for defining software development practices and processes. The SOAR meta-process is adherent to the basic concepts already established in the Essence Kernel.

Due to the complex, multidisciplinary nature of SoS and its software architecture, a modular strategy was adopted when conceiving SOAR. The proposed meta-process was organized in different levels: (i) the software development, already encompassed in the *Essence Kernel*; (ii) the development of software architectures, encompassed in the *Software Architecture (SA) Kernel*; and (iii) the development of SoS software architectures, encompassed in the *SoS SA Kernel*.

The SA Kernel provides general grounding on the construction of software architectures. It includes general elements to support the construction of software architectures independently from a system class (e.g., SoS), application domain, and specific technologies. The SA Kernel is an adap-

Table 1: Requirements to the Construction of SoS Software Architectures

SoS Characteristics	SoS Architecting Process Requirements
Operational Independence: each CS can deliver its own functionalities when not working in the SoS environment	**1.** Deal with the individual self-regulation capability of each CS **2.** Allow for feedback from SoS operations
Managerial Independence: each CS can keep its own managerial sphere	**3.** Support the integration of self-managed systems being consistent with processes and interests of individual systems
Geographical Distribution: CSs of an SoS are physically decoupled, thus only exchanging information among them	**4.** Allow for geographically dispersed system interaction **5.** Allow for connectivity support for heterogeneous CSs
Emergent Behavior: Behavior resulted from the collaboration of the CSs and that cannot be provided by any of these systems if they work as individual entities	**6.** Provide the composition of SoS capabilities in a composition scheme of emergent behaviors **7.** Validate SoS capabilities **8.** Allow for prediction of desired emergent behaviors **9.** Allocate emergent behavior requirements to CSs **10.** Continually analyze and assess SoS capabilities
Evolutionary Development: an SoS as a whole may evolve over time to respond to changes in its environment, of the CSs or of its own mission	**11.** Allow for incremental and evolutionary system deployment **12.** Revise system functionality in response to SoS operations **13.** The set of design decisions must be an explicit part of the architecture (design decision model) **14.** Develop and continually refine a SoS design decisions **15.** Develop and continually refine SoS scenarios **16.** Provide a dynamic integration strategy

tation of the meta-process proposed by Hofmeister et al. [6], which depicts the essential activities and artifacts required to any construction process for software architectures, and it is represented by using the Essence Language.

The SoS SA Kernel is the core of SOAR meta-process, being conceived to provide an adequate grounding on the construction of acknowledged SoS software architectures. This kernel extends the basic elements for constructing software architectures provided by the SA Kernel, coping with the challenges of SoS context. Moreover, it is not attached to any specific application domain or supporting technology, such as architectural style or architectural pattern.

Figure 1 shows the main elements of the SoS SA Kernel and their basic interrelationships organized in three main *areas of concern*, as proposed by the Essence Kernel: *customer*, *solution*, and *endeavor*. In the customer area, the context of the SoS must be understood with the adequate exploration of the opportunities that can be addressed by the software architecture, involving multiple stakeholders of the SoS. In the solution area, the team has to ensure the most adequate architectural solutions for the SoS in an evolutionary perspective based on emergent behaviors. Finally, in the endeavor area, an adequate coordinated/distributed development must be established and maintained, involving different development teams, stakeholders, and organizations. Since these elements are the essential ones for constructing any SoS software architecture, generic elements of SOAR provided by the Essence Kernel or the SA Kernel were omitted in this paper for sake of simplicity. Next subsections present the main elements of the SoS SA Kernel.

3.1 Alphas of SoS SA Kernel

In the Essence Language, *alphas* determine the "things to work with". The alphas of the SoS SA Kernel are:

Spheres of Interests: A group of stakeholders and organizations expressing common interests that typically influence the SoS context and related requirements. Spheres of Interests are specially relevant to SoS since they encompass one or more stakeholders and decision-making authorities, thus expressing influences in various aspects of the SoS.

SoS Architecturally Significant Concerns (SoS ASCs): Under the assumption that the general aspects of SoS (mission, goals, and whole SoS requirements) were already provided by general system engineering processes, SoS ASCs represent a set of specific interests pertaining to the development of the SoS software architecture that is derived from all available information about the general context of the SoS. These interests are related to development, operation, or any other important aspect in the architectural context (e.g., requirements from any sphere of interest, architectural patterns, or any previously agreed design decisions).

SoS Architecturally Significant Requirements (SoS ASRs): An ASR is any functional or non-functional requirement that is relevant for software architecture. Based on ASCs, SoS ASRs are obtained after analysis, negotiation, and decomposition. These ASCs are agreed through different spheres of interests to be further handled by architectural solutions.

SoS Software Architecture: It is a software structure (or a set of structures) of the SoS, which comprises CSs, their externally visible properties, and the relationships among them. A SoS software architecture encompasses concepts, properties, specifications, design principles, and design decisions and patterns of SoS and its environment.

Constituent System (CS): Operationally and managerially independent individual system that contributes to the accomplishment of the SoS mission. Each CS has its own architecture that typically is not visible, or accessible to changes.

Acknowledged System of Systems (SoS): A complex system resulted from the integration of other independent and heterogeneous systems. The collaborative work of CSs yields emergent functionalities in order to accomplish the SoS goals/mission.

Emergent Behaviors: The result of the collaborative

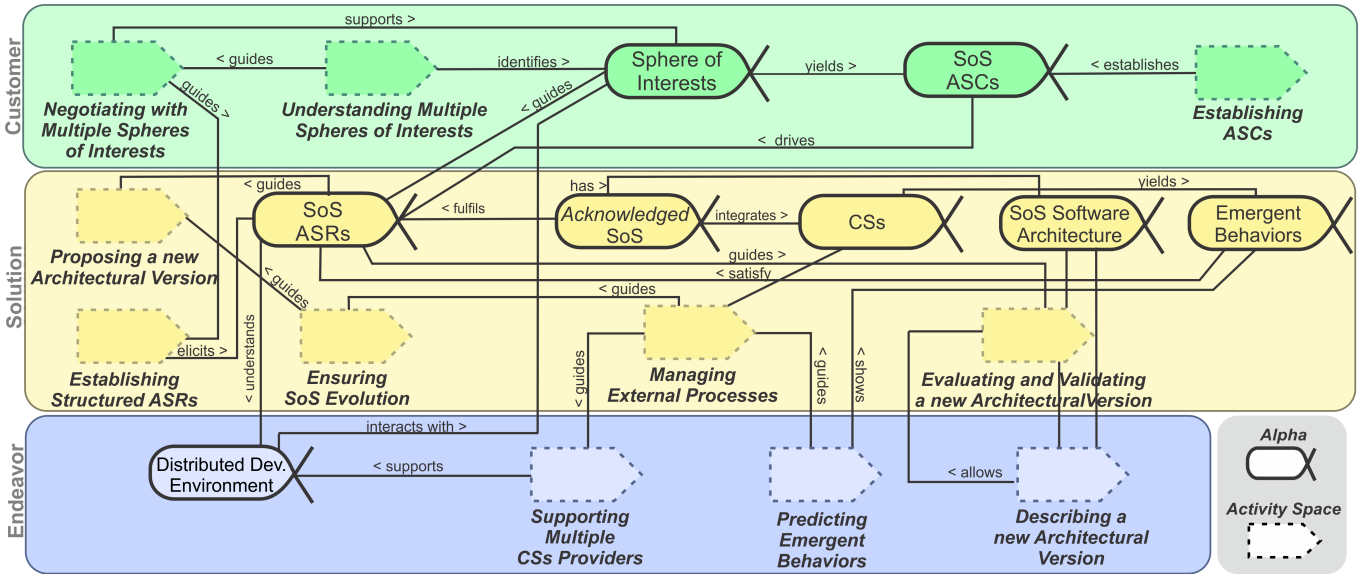

Figure 1: Alphas and Activity Spaces of the SoS SA Kernel.

work of CSs. Four basic types of emergent behaviors can be considered [7]: (i) predicted/desired; (ii) predicted/undesired; (iii) unpredictable/desired and; (iv) unpredictable/undesired. In general, predicted/desired behaviors come from architectural solutions and unpredict-able/undesired ones must be reduced as much as possible.

***Distributed Development Environment*:** This alpha expresses the complex development environment surrounding SoS, in which several distinct teams can collaborate to the SoS architectural development and evolution.

The presented alphas express the main concerns when constructing SoS software architectures. In an instantiated process, process authors must establish the most adequate work products in order to allow these alphas in the specific context of each development project.

3.2 Activity Spaces of the SoS SA Kernel

In the Essence Language, *activity spaces* determine "what must be done". Figure 2 shows a set of activity spaces provided by the SoS SA Kernel organized in a high-level workflow. This structure must be performed in each SoS development cycle and followed when instantiating activities and workflow of specific architectural projects. Therefore, activity spaces can be incrementally filled with activities, as SoS evolves becoming more complex. These activity spaces are:

Understanding Multiple Spheres of Interests: In the SoS development, there is a high amount of uncertainty that surrounds such a system, not only in a technical sense, but also in organizational and business senses. In this context, this activity space must have activities to understand, identify, and represent Spheres of Interests and their relations.

Establishing SoS ASCs: SoS ASCs must be structured/organized in order to express SoS concerns of different Spheres of Interests. This activity space encompasses understanding and establishing SoS ASCs.

Negotiating with Multiple Spheres of Interests: SoS inherently have a more complex community in which decision authorities can naturally have a more self-serving perspectives of participation. In this context, this activity space

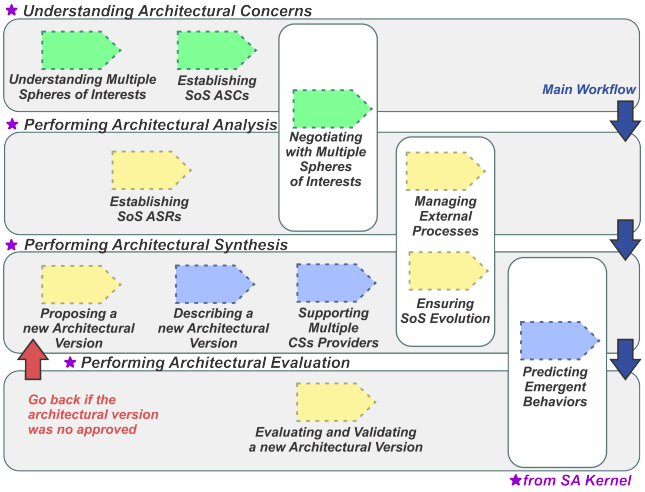

Figure 2: Activity Spaces of the SoS SA Kernel.

aims at promoting a collaborative environment in which multiple Spheres of Interests can negotiate SoS ASCs, SoS ASRs, and changes in the SoS.

Establishing SoS ASRs: The main purpose of this activity space is to define the problems that SoS software architecture must solve in terms of ASRs. For this, the examination of ASCs and negotiation with Spheres of Interests in order to come up with SoS ASRs must be also considered. By following a top-down approach, SoS ASCs must be structured/decomposed in order to meet SoS capabilities. Furthermore, the software architecture can itself be a source of requirements. In this sense, new requirements resulted from the software architecture must be also identified.

Ensuring SoS Evolution: SoS are inherently evolutionary and their architectures must be developed considering this perspective. This activity space must provide strategies to promote evolutionary development by supporting/improving other activities and alphas in the architectural anal-

ysis/synthesis.

Managing External Processes: In an SoS, the software typically interacts with other processes (e.g., physical and human) that influence its development. This activity space encompasses the understanding and management of how these processes influence the SoS software architecture context in order to keep the coherence through different SoS architectural perspectives. An example of strategy, is the definition of common taxonomies and ontologies in order to facilitate common understanding and improved communication among these different processes.

Supporting Multiple CSs Providers: This activity space support Architectural Teams to work with different CS providers. These providers can have different interests and their CSs can be in different stages of development. Therefore, support activities must be instantiated to promote a collaborative environment that enables the management of capabilities offered by CSs in the SoS context.

Supporting Distributed Architectural Development: SoS development encompasses different organizations, performing a collaborative, distributed development of the SoS software architecture. In this context, this activity space must encompass the required planning/support for this collaborative work through heterogeneous teams.

Proposing a new Architectural Version: In this activity space, candidate architectural solutions are proposed based on SoS ASCs and CS capabilities in order to meet a set of SoS ASRs. For this, a bottom-up approach must be established including the understanding of CS capabilities, operational constraints, communication protocols, and the set of emergent behaviors that can be produced by CSs working together in order to meet the SoS ASRs. The main result of this activity is the establishment of a new architectural version to be further evaluated.

Describing a new Architectural Version: In this activity space, the SoS software architecture is described according to the development context of each SoS. Different formalism levels can be considered (i.e., informal, semi-formal, or formal) covering different viewpoints (e.g., structural and behavioral). At more abstract architectural levels (e.g., systems engineering level), different aspects of SoS must be included in the representation when appropriate (i.e., software, hardware, and human). At more specific levels (i.e., the SOAR software engineering level), representation must focus on software while maintaining the compatibility with more abstract architectural levels.

Predicting Emergent Behaviors: Emergent behaviors of an SoS are not simply a sum of parts (i.e., CSs and their capabilities) and they must be predicted also considering the identification of both desired and undesired behaviors. A strongly recommended strategy is the use of executable models and architectural simulations for analyzing and predicting such behaviors. Moreover, for architectural simulations, representations with support to dynamism must be considered in the *Describing a new Architectural Version* activity space since it can help in dynamic analysis of how CSs interact with each other yielding emergent behaviors.

Evaluating and Validating an Architectural Version: The purpose of this activity space is to verify if the architectural design decisions are the right ones. The SoS software architecture is also verified against the ASRs, ASCs, and any other relevant element in the SoS context. Although multiple iterations are expected, the result is the validated architecture. If not validated, the activity spaces of architectural synthesis (see Figure 2) must be performed again.

The presented activity spaces express the main groups of activities when constructing SoS software architectures. In an instantiated process, process authors must establish the most adequate activities and more specific workflows in order to execute these activity spaces on each development project.

3.3 Uses of SOAR

SOAR can be used as a basis for specific process instances in which alphas and activity spaces can be encompassed by different elements (e.g., activities and their workflow, work products, and roles in architectural teams) in order to meet the specific needs of each SoS under development. Therefore, initially hidden details can later be elicited by each individual SoS for specific problem contexts. Furthermore, consensual solutions for specific contexts (e.g., application domains or development environments) can be included into SOAR as good practices or extended versions.

The Essence approach offers the *EssWork Practice Workbench*[2], a development environment for kernels, practices, and instance processes. This tool provides an easy, intuitive way to develop and deploy customized practices and process based on kernels. Although the use of EssWork Practice Workbench is not mandatory, the use of it is quite recommended when adopting SOAR.

4. EVALUATION

In order to evaluate the relevance of SOAR, two teams of SoS software architecture experts were consulted. The first one was composed by three experts, which provided improvements to our proposal. Afterwards, a qualitative survey was conducted with a second team formed by six experts external to our research group. These experts have been involved to the development of SoS and software architecture in both academy and industry. The survey aimed to verify if the established requirements (see Table 1) were adequate, and if SOAR meets the expectations of the SoS community as a meta-process to support the construction of SoS software architectures[3]. The chosen approach for gathering data was the use of online questionnaires. Each respondent was asked about a complete description of SOAR (including support material about Essence approach) in terms of convenience for use, completeness, correctness, coherence, intelligibility, and usability.

Table 2 summarizes the results obtained with answers of the non-discursive part of the questionnaire. In this part, the main perspectives of evaluation (EP column) were accomplished by one or more non discursive questions (RQ column). For each EP, the column "Yes" shows the percentage of experts that were favorable to SOAR in their answers, and the column "No" shows the unfavorable ones. The collected answers showed us that SOAR and the requirements are clear, their elements are described without ambiguities (80% of positive feedback for coherence), and the support material provided with SOAR is enough to enable the evaluation team to understand all of the technical terms (75%

[2]`http://www.ivarjacobson.com/EssWork_Practice_Workbench/`

[3]More information about this survey is available at `http://goo.gl/i5qYYT`

of positive feedback for usability). Results also pointed out that the process requirements have a broad coverage, encompassing all important challenges that play a role in the construction of SoS software architectures.

Table 2: Survey results of non-discursive questions

Evaluation Perspectives (EP)	RQ	Yes(%)	No(%)
Completeness: Is SOAR complete for what it is proposed?	3	73.3	27.6
Correctness: Is SOAR correct for with no wrong or misunderstood statements?	1	100.0	0.0
Coherence: Is SOAR conceptually coherent with no relevant conflicts or wrong placed elements?	1	80.0	20.0
Intelligibility: Is SOAR intelligible to its audience?	2	70.0	30.0
Usability: Can SOAR be considered well-organized, concise, helpful, and easy to use?	4	75.0	25.0

Questions with discursive answers also provided relevant insights to the improvement of SOAR and clearing how the evaluation perspectives with lower acceptance percentages could be improved. The experts confirmed that the requirements are adequate and that SOAR is adherent to such requirements. Another mentioned benefit was the flexibility brought by the independence of application domains. The simplicity and incremental perspective of SOAR and the Essence Language (in which the process can be incremented gaining complexity as SoS evolves) were also highlighted.

Regarding the lacks, downsides, and disadvantages on the usage of SOAR, the experts pointed that different types of SoS emergent behaviors were not initially covered. This issue was solved by extending the prediction of emergent behaviors to architectural synthesis and architectural evaluation phases of the SoS SA Kernel (see Figure 2). Another mentioned point was the difficulty to understand the relationship among different kernels (i.e., Essence Kernel, SA Kernel, and SoS SA Kernel) and how the instantiated process can be executed in the context of SoS development. These problems were handled in the complete documentation of SOAR with additional explanations, diagrams, and guidelines for use. In general, the evaluation results have regarded SOAR as an adequate meta-process for acknowledged SoS software architectures and the established process requirements as representative for what is expected to any construction process of SoS software architectures.

5. CONCLUSION

SOAR is a comprehensive framework that can support the construction of SoS software architectures. It was the result from an analysis of the state of the art of SoS in conjunction with lessons learned with collaborating experts. Initially focused on acknowledged SoS, SOAR can be valuable for several application domains. In this context, with the maturation of new good practices as standard solutions for SoS, new architectural decisions can be incorporated to SOAR, yielding new versions for more specific contexts.

As future work, we envision the development of specialized versions of SOAR focused on specific application domains. In this sense, it will be possible to build a family of specialized processes, each one presenting sets of design decisions for specific application domains. Moreover, we intend to develop additional elements to improve the architectural support of SOAR, such as a consensual quality model providing an essential set of quality attributes and guidelines of how they must be handled in the SoS context. Another future work is to enhance the EssWork Practice Workbench tool in order to facilitate the use of SOAR. Finally, SOAR will be extended in order to encompass other SoS categories (i.e., virtual, collaborative, and directed) within a family of meta-processes for the large, complex context of SoS.

6. REFERENCES

[1] T. Batista. Challenges for sos architecture description. In *SESoS*, pages 35–37, Montpellier, France, 2013.

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

[3] A. Chigani and O. Balci. The process of architecting for software/system engineering. *Int. Journal of System of Systems Engineering*, 3(1):1–23, 2012.

[4] C. Dagli, N. Ergin, A. Enke, D. Gosavi, R. Qin, J. Colombi, K. Rebovich, R. Giammarco, P. Acheson, K. Haris, and L. Pape. An advanced computational approach to system of systems analysis & architecting using agent-based behavioral model. Technical Report 021-2, SERC, 2013.

[5] D. A. DeLaurentis. Appropriate modeling and analysis for systems of systems: Case study synopses using a taxonomy. In *SoSE*, pages 1–6, USA, 2008. IEEE.

[6] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran, and P. America. Generalizing a model of software architecture design from five industrial approaches. In *WICSA*, pages 77–88, Pittsburgh, PA, USA, 2005.

[7] O. T. Holland. Taxonomy for the modeling and simulation of emergent behavior systems. In *SpringSim*, pages 28–35, San Diego, CA, USA, 2007.

[8] P. Kruchten, H. Obbink, and J. Stafford. The past, present, and future for software architecture. *IEEE Software*, 23(2):22–30, 2006.

[9] J. Michael, R. Riehle, and M.-T. Shing. The verification and validation of software architecture for systems of systems. In *SoSE*, pages 1–6, Albuquerque, NM, USA, 2009.

[10] S. Mittal and J. Risco Martin. Model-driven systems engineering for netcentric system of systems with devs unified process. In *WSC*, pages 1140–1151, Washington, DC, USA, 2013.

[11] E. Y. Nakagawa, M. Gonçalves, M. Guessi, L. B. R. Oliveira, and F. Oquendo. The state of the art and future perspectives in systems of systems software architectures. In *SESoS*, pages 13–20, Montpellier, France, 2013.

[12] A. ODUSD. *System Engineering Guide for Systems of Systems*. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, 2008. Version 1.0.