Supporting architectural design of acknowledged Software-intensive Systems- of-Systems

Marcelo Benites Gonçalves

THESE / UNIVERSITE DE BRETAGNE-SUD
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE-SUD

Mention :
Ecole doctorale: SICMA

Présentée par

Marcelo BENITES GONÇALVES

Préparée à l'unité mixte de recherche 6074
Institut de Recherche en Informatique et Systèmes Aléatoires
Université Bretagne Sud

Support à la conception architecturale de systèmes-de-systèmes reconnus à logiciel prépondérant

Thèse soutenue le 12 décembre 2016, devant le jury composé de :

M. Jair Cavalcanti LEITE
Professeur Titulaire, Université Fédérale du Rio Grande do Norte (UFRN), Natal, Brésil
/ Rapporteur

M. Yann POLLET
Professeur, Conservatoire National des Arts et Métiers (CNAM), Paris, France
/ Rapporteur

M. Khalil DRIRA
Directeur de Recherche CNRS, LAAS-CNRS, Toulouse, France
/ Examinateur

Mme. Cecília Mary Fischer RUBIRA
Professeur Titulaire, Université d'Etat de Campinas (UNICAMP), Campinas - SP, Brésil
/ Examinateur

Mme. Elisa Yumi NAKAGAWA
Maitre de Conférences HDR, Université de São Paulo, São Carlos, Brésil
/ Directrice de thèse

M. Flavio OQUENDO
Professeur des Universités, IRISA - Université Bretagne Sud, Vannes, France
/ Directeur de thèse

Marcelo Benites Gonçalves

Supporting architectural design of acknowledged SoS

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação - ICMC-USP and the Institut de Recherche en Informatique et Systèmes Aléatoires - IRISA, in partial fulfillment of the requirements for the degree of the Doctorate in Science and Docteur en Informatique in the agreement between the Graduate Program in Computer Science and Computational Mathematics (USP) and the Doctoral Program in Mathematics and Information and Communication Science and Technology (UBS). *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics / Mathematics and Information and Communication Science and Technology

Advisor: Profa. Dra. Elisa Yumi Nakagawa
(ICMC-USP, Brazil)
Advisor: Prof. Dr. Flavio Oquendo
(IRISA, France)

ICMC-USP, Brasil / IRISA, France
November 2016

Abstract

System-of-Systems (SoS) refer to complex, large-scale, and sometimes critical software-intensive systems that has raised as a promising class of systems in several application domains. In parallel, software architectures play a significant role in the development of software-intensive systems, dealing with both functional and non-functional requirements. In particular, systematic processes to design SoS software architectures can tackle challenges from SoS development, including to handle collaboration of independent constituent systems with different owners, missions, and interests. Despite the relevance and necessity of software-intensive SoS for diverse application domains, most of their software architectures have been still developed in an ad hoc manner. In general, there is a lack of structured processes for architecting SoS, hindering the secure adoption of SoS, reducing possibilities of sharing common architectural solutions, and negatively impacting in the success rate for these systems. This thesis presents SOAR ("General Process for Acknowledged SoS Software Architectures") that supports the establishment of architectural design processes for acknowledged SoS. Conceived to provide different levels of support according to each SoS development context, it comprises a high level kernel that describes what must be done when architecting SoS and also three practices with specific activities and work products to guide how to perform architectural analysis, synthesis, and evaluation. To evaluate SOAR, three surveys, a viability study, and an experiment were conducted. Results achieved in these evaluation studies indicate that SOAR can positively support the instantiation of architecting processes for acknowledged SoS and, as a consequence, contribute to the development and evolution of these complex, software-intensive systems.

Keywords: Acknowledged System-of-Systems, Software architecture, design process.

Résumé

Systèmes-de-systèmes (SoS) sont des systèmes à logiciel prépondérant de grande échelle, complexes et parfois critiques qui s'ont soulevés comme une classe de systèmes prometteuse dans plusieurs domaines d'application. En parallèle, architectures logicielles ont un rôle important dans le développement de systèmes à logiciel prépondérant, en traitant des exigences fonctionnelles et non fonctionnelles. En particulier, processus systématiques pour concevoir des architectures logicielles de SoS peuvent adresser les défis du développement de SoS, y compris la collaboration des systèmes constituants indépendants avec différents propriétaires, des missions et des intérêts. Malgré la relevance et la nécessité des SoS à logiciel prépondérant dans plusieurs domaines d'application, la plupart de leurs architectures logicielles sont encore développées de manière *ad hoc*. En général, il y a un manque de processus structurés pour concevoir architectures de SoS. Cette condition entrave l'adoption assurée de SoS, réduit les possibilités de partager solutions architecturales communes et a un impact négatif sur le taux de réussite pour ces systèmes. Cette thèse présente SOAR (General Process for Acknowledged SoS Software Architectures) qui soutient la mise en place des processus de conception architecturale pour SoS reconnus. Ce processus a été conçu pour fournir différents niveaux de soutien en fonction de chaque contexte de développement de SoS. Il comprend un noyau de haut niveau qui décrit ce qu'il faut faire pour la conception des architectures de SoS et ainsi que trois pratiques avec des activités spécifiques et des produits de travail pour guider l'analyse, synthèse et évaluation architecturale. Afin d'évaluer SOAR, trois enquêtes, une étude de viabilité et une expérimentation ont été menées. Les résultats obtenus dans ces trois études d'évaluation indiquent que SOAR peut soutenir positivement l'instanciation des processus de conception architecturale pour des SoS reconnus et par conséquent contribuer au développement et l'évolution de ces systèmes complexes à logiciel prépondérant.

Mots clés: Systèmes-de-systèmes, Architecture logicielle, Processus de conception.

Contents

List of Figures

List of Tables

Chapter

1

Introduction

With the advances of computational, network, and communication technologies, software-intensive systems have become increasing ubiquitous, larger, and complex, with considerable dissemination in various application domains and society sectors. In this scenario, System-of-Systems (SoS) arises as a class of systems with the potential of encompassing new development challenges brought by these advances. An SoS can be defined as a set or arrangement of independent systems, cooperating into a larger resulting system that delivers unique capabilities (DoD, 2008). Despite SoS concept has been used since the 1950s, only recently its popularity has grown with the current network and communications technologies that have made possible to conceive and maintain a structure of independent systems from different organizations producing emergent behaviors that are not possible to be delivered by any of these systems working separately (Nielsen et al., 2015). With this perspective of cooperation, SoS has gained space on several domains, such as the internet (Maier, 1998), global earth observation (Johnson, 2008), healthcare (Hata et al., 2007; Nielsen et al., 2015), emergence response (Nielsen et al., 2015), energy (Agusdinata and DeLaurentis, 2008; Nielsen et al., 2015), transportation (DeLaurentis, 2005a; Nielsen et al., 2015), airport control (Jamshidi, 2008b), automotive (Aoyama and Tanabe, 2011), avionics (Farcas et al., 2010), and robotics (Bowen and Sahin, 2010). SoS development demands new software engineering approaches capable to encompass the SoS challenges in software-intensive domains. In this context, the software architectures of SoS have been noticed as an important element to the success of such systems (Brondum and

Liming, 2010; Jamshidi, 2008b; Maier, 1998; Schaefer, 2005), and an important related issue is to promote the adequate design of these architectures.

Software architecture is a fundamental element to guide development (Bengtsson et al., 2004), being considered the backbone for any successful software-intensive system and determinant to its quality (Shaw and Clements, 2006). A software architecture represents the system structure including its software elements, their externally visible properties, and the relationships among them. It must act as a bridge between the business goals and the aimed system, providing a path from abstract customer needs to a final concrete system (Bass et al., 2012). Architectural decisions directly impact on the achievement of business goals as well as functional and quality requirements (Shaw and Clements, 2006). Therefore, software architecture is essential not only in the initial construction of the system but also in the further life cycle.

Due its relevance, the development of software architectures arises as a rich field of concepts, methods, standards, frameworks, tools and so on to create, document, analyze, and assess structural and behavioral specifications. In this sense, the use of systematic architecting processes has become indispensable in the systems development, facilitating the design and reducing development costs (Bass et al., 2012; Shaw and Clements, 2006). Several approaches are proposed to guide and support the conduction of these processes in traditional software engineering (Bass et al., 2012; Bosch, 2000; Clements et al., 2010; Dikel et al., 2001; Garland and Anthony, 2003). Despite many of these approaches were developed independently, a set of three macro-activities are commonly identified among them (Hofmeister et al., 2007): (i) architectural analysis, which comprises the identification of what problems in the system context software architecture can potentially solve; (ii) the architectural synthesis, in which candidate architectural solutions must be proposed to effectively solve previously identified problems; and (iii) architectural evaluation, which deals with evaluation of proposed solutions.

Architectural design processes are already considered an important research field in systems engineering and even more when considering complex scenarios of SoS (Brondum and Liming, 2010; Jamshidi, 2008b; Maier, 1998; Schaefer, 2005). SoS differ from monolithic systems in several characteristics, such as the dynamicity and complexity of their operational environments, boundaries, communication protocols, collaborating constituent systems, and emergent behaviors (Acheson et al., 2012; Dagli et al., 2013). Therefore, establishment and consolidation of systematic processes to overcome challenges and guide the architectural design of these systems can bring important advantages, such as the early identification of quality attributes, more comprehensive understanding of entire SoS and its emergent behaviors, and cost savings in enabling constituent systems participation (Kazman et al., 2012).

1.1 Problem Statement

The development scenarios of SoS call for a paradigm shift to develop these systems. SoS demand not only different models, views, and architectural levels, but also consistency among them while the architecture evolves (Dagli et al., 2013). Several studies can be found addressing SoS architectures and their construction from a Systems Engineering perspective (Dandashi and Hause, 2015; DoD, 2008; Jamshidi, 2008b; Klein and van Vliet, 2013; Schuitemaker et al., 2015). Despite these studies can give important directions to construct SoS software architectures, these studies still encompass the design by a Systems Engineering level, not encompassing specific software development issues.

In the Software Engineering area, the complexity of SoS reaches a threshold where customary approaches are no longer sufficiently reliable, naturally demanding for adaptations and proposition of new ones, as well as the formation of skilled engineers and computer scientists to enable their accomplishment (Boehm and Lane, 2006; Dvorak et al., 2005; Greaves et al., 2004). In terms of processes, SoS software architectures have been constructed in an ad-hoc way, in which each architectural project is conducted on a particular manner. In this context, current processes for software architectures do not consider the particular challenges of SoS development, opening a new field to explore particularities of software architectures development in SoS contexts.

The architectural processes in the SoS context must encompass several challenges, such as: (i) increased complexity of scope and costs for planning and engineering (DoD, 2008); (ii) management of governance issues across multiple organizational boundaries and stakeholders with competing interests and priorities (Dahmann et al., 2011; DoD, 2008); (iii) need to deals with emergent behaviors and likelihood of unpredictable ones (DoD, 2008); (iv) achievement and maintenance of constituents cooperation as expected, balancing individual characteristics, communication protocols, multiple lifecycles, and individual needs between constituent systems and SoS (Dagli et al., 2013; Dahmann et al., 2011; DoD, 2008); and (v) establishment of an evolutionary development through continuous changing scenarios (DoD, 2008). From this perspective, the complexity of SoS demands for heavyweight plan-driven architecting processes in order to ensure an adequate level of documentations, balance and alignment of architectural processes with other processes of SoS development, reach the broad agreement among stakeholders, ensure high predictability of behaviors, and manage constituents participation in the SoS context. Furthermore, architectural design processes can naturally exhibit domain characteristics and emphasize different goals according to each system domain (Hofmeister et al., 2007); the complexity of this process is increased in multidomain context, in which an SoS typically has constituents with own domains. Moreover, each category of SoS (i.e.,

virtual, collaborative, acknowledged, and directed) can also bring specific challenges for development (DoD, 2008).

1.2 Objectives and Research Questions

According to research gaps presented in previous section, the general research question to be investigated in this thesis is whether or not a general architecting process can support instantiation of specific architectural design processes for acknowledged SoS software architectures. Based on this general research question, the main objective of this thesis is to establish a general process, named SOAR ("General Process for Acknowledged SoS Software Architectures"), conceived to support the establishment of design processes[1] for acknowledged SoS software architectures. In this context, we focused our approach on acknowledged SoS, an SoS category in which goals, management, resources, and authority are all recognized at SoS level, while constituent systems retain their independence and changes depend on negotiation between owners of SoS and of their constituent systems. Furthermore, this general goal will be achieved through the satisfaction of following sub-goals:

SG1: To describe what must be encompassed in terms of process when designing acknowledged SoS software architectures;

SG2: To provide support to establish processes for architectural analysis of acknowledged SoS;

SG3: To provide support to establish processes for architectural synthesis of acknowledged SoS; and

SG4: To provide support to establish processes for architectural evaluation of acknowledged SoS.

With SOAR, we aim at providing prescriptive guidance for project teams to design acknowledged SoS software architectures, the further implementation and evolution of these systems.

1.3 Contributions

The SoS development process is a large set of processes that are multiple and interdependent in terms of work products (Perry, 1994; Sage and Biemer, 2007). In this context,

[1] In this thesis, we stipulate that a process can be formed by other processes.

SOAR was conceived to make it possible to create instances of processes in consonance with an specific SoS development process. For this, we established an interface with inputs and outputs that must be adequately dealt in the SoS development process. SOAR is also compatible with the general process for SoS development proposed in (DoD, 2008), which describes a process at systems engineering level that reflects the current state of the art in acknowledged SoS projects. Based on this process and on additional studies related to this process (Acheson et al., 2012; Dahmann et al., 2011), Figure 1.1 represents how SOAR can be used in the context of an iterative and evolutionary development cycle of acknowledged SoS. At the central strip, architecting processes are executed receiving the general context analysis, i.e., analysis performed at system engineering level considering different layers of SoS, i.e., physical, organizational, software, etc. SOAR is conceived to support the establishment of architecting processes at software layer and its activities are performed in this stage, delivering the respective software architecture. Despite interdependencies with other external processes of SoS development, software architecture is the central concern to be encompassed in SOAR instances.

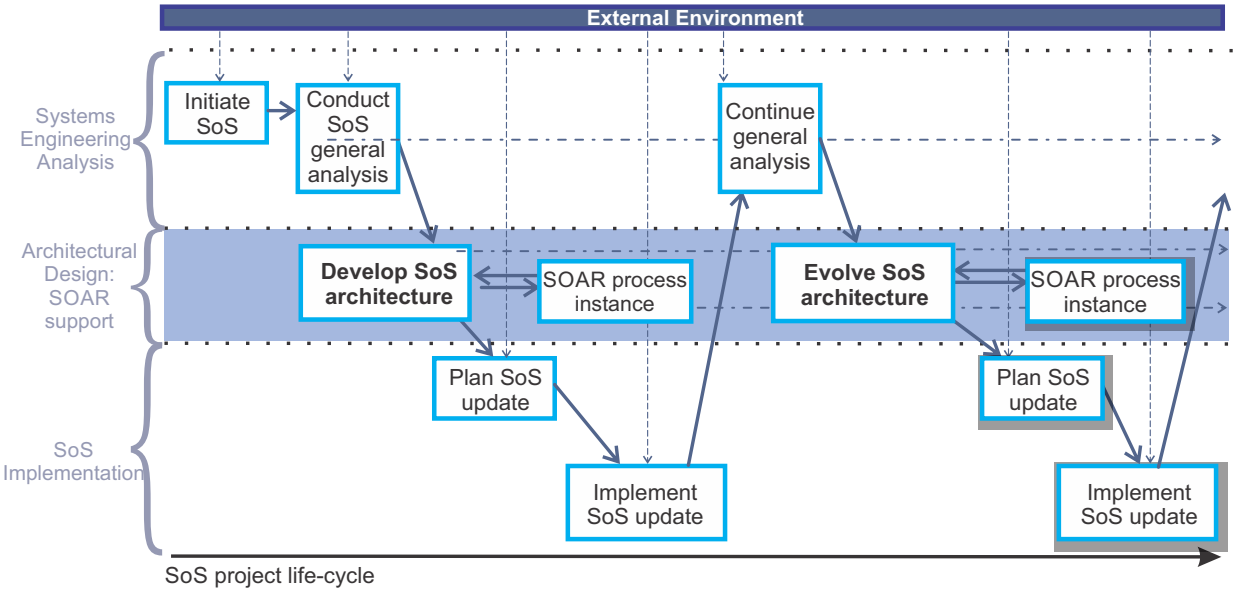


Figure 1.1: Context of application of SOAR (adapted from (Acheson et al., 2012))

SOAR was conceived to provide a well-defined documentation that includes activities, work products, and guidelines. For this, we adopted the OMG's Essence Standard that includes a process authoring language, called Essence Language, which is a flexible notation that make possible authoring of software engineering processes in different scopes and levels of specificity, and a general kernel for software development process, called Essence Kernel, which provides a basis of general concepts useful for any software engineering projects. Despite the existence of more mature languages for processes description, such as SPEM (Software and Systems Process Engineering Meta-Model Specification) (Ob-

ject Management Group (OMG), 2015a), this new standard was proposed to be flexible, easy to use, and adequate to evolutionary development projects, in which development processes can also evolve themselves maintaining their alignment with different stages of development.

Each SoS also has particular issues surrounding its development (DeLaurentis, 2008; Klein and van Vliet, 2013) and a general process like SOAR must have an adequate level of abstraction. Being too general can result in useless processes, since main SoS challenges could be not encompassed. Conversely, being too specific can generate very exclusive solutions with low exploration of SoS common challenges and low impact for reuse. Due to the complex, multidisciplinary nature of SoS and their software architectures, we adopted a modular strategy with two levels of abstraction to SOAR. The first level is suited for experienced teams, which must only use a kernel to check if their already well-established processes adequately encompass the design of acknowledged SoS software architectures and if improvements can be planned. The second abstraction level provides practices with more detailed guidance for project teams not experimented in design process of SoS software architectures. Furthermore, SOAR is also independent of specific architectural styles and application domains, underlying an affordable balance between generality and specificity. Proposition of different abstraction levels was possible by using two main elements of Essence Language: Kernels and Practices. Kernels are elements that describe "what must be done" in processes. Given a pre-determined scope, a kernel must provide a common ground of concepts and goals for process authors. Practices describe the "how to do" in processes. In general, they are grounded by kernels and deliver specific solutions in terms of activities and work products to support process authoring. In each individual project, the a process instance can be based on a composition of practices that are convenient to solve project needs.

To conceive SOAR, several sources of knowledge are combined in the light of the new challenges brought by SoS. These sources are: (i) knowledge from traditional software engineering that has mature approaches for software development; (ii) recent approaches on software engineering specifically proposed for SoS; (iii) knowledge from systems engineering that is the area with more mature approaches focused on SoS; and (iv) personal experience of experts who contributed in the conception and evaluation of our proposal. Figure 1.1 shows SOAR[2] organization in terms of its main elements and basic workflow. Since the complexity of design processes does not allow establishment of a mandatory workflow with a regular sequence of execution (Hofmeister et al., 2007), the presented workflow is only a reference of execution and project teams can plan and adopt a work-

[2]A complete description of SOAR is available at: `http://www.start.icmc.usp.br/html/SOAR/`

flow in each development context and evolution stage. SOAR kernel and practices are briefly introduced as follows:

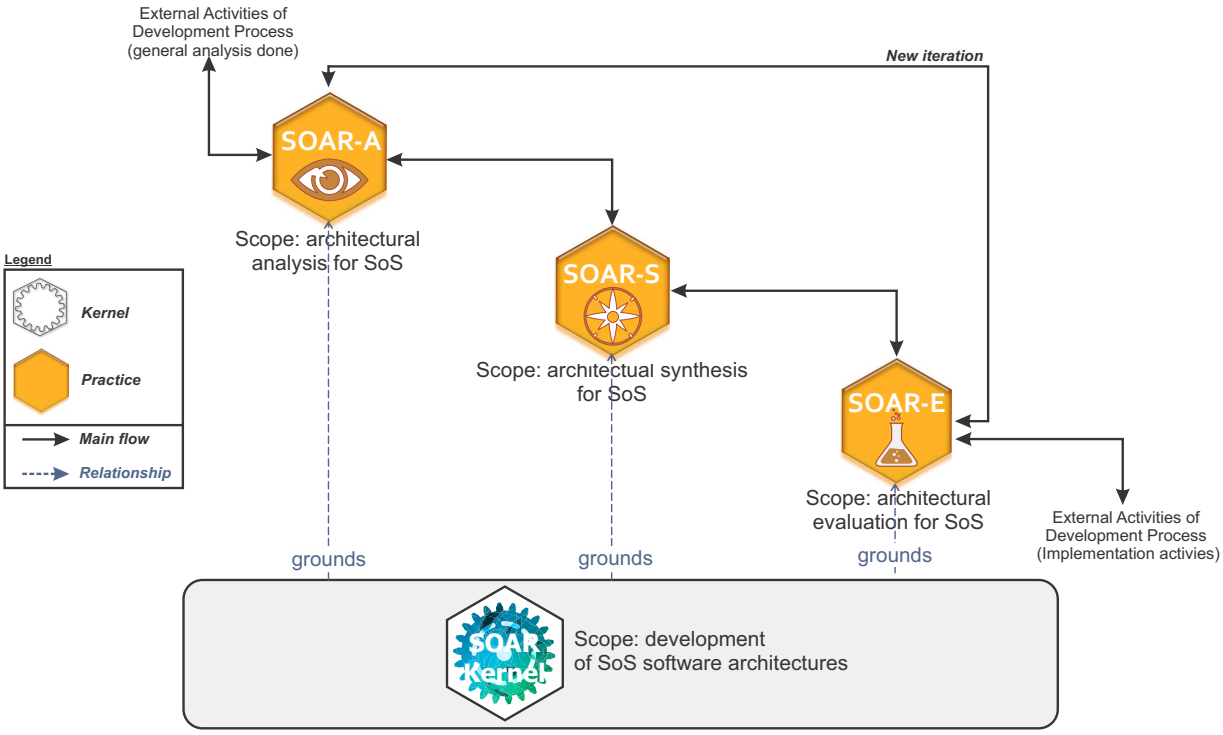


Figure 1.2: SOAR overview

- SOAR Kernel is the SOAR element at highest level of abstraction and was conceived to provide a common ground to the construction of acknowledged SoS software architectures, including common vocabulary, concepts, and general guidelines.

- SOAR-A is a SOAR practice that describes a set of activities and related work products recommended to architectural analysis of acknowledged SoS. The main output of this practice is a set of architectural requirements to further support the architectural design activities.

- SOAR-S is a SOAR practice that describes a set of activities and related work products recommended to architectural synthesis of acknowledged SoS. A central concern in SoS development is the definition of an SoS architecture capable of expressing the structures of SoS with its constituent systems, including their relationships, connections, interactions, and parameters (Sanduka and Obermaisser, 2014). In architectural synthesis, fundamental design decisions are made and the main output is a candidate architecture to meet the SoS architectural requirements.

- SOAR-E is a SOAR practice that describes a set of activities and work products to encompass the architectural evaluation on acknowledged SoS. The main output of this practice is an evaluated architecture and respective evaluation reports.

Considering the objectives and an overview of the contributions this thesis, next section describes the context of its chapters.

1.4 Thesis Outline

Chapter 2 presents the state of the art of SoS software architectures. Initially, terminology and key concepts related to SoS are presented including a general model proposed to support characterization of SoS and published in (Gonçalves et al., 2014). After that, theory associated with SoS software architectures is presented, including results of a Systematic Mapping(SM) that explored this topic. Finally, results of a Systematic Literature Review (SLR), which leveraged the state of the art of processes for SoS software architectures, is presented.

Chapter 3 describes SOAR Kernel and its main elements. In this thesis, we present an enhanced/updated version of this Kernel initially published in (Gonçalves et al., 2015). We also conducted a survey with experts to evaluate SOAR Kernel. Results indicate that SOAR Kernel can be used to its grounding purposes of being a general basis for supporting the development of acknowledged SoS software architectures and facilitating the improvement of development processes of these systems.

Chapter 4 describes SOAR-A, the SOAR practice for architectural analysis. We conducted a survey with experts to evaluate if SOAR-A is adequate to guide architectural analysis in acknowledged SoS projects. Obtained results show a good acceptance of this practice indicating confidence for its use.

Chapter 5 describes SOAR-S, the SOAR practice for architectural synthesis. For this practice, we conducted two evaluation studies with different evaluation goals. The first one explored the capability of generating process instances from SOAR representation. Obtained results were useful as a proof of concept, providing more confidence to further conduct the second study, which was an experiment focused on confronting process instances generated with SOAR-S against the ones generated at ad-hoc manner. Regarding the comparison criteria, results showed up that process instances generated with support of SOAR-S were significantly better than the ones generated in ad-hoc manner.

Chapter 6 describes SOAR-E, the SOAR practice for architectural evaluation of software architectures in acknowledged SoS projects. We conducted a survey with experts to evaluate SOAR-E, obtained results showed up the relevance and acceptance of this practice.

Finally, Chapter 7 concludes this thesis, revisiting achieved contributions, summarizing limitations, and presenting perspectives of future research.

Five appendices are also included. Appendix A presents both the research protocol and the list of included studies regarding the SM presented in Chapter 2. Appendix B presents both the research protocol and the list of included studies of the SLR presented in Chapter 2. Appendix C presents the summarized description of Essence OMG's Standard, including a process authoring language and a general kernel adopted for SOAR process. Appendix D illustrates the use of SOAR-S to produce a process instance for a flood monitoring SoS. Finally, Appendix E presents questionnaires applied in the surveys.

Chapter

2

State of the art of SoS Software Architectures

Systems-of-Systems (SoS) have brought considerable challenges to the area of Software Architecture, mainly due to their inherent, unique set of characteristics. Hence, this chapter presents the state of the art on the research being conducted in SoS software architectures. Firstly, in order to present a background about SoS, Section 2.1 describes SoS, their characteristics, and types, and also examples of SoS. A conceptual model that we have established to classify if a system can be classified as a SoS is also presented. Following, Section 2.2 presents the state of the art on SoS software architectures. In more details, this section presents results obtained by conducting a systematic mapping and a systematic literature review encompassing the design issues and the architecting processes for SoS software architectures. Additionally, as results of these studies, a framework to characterize research in SoS software architectures and a list of process requirements to their design processes are also presented. Final remarks of this chapter are presented in Section 2.3.

2.1 System-of-Systems (SoS)

In the last years, there has been a growing interest in the research and development of complex systems called *Systems-of-Systems* (SoS) resulted from the interoperation of other

independent and heterogeneous systems (Firesmith, 2010). SoS were initially applied as a paradigm of how to deliver unique capabilities that are the result of a collaborative work of a dynamic set of constituent systems (DoD, 2008; Firesmith, 2010; Maier, 1998). Nowadays, SoS have been applied to several application domains in which their perspective of emergent capabilities tend to be more adequate than monolithic ones.

From another perspective, a *software-intensive system* is defined as any system in which software essentially influences the design, construction, deployment, and evolution of the system as a whole to encompass individual applications, subsystems, SoS, product lines, product families, whole enterprises, and other aggregations of interest (ISO/IEC/IEEE, 2011). By following a natural trend of complex, large-scale systems to be more software dependent, SoS tend to be also software-dependent, becoming the so-called *Software-intensive SoS* (i.e., SiSoS[1]) (Boehm and Lane, 2006). Due their complexity and growing software dependence, the SoS development is a multidisciplinary field that has gained relevance in the software community and also demanded for new investigations of software engineering solutions to support development of these systems.

2.1.1 Characterizing SoS

The first step on characterizing SoS is to differentiate them from monolithic systems (Butterfield et al., 2008). Based on Lane and Valerdi (2007) study, Table 2.1 presents a series of comparison perspectives differentiating SoS from monolithic systems. In this comparison, it is possible to notice how SoS present a more dynamic and evolutionary perspective than monolithic systems, being more suitable for contexts of collaborative operation and multiple needs.

The central idea of SoS refers to a dynamic set of independent systems that collaborate delivering unique and emergent capabilities (DoD, 2008; Maier, 1998). When analyzing what systems can be classified as SoS, it is noticeable that other classes naturally dialogue with SoS, e.g., distributed systems, complex systems, and federated systems. Despite the relation among them, SoS have a set of characteristics that place them as an unique system class. Therefore, we recognize that overlaps indeed exist, but they are not determinant for general classifications or direct correspondences. For example, SoS are distributed systems, however, not any distributed system will be an SoS. In this perspective, SoS present a level of singularity that encourages research to investigate solutions for their particular challenges.

Since SoS are inherently multidisciplinary, several researchers have proposed different definitions for them over the past years (Firesmith, 2010; Lane and Valerdi, 2005). Among

[1]For sake of simplicity, we adopt the term "SoS" when referring to "SiSoS". This section is an exception for this convention because here we describe and differentiate these terms.

Table 2.1: SoSs vs. monolithic systems (Adapted from (Valerdi et al., 2007))

Comparison Perspective	**Monolithic Systems**	**SoS**
Architecture	Single system (dependent constituents)	Multiple independent systems (monolithic ones or even other SoS)
Boundaries	Static	Dynamic
Problem	Defined	Emergent
Life cycle	Predictable	Evolutionary and continuous
Information Flow	Well-defined	Continuously changing
Development Focus	Defining and optimizing	Culture and cooperative tools
System Architecture	Established early in the life cycle; expectation set remains relatively stable	Dynamic adaptation as emergent needs change

these definitions, there is an absence of a single precise categorization for SoS (Abdalla et al., 2015; Firesmith, 2010; Jamshidi, 2008a; Klein and van Vliet, 2013) and, consequently, SoS are frequently developed without the "SoS" label (Klein and van Vliet, 2013). However, existing literature offers a rich set of descriptions of SoS and their characteristics, allowing to understand and extract what is already consensual in this field and what establishing SoS as an unique class of systems (Nielsen et al., 2015).

The initial efforts to support SoS categorization were introduced in systems engineering area. The first taxonomy for SoS was proposed by Maier (Maier, 1998) in 1990's in which three basic types (*virtual*, *collaborative*, and *directed*) and five main characteristics (*operational independence*, *managerial independence*, *evolutionary development*, *emergent behavior* and *geographic distribution*) were specified. With the evolution of the SoS community, several definitions for SoS in different contexts were proposed later. Some studies (Firesmith, 2010; Jamshidi, 2008a; Lane and Valerdi, 2007; Sharawi et al., 2006) analyzed these definitions to understand their differences and commonalities. In general, these studies ratified the characteristics initially proposed by Maier as the consensual ones for SoS.

2.1.2 Conceptual Model to Classify SiSoS

By taking advantage of the available literature of Systems Engineering, we proposed a conceptual model to support classification of SoS in the software-intensive context (Gonçalves et al., 2014). At first, we considered studies that address consensual characteristics and sets of definitions related to SoS (DoD, 2008; Firesmith, 2010; Jamshidi, 2008a; Maier,

1998; Sauser et al., 2010). We also analyzed existing taxonomies and ontologies that described SoS at a systems engineering level as references to establish our model. Finally, these concepts were analyzed in the light of software-intensive systems. Our strategy was to extend the ISO/IEC/IEEE 42010 International Standard (ISO/IEC/IEEE, 2011), which provides a conceptual framework for creation, analysis, and specification of architectures of software-intensive systems. We chosen it as a basis for our conceptual model since it is a standard that addresses important elements regarding software-intensive systems, such as notions of system, stakeholders, purpose, and environment.Figure 2.1 depicts elements of our model and relationships among them. As in ISO/IEC/IEEE 42010(ISO/IEC/IEEE, 2011), this conceptual model uses conventions for class diagrams defined in ISO/IEC 19501(ISO/IEC, 2005).

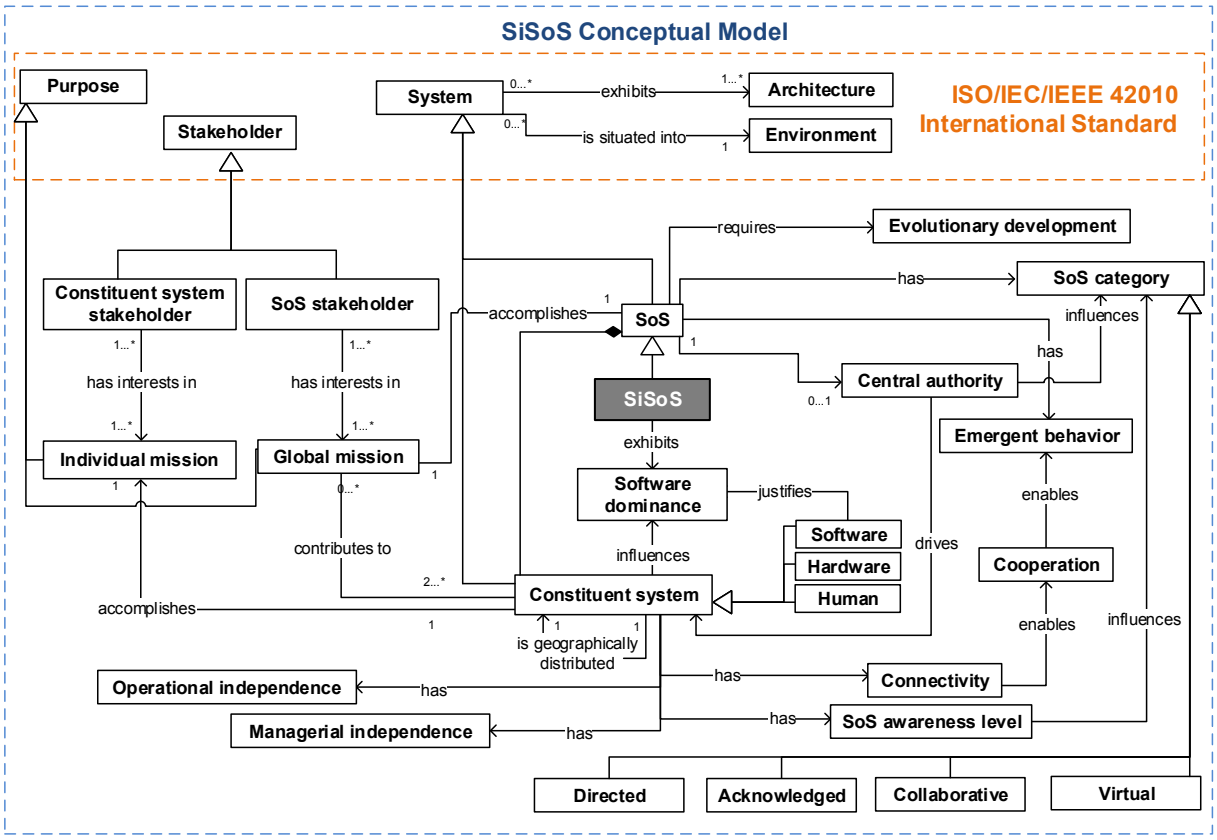


Figure 2.1: Conceptual model for SiSoS (Gonçalves et al., 2014).

Table 2.2 presents the definition of the key concepts required to understand if a given system can be a software-intensive SoS (i.e., SiSoS) by using the model. At first, an SoS exists to accomplish a *global mission*, which represents its major goals. An SoS is composed of *constituent systems*, which contribute to the accomplishment of the global mission of the SoS and they can have *managerial independence* and an *individual mission*.

Moreover, constituent systems have *operational independence* and present the ability of connecting to each other (*connectivity* (Boardman and Sauser, 2006; Zhou et al., 2011)), thus enabling their *cooperation* to yield emergent functionalities, the so-called *emergent behavior*. In addition, both an SoS and its constituent systems can have their respective stakeholders (*SoS* and *constituent systems stakeholders*), which have interests in their missions.

Table 2.2: Key concepts for SoS (Boardman and Sauser, 2006; DoD, 2008; Firesmith, 2010; Maier, 1998; Zhou et al., 2011)

Key concepts	**Definition**
Global mission	A set of goals to be accomplished by a SoS through the collaboration of constituent systems.
Software dominance	Software-based constituent systems are the dominant factor for operation of a SoS, so that it cannot operate or accomplish its global mission without contribution of these systems.
Operational independence	All constituent systems of a SoS can deliver their functionalities when not working with other systems. They must also have at least the capability of individually operating.
Managerial independence	All constituent systems are autonomously managed by independent sources. They must also have at least the capability of being individually managed.
Evolutionary development	A SoS as a whole may evolve over time to respond to changes on its environment, on constituent systems, or on its own mission.
Emergent behavior	It is a behavior resulted from the collaboration of constituent systems and that cannot be provided by any of these systems if they operate as individual entities.
Distribution	Constituent systems of a SoS are physically decoupled, thus only exchanging information among them.
Connectivity	Constituent systems must have the ability to form effective connections with other constituent systems to accomplish common goals.

Particularly, a SiSoS is an SoS in which software is the dominant factor in its operation, i.e., it exhibits a *software dominance*. Despite a SiSoS can comprise human-based, hardware-based or software-based constituent systems(DeLaurentis, 2008). Therefore, for SiSoS, the set of participating software-based constituent systems justifies the software dominance, so that a SiSoS cannot operate without them. Furthermore, SoS are typically complex, large-scale systems whose functionalities and purposes can change and dynamically evolve, i.e., they present an *evolutionary development*(Firesmith, 2010; Maier, 1998).

As a consequence, the *architecture* exhibited by an SoS is dynamic and reactive, since it must evolve according to the dynamic collaboration of constituent systems that are also changing. Moreover, representation and simulation of these architectures allow to understand possible emergent behaviors, even the not desired ones.

Table 2.3 complements our model presenting four categories in which any SoS can be classified. Three of them were defined by Maier (1998) and the *acknowledged* category was proposed by Dahmann and Baldwin (2008). In this context, the *central authority* determines how constituent systems cooperate and how subordinated they are to a global authority at SoS level. Virtual SoS is the only category with no central authority, in which constituent systems are completely unaware of their participation in the system. In other categories, i.e., collaborative, acknowledged, and directed, constituent systems are aware of such participation in different levels. Directly related to this concept, the *awareness level* stands for the degree in which constituent systems are aware of their participation in SoS operation.

Table 2.3: Basic categories of SoS (Dahmann and Baldwin, 2008; Maier, 1998)

Category	**Definition**	**Level of central authority**
Virtual	Constituent systems are independently managed in a distributed and uncoordinated environment where mechanisms to maintain the whole SoS are not evident.	Nonexistent.
Collaborative	Constituent systems voluntarily collaborate more or less to address shared or common interests.	The authority offers standards to enable the collaboration of constituent systems.
Acknowledged	Goals, management, resources, and central authority of the SoS are all recognized, but the constituent systems still retain their independent management.	The authority is based upon negotiation between constituent systems and the SoS as a whole.
Directed	Constituent systems can have their operational and managerial independence, but their behavior is subordinated to a central authority and its purposes.	The authority and its specific main purposes are evident and drive constituent systems.

Regarding the use of proposed model, we consider that a system can be characterized as a SiSoS if all key characteristics presented in Table I are encompassed. For example, the operational independence is adequately encompassed if constituent systems have at least the ability of operating in an independent way. After identifying that a system is a SiSoS, its category is determined according to definitions presented in Table II. Therefore,

concepts presented in the proposed model enable to identify if a system is a SiSoS or not and to which category of SoS it belongs.

It is noteworthy this model is not just a review of the existing literature about SoS, but it represents an effort towards standardization of concepts related to this class of systems and, by extension, to SiSoS. Therefore, this model can be useful to support the research on SoS by being a source of consensual knowledge about this class of systems and an useful resource for analyzing software-intensive systems under SoS perspective. Hence, we hope to contribute to development and research on these systems by providing means to adequately understand and classify them. Finally, despite we use the term SoS meaning SiSoS in this thesis, we make clear in this section the differences between them and reinforce that the scope of this thesis is SiSoS.

2.1.3 Illustrative Examples

Over the past decades, several improvements in software and communication technologies have enabled researchers to develop SoS of higher levels of cooperation (Nielsen et al., 2015). In this context, new examples of SoS can be found in different application domains revealing SoS as a promising system class (DeLaurentis, 2005b; Nielsen et al., 2015).

Global Earth observation: it is an application domain that refers to monitor several Earth environments, their related conditions (e.g., climates, crops, forests, and deserts), changes, and potential impacts (Johnson, 2008). This complex scenario makes a monolithic system impracticable (Khalsa et al., 2009) and the Global Earth Observation System-of-Systems(GEOSS)[2] is a good example of SoS suitable to this domain. GEOSS is also a global project encompassing over 60 nations and is constituted by integrated systems that create information for environmental decision making. GEOSS results from combined efforts and managerial independence among constituents. Each constituent system is subordinated of its own source (country, organization, research group, etc.), and the operational independence follows the same reasoning. GEOSS includes shared data and systems provided by different and geographically distributed sources, delivering unique capabilities, such as complex crossed information not possible to obtain by isolated systems working alone, what characterizes an emergent behavior.

Health assistance: it is another application domain in which SoS has been introduced. Petcu and Petrescu (2010), an SoS for health assistance to persons living alone. In such system, patients are monitored into their homes, which are distributed on large geographic areas, including different levels of collaboration and constituent systems

[2]http://www.earthobservations.org/

with different operational features, e.g., monitoring of vital signs and ambulance fleet management. The authors report the development of their system in an SoS perspective facilitated obtaining of systems cooperation and consequent achievement of expected emergent functionalities.

Transportation: this domain typically involves large-scale networks of independent and geographically distributed systems. DeLaurentis (2005a) describes how transportation systems can be developed as SoS and argues that the SoS characteristics, such as emergent behaviors and evolutionary development, are suitable for transportation sector. Nielsen and Larsen (2012) present an SoS for vehicle monitoring that is designed to improve road safety by making available the traffic information to drivers. In this scenario, vehicles are constituent systems with independence of movement, resulting in high dynamicity for generating and sharing traffic information.

Emergency management and response: in this application domain, SoS have encompassed the need of cooperation of different organizations and systems in critical scenarios of imminent emergency. For example, Payne et al. (2012) describe an SoS for major incident response, in which emergency services interoperate to response emergencies. By using the SoS perspective, emergency systems can voluntary collaborate making possible a comprehensive incident management based on emergent functionalities. Another example in this domain is the monitoring of floods in urban areas (Hughes et al., 2011). A flood monitoring SoS can support the monitoring of urban rivers and create alert messages to notify authorities and citizens about risks in case of an imminent flood. The collaborative work of different constituent systems, e.g., systems embedded in sensors and data analysis systems, is quite important to detect risks of flood and to trigger warning messages.

Several other examples of different application domains can be mentioned, such as: aerospace (Bonilla et al., 2005; Dvorak et al., 2005; Jackson et al., 2012), energy managemnt (Agusdinata and DeLaurentis, 2008; Parker, 2008; Pérez et al., 2013), and military defense (Belloir et al., 2014; Shing et al., 2006). The amount of publications related to several different application domains indicates that SoS has raised considerable attention from academia and industry being a relevant system class for nowadays systems.

2.2 Architecting SoS

SoS usually involve a number of complex constituent systems, different technologies, and several teams and organizations that apply different approaches to develop these systems.

The architecture of an SoS includes knowledge about SoS in different aspects and architectural layers, e.g., organizational, physical, and computational. For each of these layers, a proper architecture must be conceived to encompass operations, functions, behaviors, internal and external relationships, and dependencies regarding SoS and its constituents (DoD, 2008). Following, we first introduce general concepts related to software architectures (in Section 2.2.1) and next we present specific details about SoS software architectures. Results of our systematic mapping that we conducted to identify and analyze the state of the art about how SoS software architectures have been treated in terms of representation..are presented in Section 2.2.2. Finally, Section 2.2.3 presents results of our systematic literature review that investigate processes proposed to SoS software architectures.

2.2.1 Software Architecture

From the first occurrence of "Software Architecture" term in 1969, only on 1990s it became more effectively widespread (Kruchten et al., 2006). Nowadays, software architecture concept is recognized as a discipline of software engineering (Clements et al., 2010). Bass et al. (2012) define software architecture as the structure (or a set of structures) of the system, which comprises software elements, externally visible properties of those elements, and the relationships among them. From a generic perspective, a software architecture can be considered an abstraction that suppresses internal details on how software elements are implemented, being concerned with their arrangement, interaction, and composition (Bass et al., 2012). Software architecture establishes a link between abstract business goals and concrete resulting system, making possible different people and organizations to deal with the system complexity on adequate levels of abstraction (Clements et al., 2010).

Software architectures can encompass issues related to functional and non-functional requirements of systems; however, since the system quality is highly dependent on decisions made at the architectural abstraction levels, establishing adequate software architectures is a critical concern in determining non-functional requirements and quality issues (Shaw and Clements, 2006). With the emergence of software architecture as a discipline, new related concepts also emerged. In this scenario, efforts have been conducted to establish a common vocabulary in the software engineering area, in particular, standards (IEEE Computer Society, 2014; ISO/IEC/IEEE, 2011) or lists of terms and concepts (Eeles, 2008; SEI, 2015). In spite of these efforts, some different terms still have been sometimes used as synonyms and must be explicitly defined to avoid misconceptions when used in this thesis. Therefore, the main terms related to software architecture and used in the context of this research are described as follows:

Stakeholder: the one who has interest in a system and plays a relevant influence during its development (ISO/IEC/IEEE, 2011). Stakeholders can be (but are not restricted to) customers, teams, organizations, regulatory entities, and developers (IEEE Computer Society, 2014);

Concern: an interest in a system relevant to one or more of its stakeholders (ISO/IEC/IEEE, 2011). Concerns can influence in different aspects of a system, e.g., technological, business goals, operational, regulatory, legal, ecological, and social. Examples of concerns include regulatory restrictions, specific technologies to be used, purposes of use, budget constraints, and desired functionalities;

Concrete software architecture (or just software architecture): the set of concepts, properties, specifications, design principles, and patterns proposed for a particular system and embodied in its elements and their relationships (ISO/IEC/IEEE, 2011);

Architectural Decision: any decision made at architectural level that influences the resultant concrete architecture. Architectural decisions help a system to meet its quality requirements (Clements et al., 2010);

Architectural style: a specialization of software elements and relation types, together with a set of guidelines on how they can be used (SEI, 2015). Architectural styles are based on recurrent design problems and conceived to bring well-proved design experiences, defining a structural organization suitable for a particular group of systems. There are several widely accepted architectural styles in the literature, such as client-server, pipe and filters, Service-Oriented Architecture (SOA) (Eeles, 2008). Client-server is a widely-used style that proposes the physical separation of client-side processing (such as a browser) and server-side processing (such as an application server that accesses a database). Pipes-and-filters comprises a series of filters, which provide data transformation, and pipes that connect the filters. SOA proposes structure the system into modules called services (Papazoglou and Heuvel, 2007). In SOA systems, all functionalities are provided as services, which are black-boxes that hide implementation details and operate independently communicating through interfaces and providing one or more functionalities (Josuttis, 2007);

Architecture View: a way to express the architecture of a system from the perspective of specific system stakeholders and concerns (ISO/IEC/IEEE, 2011). Architectural view is a representation of a particular system from a particular perspective of interest that discloses the system architecture and supports several engineering activities,

such as construction, management, exploration, training, and testing (Garland and Anthony, 2003);

Architecture Viewpoint: the conventions for construction, interpretation, and use of architecture views to meet specific system stakeholders and concerns (ISO/IEC/IEEE, 2011). Architectural viewpoints can establish related stakeholders, concerns addressed by viewpoint, and other conventions to develop representations in conformance with a particular view; and

Architecture Description: a work product, or a set of work products, used to express an architecture including all required views (ISO/IEC/IEEE, 2011). Architecture description allows capturing and conveying of important architectural aspects among system stakeholders, such as design decisions, system behavior, and constraints. In this perspective, architecture description must provide elements to adequately express a system architecture, such as specific documentation, architectural viewpoints, and architectural prototypes (ISO/IEC/IEEE, 2011).

Based on ISO/IEC 42010 2011, Figure 2.2 shows a relational scheme that depicts key relationships among defined terms.

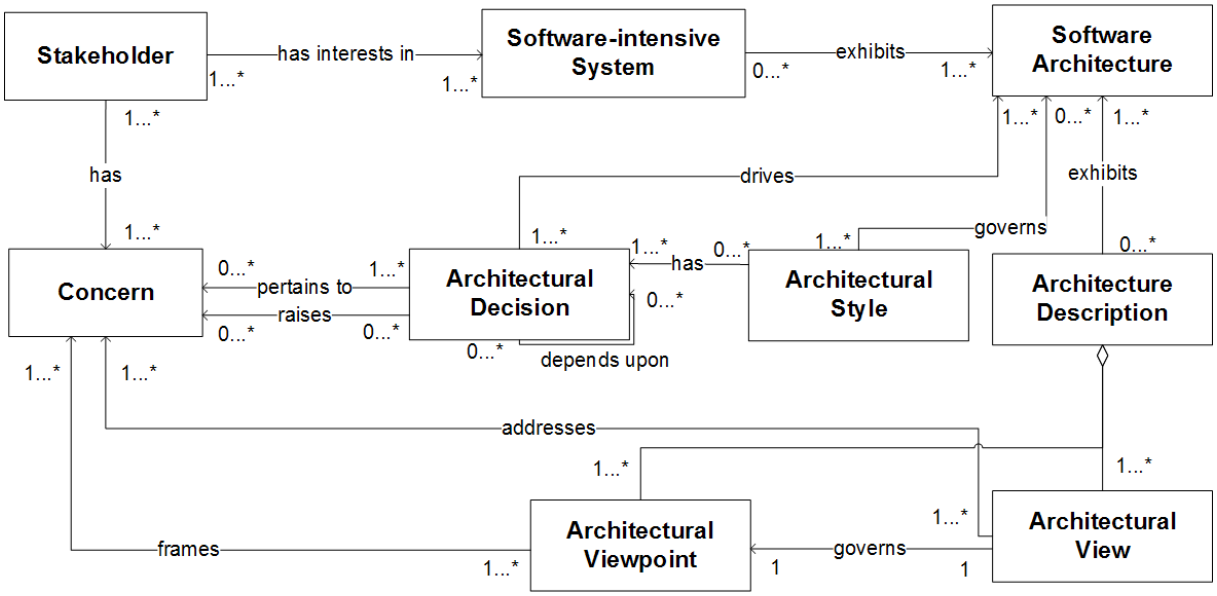


Figure 2.2: Context of software architecture (Adapted from ISO/IEC 42010 (2011))

2.2.2 SoS Software Architectures: a Sytematic Mapping

The design of SoS software architectures is a new challenging research area, with an increasing amount of publications the last years (Nakagawa et al., 2013). In this context,

we conducted a SM study[3] that aims to establish a fair, broad overview of the research involving SoS software architectures. We also proposed in this study a framework to characterize existing research in this area. Aiming at identifying as many evidences as possible, the following Research Questions (RQs) were established:

- **RQ 1:** What are the main architecturally significant characteristics of SoS?
- **RQ 2:** What are the main quality attributes of SoS software architectures?
- **RQ 3:** How have SoS software architectures been represented?
- **RQ 4:** How have SoS software architectures been evaluated?
- **RQ 5:** How have SoS software architectures been constructed?
- **RQ 6:** How have SoS software architectures been evolved?

This SM is an ongoing work and its initial results were published in (Nakagawa et al., 2013). Following, we present the updated results regarding last finished selection of this SM, which included studies published until January 2014. However, this study is being updated to produce a new version, including studies published until July 2016. In the last finished selection, we obtained a final set of 91 primary studies as relevant. Before to present results related to the main topics of investigation (i.e., architectural representation, evaluation, construction, and evolution, as well as the architecturally relevant characteristics and quality attributes of SoS), we following provide general results with regard to: (i) distribution of primary studies over time; (ii) main research groups in SoS software architectures; and (iii) application domains addressed by the studies.

Figure 2.3 shows the distribution of primary studies through their years of publication. As the SoS software architecture is a relatively new research field, most of the studies were published in the last seven/eight years. From a total of 91 studies, 92.3% were published from 2005 to 2014. If this trend continues, for the next years, we can foresee a number of contributions for this area.

An analysis for the identification of the research groups/institutions that have more widely contributed to the area of SoS software architecture was also conducted. For each primary study, we identified the groups/institutions of its authors. No group/institution can be considered representative in the area of SoS software architecture. In particular, Naval Postgraduate School/ Computer Science Department (USA), Carnegie Mellon University/Software Engineering Institute (USA), and University of California (USA) have seven, six, and four primary studies, respectively. Other groups/institutions have fewer

[3]Details about the research protocol followed in this SM and the list of included studies are both presented in Appendix A.

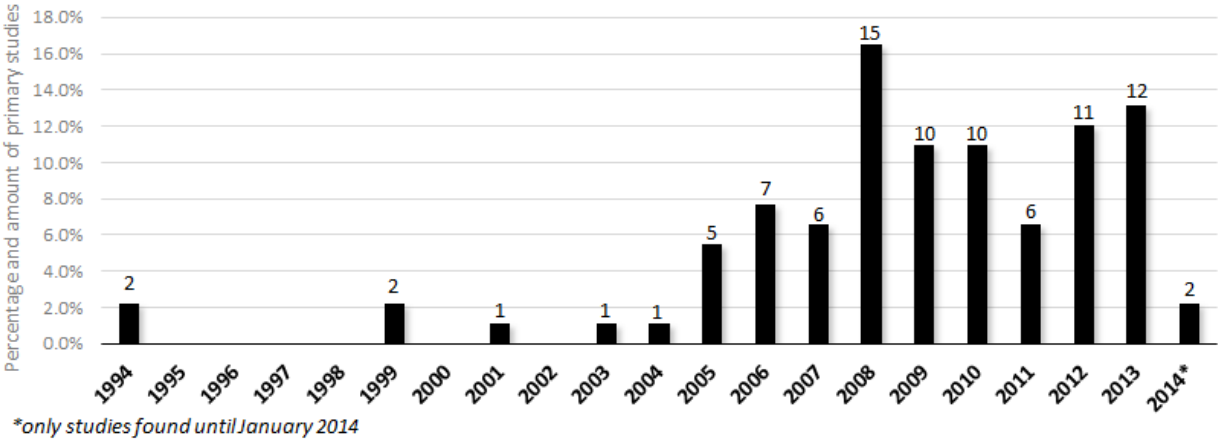


Figure 2.3: Distribution of primary studies through the years

than three studies. Moreover, approximately half of the studies are single publication of a group/institution. Therefore, there is still a lack of mature, representative research groups/institutions widely contributing to the SoS software architectures. Figure 2.4 shows the distribution of primary studies per country. For each study, the country of all its authors was considered. For instance, there are five studies that have one or more authors from Brazil. Overall, USA has stood out in terms of the number of studies, mainly with studies on SoS for the military domain.

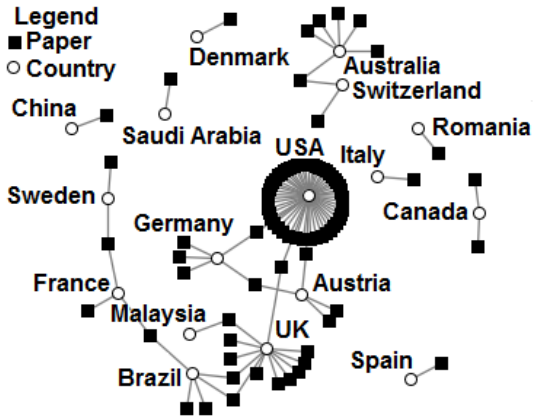


Figure 2.4: Countries of the authors of the primary studies

Primary studies of our SM address different application domains: (i) 26 for civil domain; (ii) 24 for military domain; and (iii) 13 address both military and civil. Other 29 studies do not present SoS for any domain. With regard to military domain, four subdomains can be found, as presented in Figure 2.5.a. For civil domain, diverse subdomains can be also identified, as illustrated in Figure 2.5.c. We classified the remaining studies as possibly applicable to both military and civil domains as showed in Figure 2.5.b. An important observation is that most SoS found in the primary studies can be classi-

fied as critical SoS, which requires considerable efforts and attention for their adequate development.

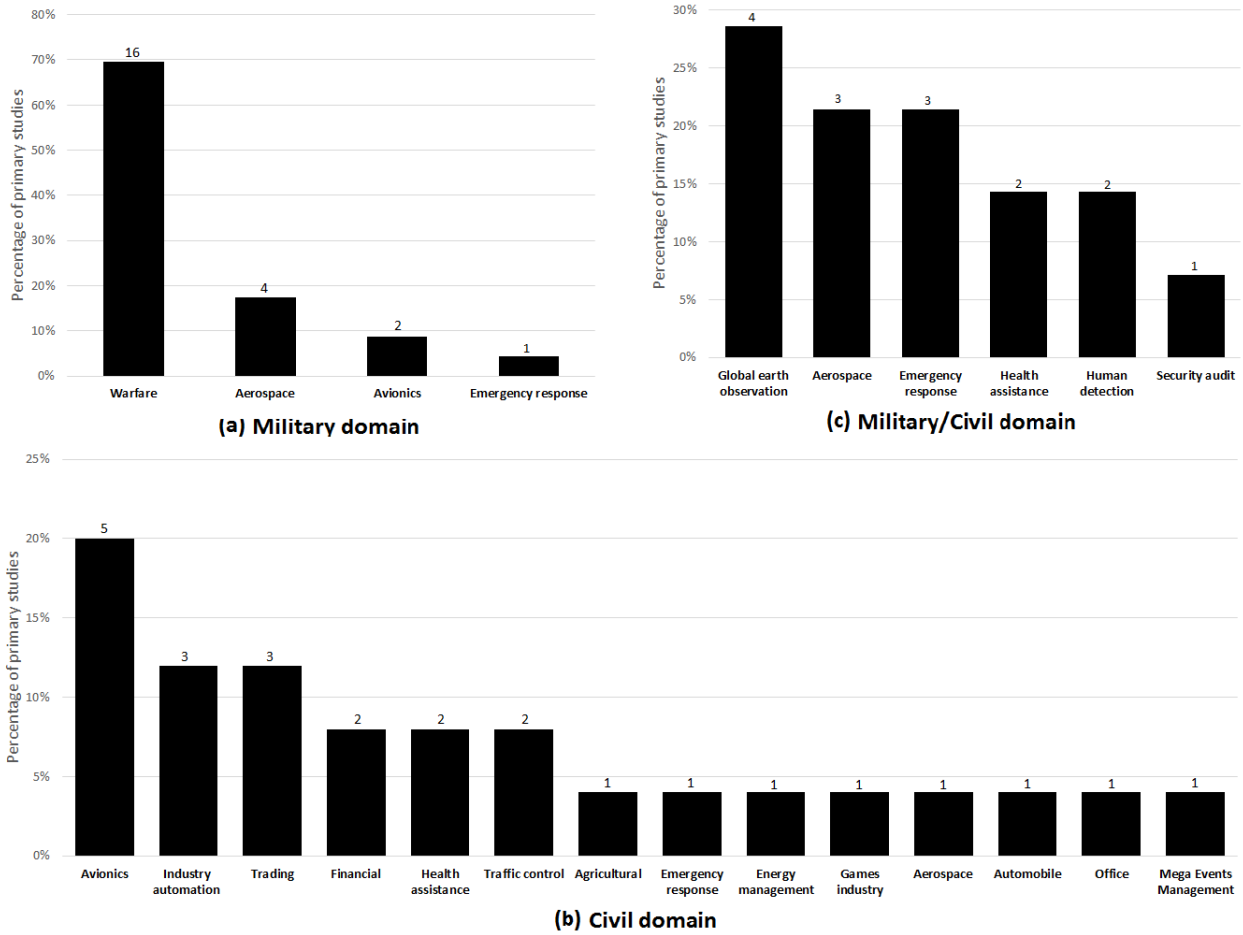


Figure 2.5: Application domains of SoS

Based on analysis of all primary studies included, we present our findings and discuss related challenges for each RQ.

RQ1: Architecturally Relevant Characteristics

We observed that 50.5% of the included studies (i.e., 46 studies) answer this RQ by addressing characteristics of SoS that affect their software architectures. We compared these characteristics and found that the most common ones are the characteristics already proposed in Maiers' work (Maier, 1998). For example, Eaton et al. (2008) focus on agricultural automation and Nguyen et al. (2012) focus on urban transportation, but both studies point out the operational independence as a characteristic relevant in the architectural design of their SoS. Figure 2.6 summarizes the common characteristics of SoS found: operational independence, managerial independence, emergent behavior, geographic distribution, and evolutionary development. The five characteristics proposed by Maier (Maier, 1998) directly affect the SoS software architectures. Although evolutionary

development is a characteristic that refers to the capability of SoS of adapting in runtime, it does not explicitly address the inherent dynamism of SoS software architectures. An inherent characteristic is then the dynamic architecture. SoS can dynamically change their architecture to fit to changes in their constituents and missions.

In general, the characteristics identified in this SM seem to be the most relevant ones for SoS and these characteristics that can directly affect the conception of their software architectures. Therefore, they must be considered in processes/methods that systematize the development of such architectures.

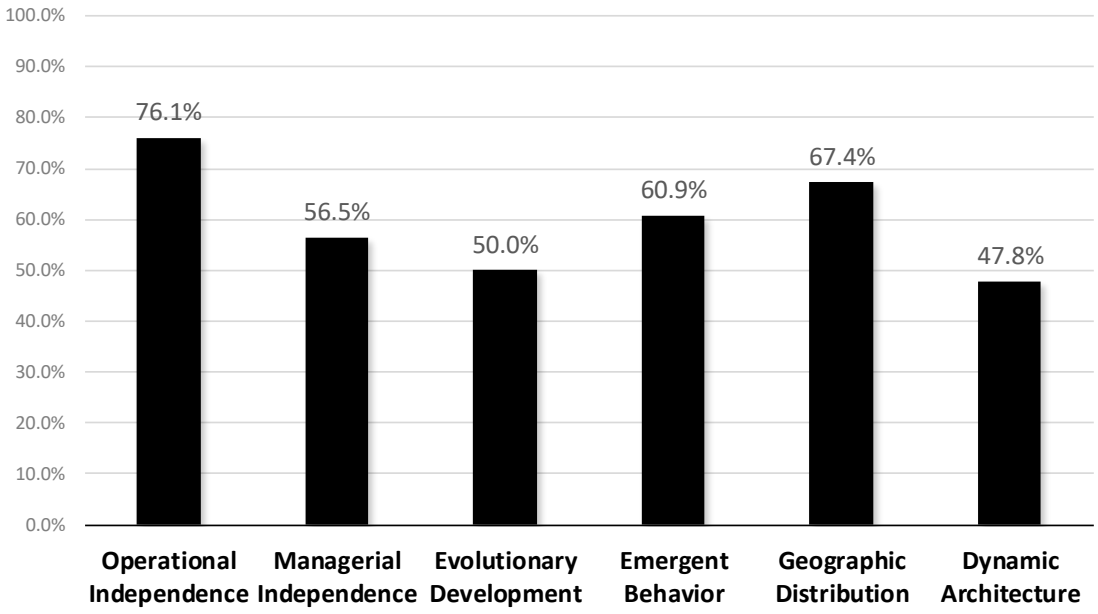


Figure 2.6: Architecturally relevant characteristics of SoS

RQ2: Quality Attributes of SoS

Quality attributes that directly affect the SoS software architectures were identified in 51.6% of studies (i.e., 47 studies). Figure 2.7 shows the set of attributes and their percentage of occurence among the studies. Such attributes have the same definition/meaning of those presented in the international standard ISO/IEC 25010 (ISO/IEC, 2011). It was not our concern to organize these attributes in a hierarchical level, for instance, considering availability as a sub-characteristic of reliability. This decision was motivated to keep the fidelity of our findings. Furthermore, we found two new quality attributes: "integrability" (the ease of integration of new constituent systems in an SoS) and "predictability" (capability of predicting emergent behaviors, even those not initially foreseen). Therefore, it is quite important to highlight that this international standard, which should encompass

quality attributes/characteristics and sub-characteristics of any type of software-intensive systems, needs to be updated to meet with software-intensive SoS.

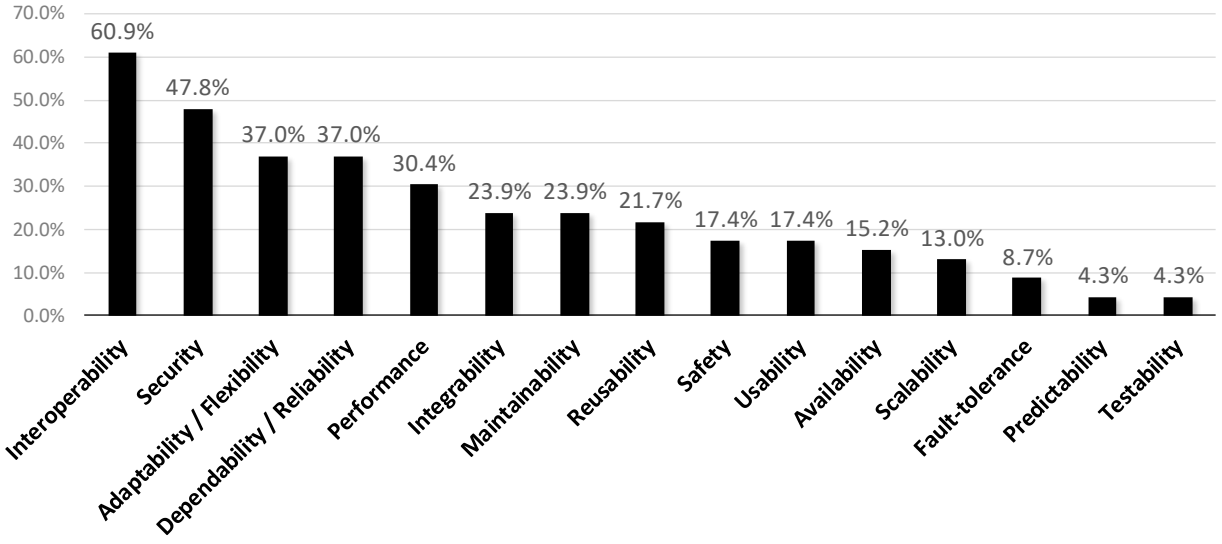


Figure 2.7: Quality attributes of SoS

Interoperability is the most recurrent quality attribute, present in 28 studies (i.e., 60.9% of a total of 47). It can be understood as the capability of two or more systems to exchange information and use it (ISO/IEC, 2011). Interoperability is also determinant for promoting several common SoS characteristics, mainly emergent behaviour and evolutionary development. According to Butterfield et al. (2008), interoperability can also contribute with other quality attributes, specially performance and reliability. However, more studies are required in the context of SoS, even in particular application domains, to better understanding of the relationship and trade-off among these attributes.

Other relevant quality attributes that influence the development of SoS software architectures are security, performance, adaptability/flexibility, dependability/reliability, integrability, maintainability, and reusability. They were found in 21.7% to 47.8% of the total of 47 studies that answer this RQ. The remaining quality attributes have lower occurrence level and appear in isolated studies sometimes related to specific application domains. This first set of main attributes can provide directions on what is important, or even essential, in the development and evaluation of SoS software architectures. However, considering the knowledge we gained reading and analyzing those primary studies, we can conclude these attributes must be deeply investigated. For example, the low occurrence level in some quality attributes does not mean they are not relevant to SoS, but because SoS has been a more recent topic of research, these attributes will require

to be more deeply studied. The area of SoS requires a set of common quality attributes independently of application domains, technologies, and types of SoS.

Regarding the application domains, we observed that quality attributes are almost equally distributed through them. Certainly, these attributes should have been found for all application domains. For example, SoS for medicine presents interoperability, security, and dependability, but it does not treat other important attributes, such as performance and safety, which should be considered in systems of this nature. Furthermore, it was not still possible to identify a clear relationship between SoS characteristics and quality attributes. It will be important to know how, when, and which quality attributes should be considered/implemented in an SoS, intending to meet, for instance, the characteristic of evolutionary development. From this perspective, SoS characteristics and quality attributes could be jointly treated during the development of an SoS. Moreover, the primary studies found in our SM enable no identification of the architectural decisions (e.g., architectural styles, communication protocols, technologies, and so on) could be adopted for meeting SoS characteristics and achieving the quality attributes. The next four sections provide a panorama of how SoS software architectures have been developed and can contribute towards clarifying the way how some architectural decisions affect SoS software architectures and quality attributes.

RQ3: Architectural Representation

A precise description can support the development, maintenance, and evolution of SoS software architectures. Therefore, we investigated how their software architecture has been represented. From a total of 91 studies, we observe that 38.5%(i.e., 35 studies) of them address the representation of SoS software architectures using informal, semi-formal, and/or formal languages. Figure 2.8 shows the representation approaches proposed in these studies. Informal representations were found in seven studies, of which two (Bhasin and Hayden, 2008a; Bonilla et al., 2005) adopt architectural views proposed by Department of Defense Architecture Framework (DoDAF) (DoD, 2010) to represent their architectures. In general, these seven studies are more focused on presenting the architecture rather than proposing a solution for their representation. On the other hand, semi-formal representations have been widely used (26 studies). In general, these studies use Unified Modeling Language (UML). Although most SoS have been built for critical domains, such as aerospace and automotive, formal techniques and languages have not been widely considered. Only six initiatives address the use of a formal representation. For instance, Gamble and Gamble (2008) extended the UNITY formal specification language to capture programmatic, structural, and scoping properties of SoS and analyze architectural properties. Different representation techniques are also used and combined for the SoS context. For example, Wang Wang and Dagli (2011) uses SysML-based specifications

and Colored Petri Nets (CPN) in a paradigm that enables static and dynamic system analyses. Regarding representation techniques, the most used ones are UML sequence diagrams, state diagrams, and Message Sequence Charts (MSC).

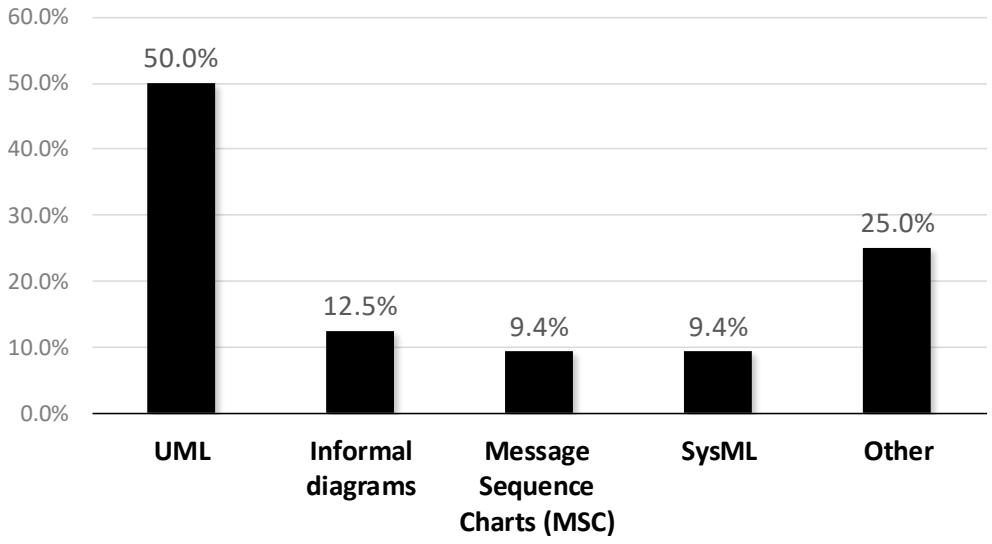


Figure 2.8: Approaches for architectural representation of SoS

We also investigated architectural views adopted to SoS software architectures and four views proposed in RUP 4+1 views (Kruchten, 1995) were found: (i) logical view, which concerns the functionality the system provides to end-users; (ii) process view, which deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system; (iii) deployment view, which illustrates a system from a programmer's perspective and is concerned with software management; and (iv) physical view, which concerns the topology of the SoS on the physical layer, as well as the connections among their constituent systems. In particular, the process view can address concurrency, distribution, integrators, performance, and scalability and, therefore, this view seems to be necessary for representing software architectures of SoS.

In summary, logical, process, and physical views are the most addressed views, UML is the most used language and, as a consequence, semi-formal representations are in general more disseminated than other techniques. While the use of formal representations is needed to support analysis and other automated tasks, we observe that formal representations are not generally widespread in industry; therefore, as SoS have been sometimes provided by initiatives from the industry, it is not expected that formal representations are being used.

This is a first panorama on how SoS software architectures have been represented. However, considering the studies found in our SM, a more adequate, complete way to represent software-intensive SoS can not be identified. For the future, there are still several open issues involving SoS representation, architectural views, and related techniques. A recent SLR from our research group showed a broader panorama on the architectural description of SoS (Guessi et al., 2015). It can complement this RQ, as it provides additional information and analysis, including the purpose of the architectural description of SoS and type of information represented (e.g., orchestration and communication among the constituent systems).

RQ4: Architectural Evaluation

The evaluation of software architectures ensures that architectural decisions are correctly made. It is also important for SoS software architectures, because of the critical aspects of such systems. 19.8% of the included studies (i.e., 18 studies) address the architectural evaluation. Figure 2.9 shows approaches considered by these studies for evaluation. Some studies address more than one approach/method and four studies explored Architecture Tradeoff Analysis Method (ATAM) and/or Software Architecture Analysis Method (SAAM), two well-known methods for the evaluation of quality-related issues for software architectures. For instance, Kazman et al. (2012) extended ATAM to address SoS software architectures and Gagliardi et al. (2009) present an approach also based on ATAM to identify architectural risks and inconsistencies in quality attributes across the constituent systems. At first, both ATAM and SAAM seem to be also adequate to SoS software architectures; however, more research must be conducted. The remainder of studies address isolated initiatives. For instance, Michael et al. (2009) introduced a mathematical model to combine non-functional requirements of SoS (in particular, those related to integrability) for analyzing the quality of software architectures.

Quality attributes have also been considered during the evaluation. Five studies address quality in a more general manner. With respect to specific quality attributes, security, dependability, performance, interoperability, and integrability were included in evaluation. However, based on this small number of studies, it is not possible to have a consensus on what exactly should be considered when evaluating SoS architectures. Several other attributes were found for SoS architectures (as previously presented in Figure 2.7). Therefore, investigation and inclusion of these attributes in the evaluation must be conducted. For example, adaptability refers to the degree to which an SoS can be adapted to be executed in a different environment; therefore, it is interesting to consider this attribute in the evaluation of SoS architectures. Other good examples are safety, availability, and fault-tolerance.

Commonly, after the architectural design (that commonly occurs after architectural analysis), software architectures are evaluated. Nevertheless, in the case of SoS, their architectures should be evaluated not only during their design, but also during evolution that occurs during their execution. SoS evolution can cause changes in their architectures, what requires a continuous evaluation. Our SM did not find any study that deals with this issue. Furthermore, SoS evolutions make evaluations quite difficult and expensive (Chen, 2006).

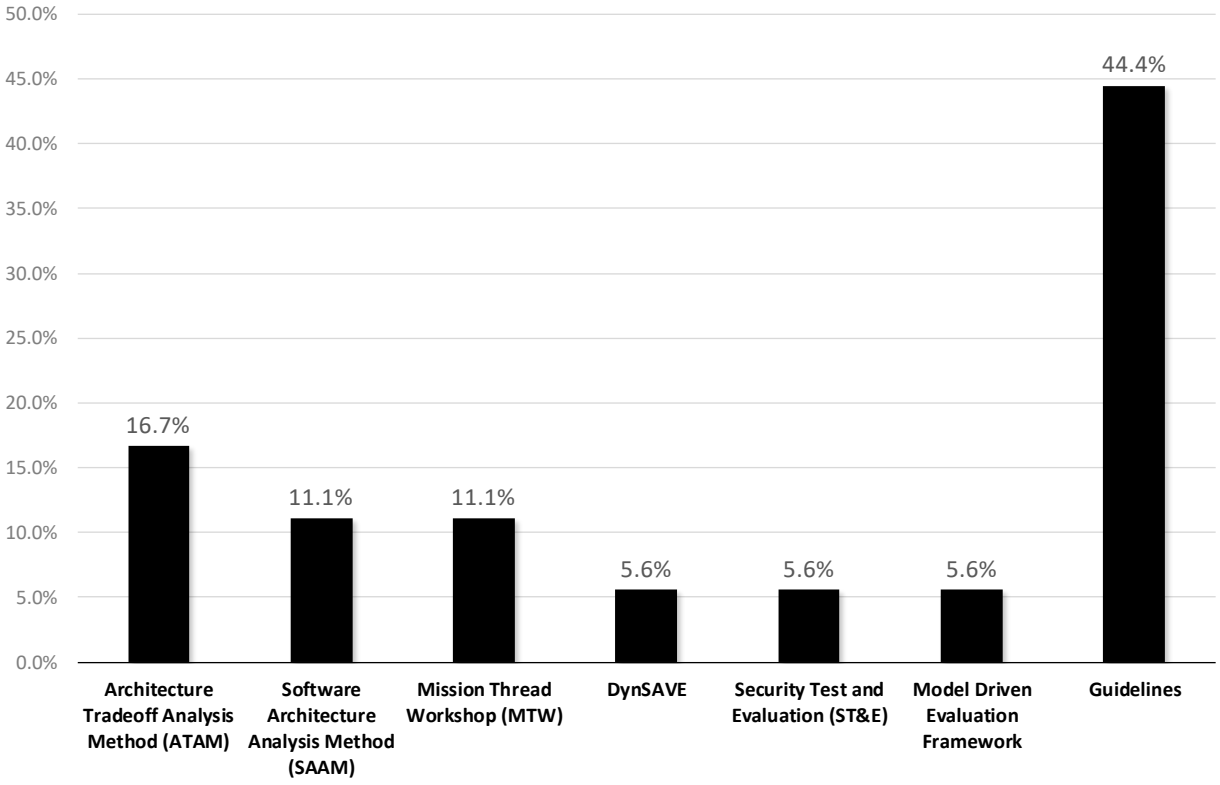


Figure 2.9: Approaches for architectural evaluation of SoS

RQ5: Architectural Design

This RQ explores approaches to support design of SoS software architectures, such as architectural styles, methods, processes, and guidelines. These different approaches are not directly comparable; however, our discussion was based on the set of 47 studies (i.e., 51.6%) studies found, providing the approaches, methods, processes, their advantages, and drawbacks.

Regarding processes for the architectural design of SoS, two studies ((Chigani and Balci, 2012; Liang and Luqi, 2003)) were proposed to organize the steps of the complexity associated with the design of SoS software architectures. The more complete study is Chigani's work (Chigani and Balci, 2012), which provides these main steps for the SoS

architecture construction, organized in five specific goals: Identify System Components; Establish Relationships among the Identified Components; Create the Architecture; and Describe the Architecture. Chigani's work also proposes specific practices to support building of SoS architectures. This is an important initiative to the direction of establishing a process to build such architectures; however, this and other proposed processes must be widely used in the industry to achieve a more consolidated, experimented approach. Moreover, this topic is explored in more details in the next subsection.

Besides architectural processes, we also identified other approaches presented in Figure 2.10, in which one study can include one or more of these approches. SOA has been advocated as a promising architectural style for SoS. SOA-based development appears in 27 of a total of 47 related to architectural design. For instance, Simanta et al. (2010) developed a detailed discussion about the use of SOA in SoS and argued that given the existing similarities between service-oriented systems and SoS, approaches and techniques that have been developed to support identification, publishing, discovery, and governance in service-oriented systems can be used to support SoS. Kruger et al. (2006) proposed the use of a SOA infrastructure for the cooperation of constituent systems in SoS. Similarly, in Bull et al. (2010) and Farcas et al. (2010), the authors suggested the use of SOA to facilitate the integration of constituent systems in SoS and a hierarchical architecture pattern, called Rich Services, to encapsulate the various capabilities and functionalities of SoS.

Another approach to design SoS software architectures is DoDAF, which is a comprehensive framework that provides guidelines for developing a standardized architecture (Bonilla et al., 2005). Although DoDAF has been initially conceived for military applications, it has been employed to other SoS application domains, e.g., aerospace (Bhasin and Hayden, 2008a). Additionally, there are isolated initiatives that explore software engineering approaches to support SoS design. Some examples include Capability Maturity Model Integration (CMMI), aspect orientation, reference architecture, design by contract, and architectural patterns. In Figure 2.10, these initiatives were put together in "Other" column.

RQ6: Architectural Evolution

As previously stated, evolutionary development is one of the main characteristics of SoS that directly influences their architecture. Therefore, these architectures must support the dynamic evolution of SoS, so as to incorporate new functionalities (or SoS missions) in runtime. Only two studies directly address evolution and its impact on the SoS software architectures. In particular, Chen and Han (2001) can be considered the most aligned with SoS evolution. The authors proposeSelberg and Austin Selberg and Austin (2008) presented key characteristics (for instance, standardized interfaces and interface layers)

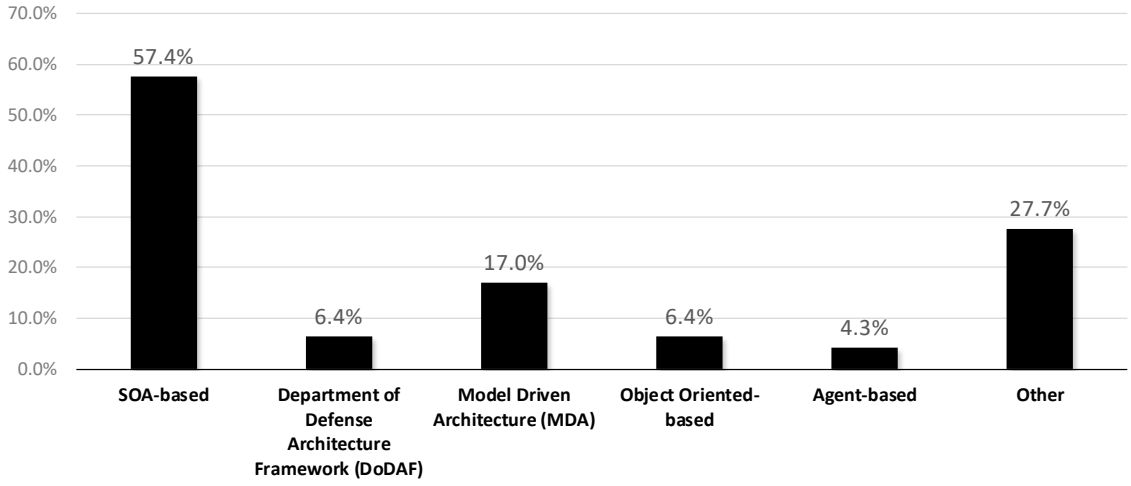


Figure 2.10: Approaches to support architectural design

that could provide a dynamic architecture to the SoS. No study has reported concrete experiences of how real SoS and their architectures have evolved. The lack of evidences in this RQ indicates evolution has occurred possibly without an adequate and systematic control and still requires attention and research efforts.

Framework for the Characterization of Research into SoS Software Architectures:

Another important contribution of this work is the first version of a framework that enables the characterization of research in the area of SoS Software Architecture, as showed in Figure 2.11. This framework is the result of an iterative process of classification of the primary studies found in our SM and is organized through two main perspectives: *Problem Space* and *Solution Space*. *Problem Space* comprises three main categories: *Quality Attribute*, which encompasses the main quality attributes found in our SM and seems to directly influence on the SoS software architectures; *Characteristic*, which presents the SoS characteristics that affect SoS software architectures; and *Application Domain*, which makes it possible to classify the application domain addressed by SoS as civil or military. *Solution Space* refers to the activities commonly performed to build SoS software architectures: *Architectural representation*, *Architectural evaluation*, *Architectural design*, and *Architectural evolution*. Each of these seven categories comprises subcategories for a more refined characterization of the research developed in the area.

Such framework presents the most common and apparently stable elements/classes for the primary studies found and can be useful for the understanding of a given study or a set of studies. Additionally, it can provide an understanding of a research project or research of a group/institution, or even to identify lack of research in the area of SoS software architecture. This framework must be evolved so as to encompass new lines of research.

In order to clarify a use of this framework, we present an example that characterizes the Oliveira's work (Oliveira and Pereira, 2007), which proposes a reference architecture that supports the development of SoS. This work had a good evaluation in our quality assessment and also contributed to answering almost all RQs (RQ1 to RQ5). Figure 2.12 shows an instantiation of the classes from our framework. This study deals with both solution and problem spaces; moreover, for instance, it addresses architectural design, representation, and evaluation activities for SoS in the civil domain, also encompassing some architecturally relevant characteristics and quality attributes. From this figure, it is possible to observe that, for example, this study does not deal with architectural evolution.

Discussions and Challenges for Future Research:

An SM aims to influence the future direction of a research area (Kitchenham and Charters, 2007). Our SM identified evidences that SoS software architectures are an essential element for the development of software-intensive SoS. As previously stated, it is a relatively new research field and publications are concentrated in the last years. SoS for military domain and other application domains corroborates SoS as a comprehensive, complex class of software-intensive systems. SoS usually involve a number of complex constituent systems, different technologies, and several teams and organizations that apply different approaches to develop these systems. The aforementioned issues are coherent with new perspectives of current software-intensive systems. Although these issues can be also observed in other system classes, their exclusive combination makes SoS particularly challenging for software engineering community. In this scenario, we identify challenges for the future research in Software Architectures for SoS area:

- **Challenge 1: Mapping SoS characteristics for software architectures:** Architecturally relevant SoS characteristics are coherent with the new perspectives of current software-intensive systems. Although these characteristics can be also observed in other classes, their exclusive combination makes SoS a trend for future software-intensive systems, enabling the building of solutions for several highly interconnected systems. Therefore, future research requires a better understanding of how these characteristics can be treated in combination to be incorporated in the architectures of such systems. For instance, how to better manage operational and managerial independent constituent systems in the context of a dynamic architecture and, at the same time, complying with changing missions.

- **Challenge 2: Improvement of the quality of SoS:** quality attributes can vary according to different application domains and this variation is an important issue related to SoS, since these systems can involve several domains with different constituents, organizations, and stakeholders with different interests. Therefore,

their software architectures must encompass complex structures of quality prioritization. The set of main quality attributes obtained in our work is the initial step for research on the quality of SoS software architectures. Furthermore, a quality model for SoS containing quality attributes considered as essential and their relationship could be established. In this same perspective, well-known quality model for software-intensive systems, namely international standard ISO/IEC 25010, can be evolved encompassing quality attributes that may arise. Quality models should also be established for specific types of SoS (e.g., service-oriented SoS and network centred SoS) or domains (e.g., avionics, embedded systems and information systems), since such diversity can influence the importance of each attribute. These models are essential to guide the evaluation, design and evolution of SoS architectures, contributing to the establishment of quality-based processes, methods, and techniques to build such architectures.

- **Challenge 3: Representation of SoS software architectures:** The adoption of architectural views together with a semi-formal approach and UML have been used to represent SoS software architectures. However, there is still room for research in this direction, mainly focused on a clear understanding on how to improve the description of SoS software architectures. A challenge related to such description is the management, dissemination, and integration of different representations among several organizations, stakeholders, and perspectives of interest. Due to the complexity of SoS structure, it can involve different views, complex rules for accessing SoS information and copyright, handle the dynamism and constant evolution, and allows to predict and understand the emergent behaviors in SoS operation. At first, the necessary level of formalism and also the situations for adequately applying each level of formalism must be defined. Formal techniques and languages could be introduced to adequately represent SoS architectures and contribute to an automatized verification and simulation. Furthermore, it is necessary to investigate if existing ADLs are sufficient to represent such architectures and to possibly propose new ADLs or to extend existing ones. As a consequence, a standard of architectural description for SoS could be established so as to improve the understanding of organizations and partners that will integrate or deliver their systems to be part of an SoS. Such ADL could be generic (i.e., destined to any type of SoS) or specific (i.e., destined to a given type or application domains). Additionally, empirical evidences on viability and advantages of semi-formal and formal representations must be obtained.

- **Challenge 4: Evaluation of SoS software architectures:** The inherent dynamism of SoS software architectures requires new approaches that could verify, even at runtime, if changes in the architectures cause their degradation regarding quality attributes, such as interoperability, adaptability, and performance. Existing approaches for the evaluation of dynamic architectures should be investigated and, if possible, reused in the SoS context. Furthermore, processes, methods, and techniques for evaluating those architectures must to be proposed and also widely adopted. Software tools must to be also developed for the automation and optimization of tasks related to evaluation and a considerable reduction in the evaluation efforts.

- **Challenge 5: Design of SoS software architectures:** The first challenge related to architectural design of SoS is related to the establishment of architectural requirements. In monolithic systems, there is often a clear, predefined set of stakeholders concerned with the system under production and hence architectural requirements could be established from them (Bass et al., 2012). In turn, each SoS encompasses a broader range of stakeholders, including stakeholders from both constituent systems and SoS. These stakeholders may have their own interests and a more self-serving perspective of participation that typically occurs in traditional systems (Bellomo and Smith, 2008). Diverse perspectives of interest, sometimes conflicting ones, directly influence architectural decisions during the development of SoS software architectures. For instance, an SoS for the medical area can present diverse capabilities, including those related to monitoring of patients in their houses. Several stakeholders are involved and influence in how SoS must be designed, such as doctors, emergency of a hospital, relatives, and researchers (Petcu and Petrescu, 2010). In another perspective, SoS software architecture includes at least two architectural levels: global architecture and individual architecture of constituents. Despite the independence of constituents, interests of an SoS can interfere in their architectures in different ways according to several variants, such as the SoS categories, level of global authority from SoS, perspective of participation of each constituent, and development context of such SoS. Therefore, several possibilities can be possible to develop each of these architectural levels. SoS designers are challenged to architect more and more complex systems, which present a set of unique characteristics. Despite the initiatives found in our work, efforts for efficiently support design of architectures for software-intensive SoS still remain necessary. New scenarios must be considered in the design of SoS architectures. For example, constituent systems are usually unknown when conceiving the initial architecture. Moreover, these constituents are sometimes legacy systems that were previously developed for a par-

ticular context. In this perspective, an existing open issue is how to facilitate the construction of SoS that contain such constituents. For this, processes encompassing integrated methods, techniques, and guidelines are required. These processes should include common steps presented in most of architecture design processes and should be experimented and also widely adopted by SoS designers. Software tools and environments that support such processes, methods, and techniques are necessary to improve productivity.

- **Challenge 6: Evolution of SoS software architectures:** Most SoS evolve by nature, therefore, their evolution must be distinguished from evolution of a monolithic system. Such an evolution must be considered part of the whole development process supported by proper methods and techniques. It is also important to understand why, how, and when this evolution occur to avoid future architecture degradation. Initiatives that systematize the SoS evolution must be investigated, adapted or proposed, and widely adopted. it is desirable to SoS the possibility of dynamically change/reconfigure their architectures to accommodate changes in their constituents and missions. This polymorphism enables SoS to dynamically change their own structure of collaboration to keep them self-operational upon environmental changes. Due to the operational independence, constituents can unpredictably change their participation at runtime (Bhasin and Hayden, 2008a). Therefore, the architectural dynamism is related to different architectural configurations and architectural strategies to handle it. Integrated environments and tools, mechanisms, and technologies that could transparently evolve SoS are required to manage such dynamic architectures and to deal with complicated SoS evolution challenges.

Software architectures are the key element to the success and quality of software-intensive SoS. This work provided results of an SM on SoS software architectures. These results show that the SoS characteristics and quality attributes can directly influence the conception of architectures for SoS. The way how these architectures have been designed, represented, evaluated, and evolved was also discussed. We found several extension of existing approaches (from software architecture or other areas) to deal with SoS architectures, although most of these extensions are not still completely adequate to software-intensive SoS. Most studies are typically focused on specific application domains; therefore, the existence of general solutions for SoS must be investigated. There are not also consolidated research groups or well-connected communities.

This SM provided also a framework that can characterize works in the SoS software architecture area, aiming at providing a understanding of the existing research and also to identify future research in this area. We believe our results support the advance of the

state-of-the-art of SoS software architectures by identifying new lines of research. Indeed, the eminence of SoS together with the exclusive combination of their characteristics brings challenges of research, requiring attention from practitioners and researchers.

SoS software architecture is a challenging research area, with approximately 75% of the works published in the last 5 years and 90% in the last 10 years. Most of these works raise open-issues after having experimented existing approaches for architecting SoS. There is also no conference focusing software-intensive SoS. The first workshop have occurred in the last few years, such as the International Workshop on Software Engineering for Systems-of-Systems (SESoS). For the future, considerable efforts must be still dedicated to consolidate this area and, as a consequence, make possible high-quality software-intensive SoS for the industry and society.

2.2.3 Architectural Process of SoS: a Systematic Literature Review

A software process can be defined as a roadmap that provides a series of predictable steps to be followed when developing software products (Pressman and Maxim, 2015). Each software development life cycle includes most convenient software processes to handle different development aspects, such as software requirements or software architecture (IEEE Computer Society, 2014). An architectural process consists of conceiving, defining, expressing, documenting, communicating, certifying proper implementation, maintaining, and improving an architecture throughout a system life cycle (ISO/IEC/IEEE, 2011). Because software architecture is essential to any system design, any software development process should include activities related to the architecting process (Bass et al., 2012), and these activities must occur in conformance with whole development process (Garland and Anthony, 2003).

Regarding SoS software architectures, there is a lack of investigation about how their design processes might be. In this perspective, we performed a SLR[4] to leverage the state of the art of these processes. In this study, we also established a set of process requirements to guide the conception of design processes for these systems. This SLR included the following research questions:

- **RQ1:** What are the important steps and artifacts to be considered in the construction of SoS software architectures?
- **RQ2:** What is required for systematizing the construction of SoS software architectures?

[4]Details about the research protocol followed in this SLR and the list of included studies are both presented in Appendix B.

In our research, we used the structure of a general architecting processes presented by Hofmeistter et al. (2007) as a basis to compare the specific processes found for SoS software architectures. In this sense, we first introduce general processes from related literature and further present results of our investigations.

Several works have recognized the value of explicitly considering software architectures in software development processes (Bass et al., 2012; Kruchten, 2003; You-Sheng and Yu-Yun, 2003). In the end of 1990s, Bass (1999) defined an architecture-centric process, focused on architectural requirements in addition to functional requirements. This process presented a set of steps, i.e., requirements identification, creation/selection of the architecture, representation/communication of the architecture, analysis/evaluation of the architecture, and implementation of the system. More recently, Bass et al. (2012) proposed a set of essential activities to be encompassed in any development life cycle. The authors argued that different processes and system context will determine particularities of how each of these activities will be performed (Bass et al., 2012):

Making a system business case: the initial study of the viability to build a system. The expected result is the justification of an investment for a given opportunity;

Understanding the architectural requirements: is the elicitation of requirements that influences the architecture;

Creating or selecting the architecture: construct architecture by taking architectural decisions. It can also include reuse of already existent architectural approaches;

Documenting and communicating the architecture: produce architectural documentation and the subsequent disclosing of this documentation to system stakeholders;

Analyzing or evaluating the architecture: is the verification of the conformance of the architecture to the architectural requirements;

Implementing and testing based on the architecture: refers to the assurance of adequate support for an implementation in conformance with the proposed architecture; and

Ensuring implementation conformance to architecture: is the verification and maintenance of the conformance of the developed system to the proposed architecture.

Hofmeister et al. ((Hofmeister et al., 2007)) also present a comparison of five industrial methods for software architectures: Attribute-Driven Design (ADD) (Bass et al.,

2012); Siemens 4 Views (S4V) (Hofmeister et al., 2000); Rational Unified Process 4 + 1 views (RUP 4+1) (Kruchten, 1995, 2003); Business Architecture Process and Organization (BAPO) (America et al., 2004; Obbink et al., 2000); and Architectural Separation of Concerns (ASC) (Ran, 2000). As a result, this study pointed out similarities and differences among analyzed processes and established a general model comprising the main elements expected on any construction process for software architectures. This model was proposed to support understanding the strengths and weaknesses of different existing methods for software architectures as well as to provide a framework for developing new methods. Figure 2.13 shows this model, including a set of macro activities and generic artifacts in a basic data workflow.

In this model, three basic macro activities are (Hofmeister et al., 2007):

Architectural analysis: analysis of architectural concerns and system context to establish, in terms of architecturally significant requirements, which problems in the system the software architecture can solve;

Architectural synthesis: candidate architectural solutions are proposed to effectively solve the architecturally significant requirements. In this sense, this activity moves from problem to solution space; and

Architectural evaluation: candidate architectural solutions are measured against the architecturally significant requirements. Although multiple iterations are expected, the eventual result of architectural evaluation is a validated architecture. Intermediate results would be the validation or invalidation of candidate architectural solutions. So that the purpose is to verify whether architectural design decisions made are the right ones.

In the same model, five generic artifacts are (Hofmeister et al., 2007):

Architectural concerns: interests that pertain to the development of system architecture. These interests are related to system's architecture and they are important to its development, operation, or any other aspect that is critical to mission accomplishment;

Context: a set of elements or circumstances that influences the system. This set can be characterized as a specification of system environment. Distinction between architectural concerns and context is whether it is specifically related to the system (an architectural concern) or is a general characteristic or goal of the organization or stakeholders (context);

Architecturally Significant Requirements: formalized architectural requirements for software architecture. An architecturally significant requirement is any requirement upon a software system that influences its architecture;

Candidate Architectural Solutions: possible solutions identified to compose the software architecture. Candidate architectural solutions may present alternative solutions, and/or may be partial solutions (i.e., fragments of an architecture). They express alternative design decisions about the software structure; and

Validated architecture: candidate architectural solutions that were evaluated and chosen for the architecturally significant requirements. In addition, these solutions must also be mutually consistent.

In this SLR, we performed in this SLR an investigation of current efforts and trends of the design processes for SoS software architectures. Following, we present results considering the last finished selection of this SLR, which included studies published until December 2015. In this selection, we obtained a set of 13 included studies. However, this study is being updated to produce a new version, including studies published until October 2016. The main contributions of this SLR are the analysis of approaches proposed in the literature on this topic and the establishment of a list of requirements to guide conception of processes in the same direction. In this investigation, it was not possible to conclude that there is a mature, well established approach to support the design of SoS software architectures. Meanwhile, results indicate the existence of new efforts that corroborate the relevance of this topic to the SoS development.

In general, we found several extensions of existing approaches (from software architecture or other areas) to deal with SoS architectures. Most studies are typically focused on specific application domains; therefore, the existence of general solutions for SoS must be investigated. Furthermore, there are no consolidated research groups or well-connected communities. In this investigation, the starting point for studies comparison was the general model proposed by Hofmeister et al. (2007). Considering the aforementioned research questions, we present our findings as follows:

RQ1 – Activities and Artifacts on Construction of SoS Software Architectures

In this RQ we investigate the activities and artifacts proposed for construction of SoS software architectures. The starting point of this investigation was the general model proposed by Hofmeister et al. (2007), which includes high level activities and artifacts common in design processes for any software architecture. In order to investigate and compare these approaches, we mapped approaches from included studies to activities and artifacts proposed in this general model. Furthermore, for each activity or artifact from this general model, a following scale-point was applied: Adherent - 1 point; Not adherent

- 0 point; and Partially adherent - 0.5 point. The total adherence score (AS) for each study fell into the range between: $0 \leq AD \geq 0.3$ (poor); $0.3 > AD \geq 0.7$ (fair); and $0.7 > AD \geq 1.0$ (excellent). Table 2.4 reports the studies adherence. In general, it reveals that main design activities are more considered (i.e., 11 on a total of 13 studies have reached fair or excellent AS) than the main artifacts expected in these processes (i.e., 7 on a total of 13 studies have reached poor AS). The findings regarding the presented scores are detailed as follows.

Table 2.4: Adherence of the proposed approaches to the Hofmeister et al.'s model (Hofmeister et al., 2007)

Activities	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**
Architectural analysis	0.5	0.5	0.0	0.5	0.5	0.5	0.5	0.5	1.0	0.0	0.5	1.0	1.0
Architectural synthesis	0.5	0.5	0.5	0.5	1.0	1.0	0.0	1.0	0.0	0.5	0.5	0.5	1.0
Architectural evaluation	0.0	1.0	0.0	1.0	0.5	0.5	0.5	1.0	1.0	0.5	1.0	0.0	1.0
Adherence	0.67	0.17	0.33	0.83	0.66	1.0	0.66	0.66	0.66	0.66	0.5	0.33	0.5
Artifacts	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**
Context	0.5	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.5
Architectural concerns	0.5	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.5	0.0	0.0	0.0	1.0
Architecturally significant requirements	0.0	1.0	1.0	0.5	0.5	0.5	0.0	1.0	1.0	1.0	0.5	0.0	1.0
Candidate architectural solutions	0.5	1.0	1.0	1.0	0.5	0.5	0.0	0.5	0.0	0.5	1.0	0.5	0.5
Validated architecture	0.5	0.0	0.5	0.5	0.5	0.0	0.0	1.0	0.5	0.5	1.0	0.0	1.0
Adherence	0.42	0.42	0.42	0.33	0.25	0.17	0.25	0.50	0.42	0.33	0.42	0.25	0.58

Architectural analysis on SoS software architectures: the challenges on architectural analysis are often related to the additional complexity of architecting a system that is a result of other independent systems working together but maintaining their own operational context. SoS software architectures are designed upon constant uncertain. For example, constituent systems are usually unknown when conceiving the initial architecture (Nielsen et al., 2015). These constituents are sometimes legacy systems that were previously developed for a particular context and strategies to support this participation must be present since the architectural conception. Therefore, understanding of architectural

requirements in this context is one of these challenges. SoS requires analysis in additional levels, i.e., requirements of SoS and requirements of constituent systems. Furthermore, there is a complex net of interests and influences involving organizations, consumers, normative authorities, and constituent system providers. The complex multi-stakeholders context of SoS requirements is cited as an challenge to be encompassed in the analysis (Ge et al., 2013; Kazman et al., 2012). Some studies proposed the identification of SoS global goals and missions. Griendling and Mavris (2010) argue that the architectural process must start with the leveraging of SoS missions answering the question "What missions need to be performed?". Moreover, identification of architecturally significant requirements architectural requirements must be after derived from established SoS missions, being in accordance with them (Kazman et al., 2012).

Moreover, Ayoama and Tanable (2011) suggest the extraction of SoS properties and their dependencies during architectural analysis having over the assumption that a requirements specification was already done. Other issue is the identification of constituents. Chigani and Balci (2012) propose the leveraging of SoS constituents at architectural analysis. The authors consider a component as any identifiable part of the SoS, i.e., human, software, and hardware. Finally, risks are also pointed in some studies. Kazman et al. (2012) propose activities for risk identification on architectural analysis in which development leaders and SoS stakeholders work together early identifying risks and problematic architectural decisions in a low cost perspective. For this, the study introduces a scaled-up version of a previous technique to engage monolithic systems stakeholders in establishing requirements and analysis risks named Mission Thread Workshop (MTW).

Architectural synthesis on SoS software architectures: in the SoS community, different architectural synthesis strategies have been proposed to accomplish the establishment of architectural decisions in SoS development projects. Ayoama and Tanable (2011) propose a behavior-oriented approach in which a meta-model of behavioral properties guides the architectural design process. In this perspective, the proposed activities related to architectural synthesis are the hierarchical structuring of properties and non-functional requirements and the subsequent design of a service architecture based on this structure. Iacobucci and Mavris (2011) propose the generation of architectural solutions from the analysis of different architecture alternatives. The authors introduce a declarative notation for describing these alternatives for automatic generation . Therefore, the architectural synthesis comprises the allocation of constituent systems to SoS functions and associated metrics for evaluation composing architectural alternatives to be automatically executed and evaluated. Chigani and Balci (2012) describe as activities of architectural synthesis the architecture creation and description. In architectural creation, architectural styles and architectural patterns are selected to make up a composite

architecture. Thus, the authors propose two strategies to design architectures: the selection of pre-existent and well-known architectural solutions; and the architectural design from scratch, if well known solutions do not provide an appropriate structure needed to satisfy the architectural requirements. Next, architectural description is performed in accordance with multiple viewpoints of SoS. Additionally, they recommend viewpoints of DoDAF (DoD, 2010). GonÃ§alves et al. (2015) argue for an incremental proposition of new architectural versions in an evolutionary perspective. The main activities are: (i) elaboration of a new architectural version that is an evolution of the previous one; and (ii) representation of this new version encompassing all the required viewpoints on each SoS context. To support these activities, it is proposed management of multiple providers of constituent systems, support and monitoring of architectural changes to ensure an evolutionary perspective of development, and formulation of strategies for prediction of both desired and undesired emergent behaviors resulting from the architectural changes.

Furthermore, since all the stakeholders must have access to suitable architectural representations to effectively contribute with SoS design, a related challenge refers to the adequate documentation of SoS software architecture. Providing a consistent architectural representation is an universal duty for any architectural process; however, in SoS, context the new challenge is on managing this representation in a complex set of stakeholders and, at the same time, ensuring the consistency of this representation to a system that constantly evolves. In this scenario, it is common on SoS that the architecture be developed and documented in parallel to an operating system evolving at runtime; i.e., the overall structure can be modified by adding, replacing, or withdrawing constituent systems and their relationships. Some studies propose architecture simulation as a solution for this issue(Butterfield et al., 2008; Chen and Kazman, 2012; Ge et al., 2013; Griendling and Mavris, 2010). Among these studies, there is an agreement on the application of this strategy for predicting emergent behaviors in the complex operational structure of SoS. Regarding the employment of technologies and tools from traditional software engineering, we observed some the use and adaptation of approaches to support design processes of SoS software architectures such as MDD techniques (e.g., common modeling languages, model transformations, and frameworks) used on different levels of abstraction (Mensing et al., 2012), SOA as an architectural style in which the constituent systems can collaborate as service providers/consumers (Chigani and Balci, 2012), and Petri-nets applied as a strategy to allow architecture simulation to support design (Ge et al., 2013; Griendling and Mavris, 2010).

Architectural evaluation on SoS software architectures: architectural evaluation must consider the operational dynamism of SoS and its constituents. Liang and Luqi (2003) propose a static design inspection that provides a means of assessing the

quality impact of design decisions before coding. This study performs the architectural evaluation on the basis of a quantifiable architectural view with formalized descriptions attaching constraints to architectural factors, such as roles, styles, and protocols. From another perspective, Griendling and Mavris (2010) propose an evaluation strategy based on simulation activities, in which alternative architectural solutions are experimented upon a set of simulation techniques considering previously established metrics.

Chigani and Balci (2012) describe tree strategies to perform architectural evaluation: (i) evaluation based on product, process, people, and project; (ii) evaluation following a risk-driven approach; and (iii) evaluation based on scenarios. Kazman et al. (2012) propose a series of evaluation sessions based on ATAM to be attended by different sets of stakeholders. The study approach has three phases: preparation, execution, and roll-up/follow-up. In preparation, evaluations are planned establishing evaluation teams and the number of evaluation sessions considering variety and availability of stakeholders. In execution, architects from both constituent systems and SoS presents their architectural approaches and the evaluation team probes it for specific mission threads, guided in particular by a knowledge on architectural design. During roll-up/follow-up phase, the evaluation team identifies problematic areas not properly addressed to explore in more focused architecture evaluations. Ge et al. (2013) propose the use of both static and executable models to perform architectural validation. In this approach, evaluation is based on both static analysis of architectural models and simulation by using executable models for predicting behaviors and explore alternative solutions in the architecture. Finally, Gonçalves et al. (2015 follow an incremental perspective of development proposes the *evaluation and validation of a new architectural version* as central activities related to architectural evaluation in which an explicit validated architecture is the expected output, if it is not reached, the process must return to architectural synthesis instead to deliver an architecture with new increments to be implemented. Additionally, the study also recommends the use of simulation to predict behaviors on architectural evaluation proposing as *predicting emergent behaviors* an activities to support the evaluation work.

Artifacts are essential to methods and processes to support their conduction and provide evidences of their well-execution. Following, we present how the studies encompassed artifacts from the general model proposed by Hofmeister et al. (2007). It is important to mention that there is no strictly disruption among these assets and a process element can express one or more of them at the same time:

Context: since architectural context is related to high level analysis, few studies have mentioned this topic in their architectural process artifacts. The general trend is to allocate analysis and documentation of context to other activities external to architectural process. For example, a pre-established register of context information produced in previ-

ous analysis performed in general SoS engineering process activities (Aoyama and Tanabe, 2011).

Architectural concerns: despite architectural concerns be directly related to the software architecture, SoS studies typically includes architectural concerns only at high level of system engineering analysis.

Architecturally significant requirements: it is a recurrent concern in the investigated studies. Liang and Luqi (2003) propose "conceptual components" which accounts for the stakeholders' requirements, representing for example the computational activities and information flows expected of SoS. Butterfield et al. (2008) propose a requirements baseline representing stakeholders goals and concerns by using UML models, such as use cases, activity diagrams, and state diagrams. Chigani and Balci (2012) propose a list describing functional and non-functional architecturally significant requirements. The authors also point out as essential that this list has a set of potential tradeoffs that may exist among the identified quality characteristics (i.e., the non-functional requirements) to support the architectural decisions when creating the architecture. Kazman et al. ((Kazman et al., 2012)) propose that architecturally significant requirements must be based on business goals and stakeholders concerns. The authors point out that quality attributes perform strongest influence on architectural design and propose the use of *quality attribute scenarios* to form the backbone of everything that will be done in architectural analysis, synthesis, and evaluation. The authors also describe quality attribute scenarios as a description of how a system is required to respond to possible stimulus and provide a set of parts that it must contain, i.e., source; stimulus; system's stimulated part; condition to the stimulus occurrence; system's response; and metric to measure this response. Mensing et al. (2012) propose the concept of "architectural rules" that consists of several properties representing crosscutting architectural concerns. These rules come from the refinement of SoS requirements, i.e., "requirements rules".

Candidate architectural solutions: some studies just indicates a path to leverage and compose the candidate architectural solutions to propose an SoS architecture. For example, Chigani and Balci (2012) suggest as possible solutions for SoS the client-server architecture, distributed objects architecture, peer-to-peer architecture, and service-oriented architecture. If well-known solutions are not adequate to the SoS, the authors recommend composition of an architecture from scratch, also called custom-made or domain-specific architectures. Butterfield et al. (2008) propose the employment of architectural representations that allow further analysis, modeling, and simulation of the proposed architectural solutions. Other studies recommend some candidate architectural solutions to be employed in the composition of an SoS architecture. Griendling and Mavris (2010) follow the same line, proposing a simulation environment based on DoDAF models in which

alternative architectural solutions are resulted from interactions among constituent systems. In order to produce this environment, the authors propose the use of Markov Chains and Petri-Nets, Network Theory, System Dynamics and Discrete Event Simulation, and Agent-based modeling. Lytra and Zdun (2013) proposed an approach that aims to provide semi-automated support for specific recurring candidate architectural solutions and resolve their inherent uncertainty by using specialized fuzzy models to structure and document alternative candidate architectural solutions. The authors use Fuzzy-logic (Zadeh, 1965) to allow the numerical encoding of the vague linguistic values software engineers describing requirements as well as forces and consequences of reusable candidate architectural solutions into a fuzzy model of rules. Therefore, design decision documentation is produced having as inputs the system requirements, information of the fuzzy rules, the actual and the alternative candidate architectural solutions.

Validated architecture: software architecture is naturally present in all studies of our investigation expressed in different ways. Furthermore, the architecture documentation can be done by using different formalism levels (i.e., informal, semi-formal, or formal) covering all required viewpoints necessary to design and evaluate the architecture against the architecturally significant requirements.

RQ2 – Requirements for Systematizing the Construction of SoS Software Architectures

With challenges brought by the construction of SoS software architectures, it is important to identify what is expected from a process for constructing them. In this RQ, we also investigated what is required for systematizing the construction of SoS software architectures. For this, we extracted from the studies of our review the challenges pointed out by the authors for constructing SwS software architectures. As a result, we proposed a list of requirements expressing what any process should satisfy to adequately support the development of SoS software architectures. These requirements are interrelated and were proposed to represent essential issues to be encompassed by any design process for SoS software architectures. When conceiving this list, we considered recurrent and well-justified requirements from the included studies. Despite none single study includes all the identified challenges, these studies acted in a complementary way forming the proposed list. We structured this list by following the same strategy proposed in (Sage and Biemer, 2007), in which requirements are mapped to the SoS characteristics. Following, we present these process requirements (PR), organized according to the SoS characteristics (see Section 2.1.2):

Distribution

PR1. Consider in the architectural design what is necessary to handle the distribution of constituent systems. In case of geographically dispersed stakeholders, provide communication means to allow the architectural design.

Emergent Behavior

PR2. Support prediction analysis and adequate representation of desired and undesired emergent behaviors at any stage of the architecting process.

PR3. Provide means to establish traceability among SoS missions, functionalities, emergent behaviors, and capabilities from constituent systems.

Evolutionary Development

PR4. Provide means to support the architectural evolution in accordance with to SoS development.

PR5. Continuously develop, monitor, update, and refine architectural decisions and respective SoS software architecture.

PR6. Maintain the management of complex range of stakeholders ensuring their involvement in the architectural design during SoS life cycle.

PR7. Establish an architecture documentation that registers the SoS software architecture and its evolution.

PR8. Include strategies to manage, negotiate, and update architectural requirements and their effect in the architecture.

PR9. Deal with quality attributes (e.g., interoperability, connectivity, and performance), providing means to earlier verify these attributes in the architecting process.

PR10. Provide means to handle uncertainties and lack of information that surround SoS development, particularly in the initial stages of its development.

Global Mission

PR11. Allow a mission-oriented design, in which SoS missions guide the SoS architecting process and the influence of individual missions of constituents is also considered.

Managerial Independence of Constituents

PR12. Support inclusion of self-managed constituent systems handling issues in SoS software architecture generated by these constituents, e.g., different organizations and own interests involved, development teams, and different stages of development.

PR13. Include means to handle the lack of detailed information about the internal architecture of constituents.

PR14. Explicit consider the SoS categories when establishing approaches to negotiate with constituent system owners.

Operational Independence of Constituents

PR15. Manage operational impact of constituent systems, which have individual capability of operation and self-regulation. Support dynamic reconfiguration/participation of constituent systems in the SoS, facilitating both operation and evolutionary changes.

PR16. Provide means to monitor and receive continuous feedback from SoS operations and deal with deviations and changes in the operation of constituent systems.

PR17. Explicitly consider SoS categories (virtual, collaborative, acknowledged, and directed) to manage different levels of awareness and operational independence that constituents can assume.

Software Dominance

PR18. Consider impacts and relevance of software in SoS and the relation of software with other architectural layers, e.g., physical and human.

After data extractions and analysis conducted to answer RQs, in this section we present and discuss our findings surrounding included studies. In general, studies point out that traditional software engineering approaches are not sufficient for SoS and, at same time, they propose that some of these approaches can be adapted in the light of SoS challenges.

The main contributions of this SLR are analysis of approaches from literature on this topic and establishment of a list of process requirements to guide future approaches in the same direction. After conducting this review, it is not possible to conclude that there is a mature, well established approach to support design processes of SoS software architectures. Meanwhile, results indicate the existence of new efforts which corroborate the relevance of this topic to SoS development.

In the analysis conducted to answer RQ2, we observed the common argument that new challenges brought by SoS development demands for new solutions from software engineering. These challenges were used as a basis to propose a process requirements list to support process authoring for SoS software architectures. Moreover, in RQ2 we analyzed how each study deals with basic activities and artifacts of a general process model for software architectures. In this analysis, we identified the knowledge from software

engineering of monolithic systems being also employed with some additions proposed to encompass SoS design challenges. Despite main challenges concerning SoS software architectures be recognized in reviewed studies, proposed solutions are not mature and complete and new efforts are clearly necessary. Finally, a general panorama indicates the design of these architectures as a new challenging field for research, which progresses can meet emerging demands of new software-intensive systems.

2.3 Final Remarks

This chapter presented the state of the art for the contributions described in the following chapters. Firstly, characterization of SoS was presented, introducing main concepts related to this class of systems and a conceptual model to support characterization of these systems. Following, the state of the art of both SoS software architectures and their processes were encompassed. Based on results of a SM, we leveraged the main challenges of SoS software architectures and produced a framework for the characterization of research in the SoS software architecture area. Then, we identified in a SLR the limitations and challenges on the current research into the design processes of such systems and produced a set of process requirements to be used in these processes. In this SLR, we conclude that, despite the existence of some design processes applied to SoS software architectures, these systems have been sometimes developed in an ad-hoc manner. Next chapters describe a process that aim at overcoming this lack of a well-structured processes to support development of software architectures in the light of SoS challenges.

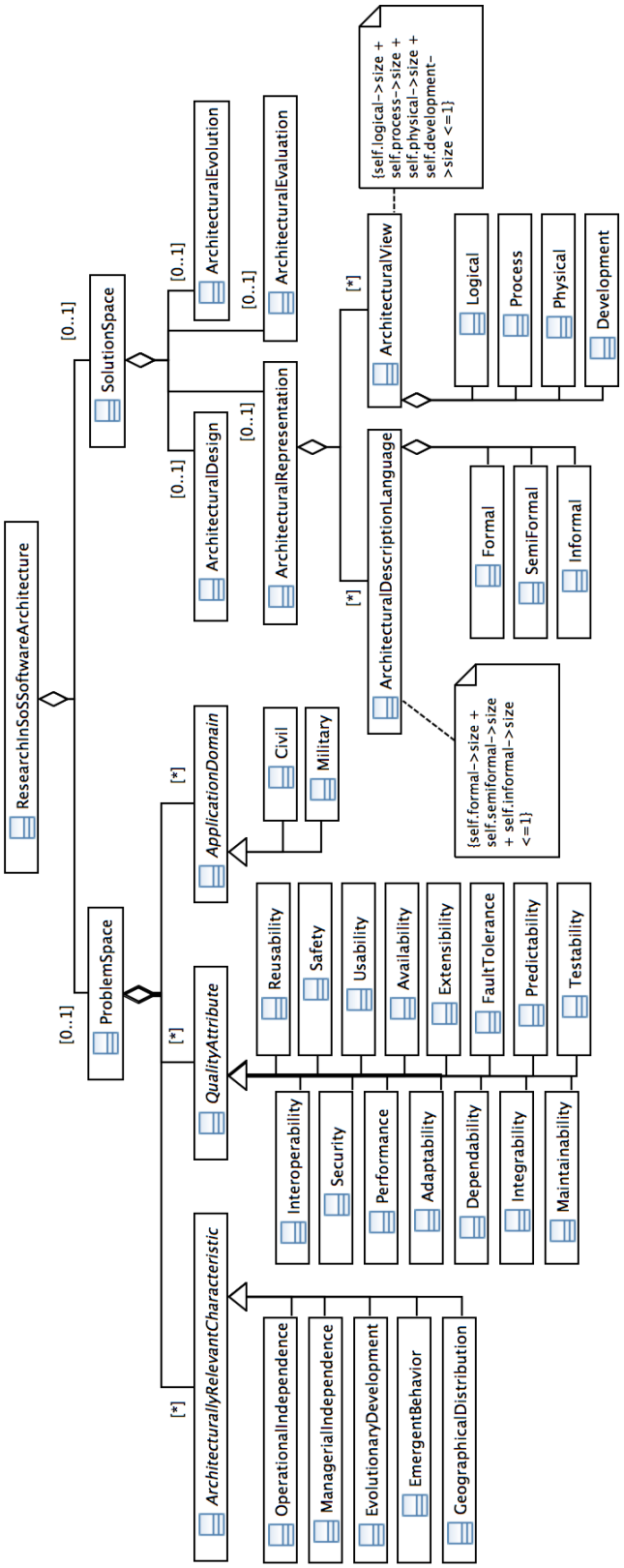


Figure 2.11: Framework for the characterization of research in the SoS software architecture area

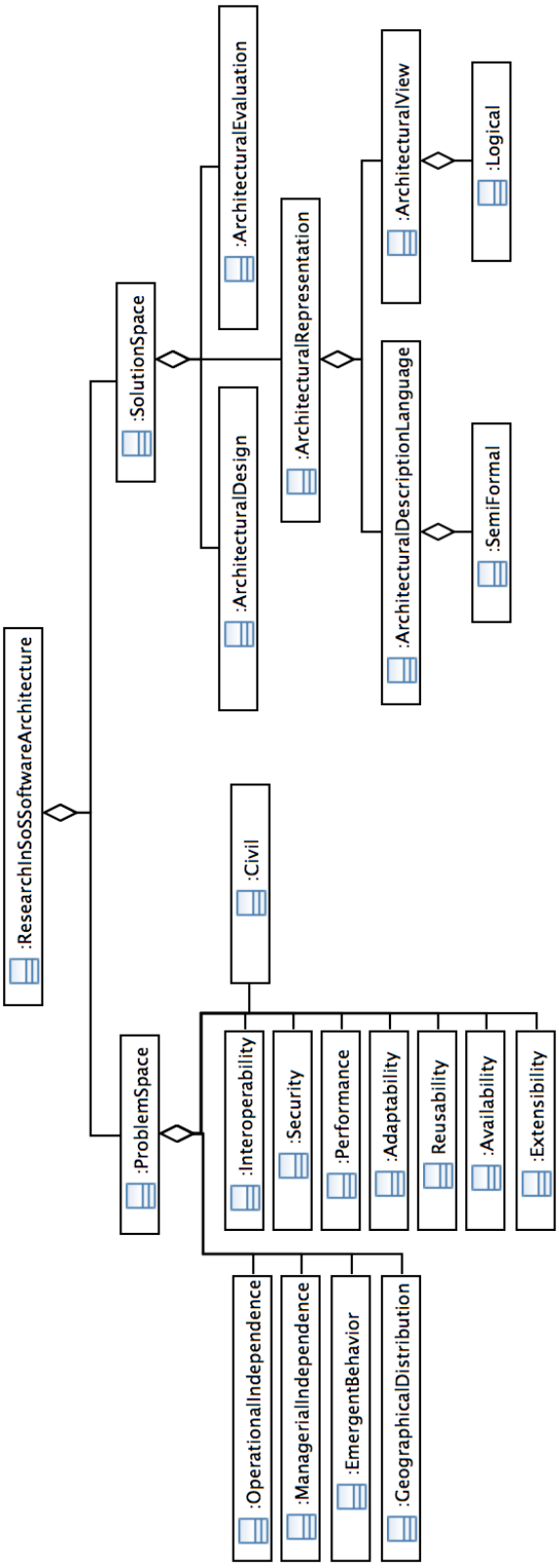


Figure 2.12: Characterization of a primary study

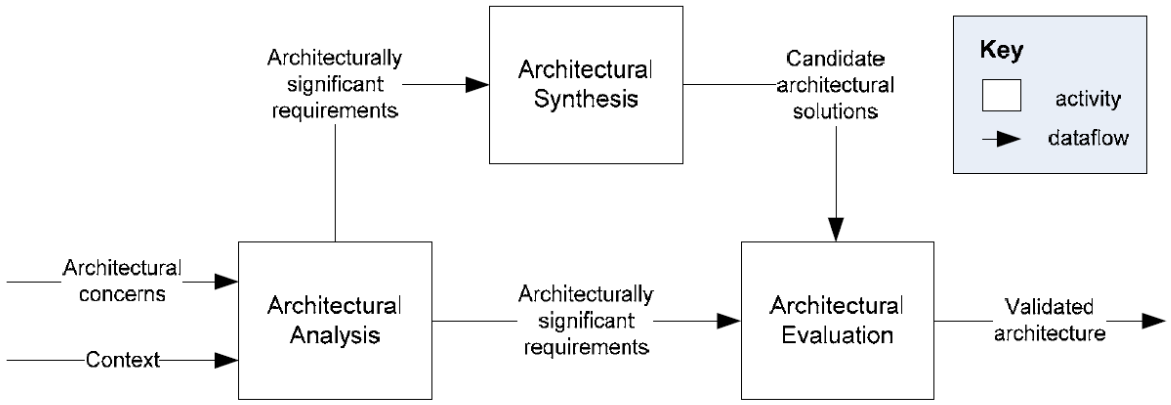


Figure 2.13: Architectural design process reference model (Hofmeister et al., 2007)

Chapter

3

SOAR Kernel: General Approach for Architecting Acknowledged SoS

This chapter presents SOAR Kernel, a part of SOAR process that grounds the design of software architectures for acknowledged SoS[1]. The organization of this chapter is as follows. Section 3.1 describes SOAR Kernel, presenting its final version, i.e., after evaluation by survey with experts through a survey. This evaluation is presented in Section 3.2. Final remarks are presented in Section 3.3.

3.1 Description of SOAR Kernel

To be effective, the architecting activities must directly link business and mission goals, focus on quality attributes, and explicitly involve system stakeholders (Kazman et al., 2012). In order to support the architectual process, facilitating the achievement of these goals, we proposed SOAR Kernel. It describes in a higher level "what must be done" when architecting SoS software architectures. Thus, project teams with well-established practices can directly use SOAR Kernel as a process model to verify enhancement points in their current design processes. The less experienced teams can use SOAR Kernel

[1]SOAR is focused on acknowledged "SiSoS". For sake of simplicity, we use only the term "SoS". Chapter 2 presents more information about the SoS variations and their differences.

combined with more detailed guidelines in SOAR practices described in the next three chapters.

This kernel was initially published in (Gonçalves et al., 2015), being the result from an analysis of the state of the art on SoS in conjunction with knowledge of collaborating experts. It is also in conformance with the macro-activities proposed by Hofmeister et al. (2007) to design software architectures, i.e., architectural analysis, architectural synthesis, and architectural evaluation, and the system engineering process proposed in DoD's Systems Engineering guide for acknowledged SoS (DoD, 2008). The initial conception was supported by three experts of our research group who provided improvements and suggestions to the kernel. In an evaluation stage, external experts participated in a survey.

Figure 3.1 shows the overall structure of SOAR Kernel with its alphas, activity spaces, and competencies as well as its workflow. Despite each project can assume different workflows according to SoS particularities and stage of evolution, the presented workflow is a reference to illustrate a typical flow of execution. SOAR Kernel elements are organized in three main areas of concern, initially proposed in Essence Kernel: *customer*, *solution*, and *endeavor*. In the customer area, the context of the SoS must be understood with an adequate exploration of the opportunities that can be addressed by the software architecture, involving multiple stakeholders of this SoS. In the solution area, architectural solutions must be provided for the SoS. Finally, the endeavor area deals with management of efforts for the architectural process, i.e., establishing, coordinating, distributing, and maintaining a way of work in an architectural development environment. The following sections present these elements of SOAR Kernel, i.e., alphas, activity spaces, and competencies.

3.1.1 SOAR Kernel Alphas: Things to Work with

Figure 3.2 presents SOAR Kernel alphas and relationships among them, expressing the main "things to work with" when constructing SoS software architectures. A total of 12 alphas were established in SOAR Kernel, 11 of which are new ones. The *Stakeholders* alpha is the only one already defined in Essence Kernel[2]. Alphas proposed in SOAR Kernel are described as follows:

Context: It is the set of elements or circumstances that influence the system architecture. This set can be characterized as a specification of the system environment (ISO/IEC/IEEE, 2011), e.g., customer characteristics, missions, and business goals. To define, prioritizing the context elements is quite relevant because everything else in the architectural project should flow from, and be aligned with, these elements (Kazman et al.,

[2]Appendix C summarizes Essence Kernel and its elements.

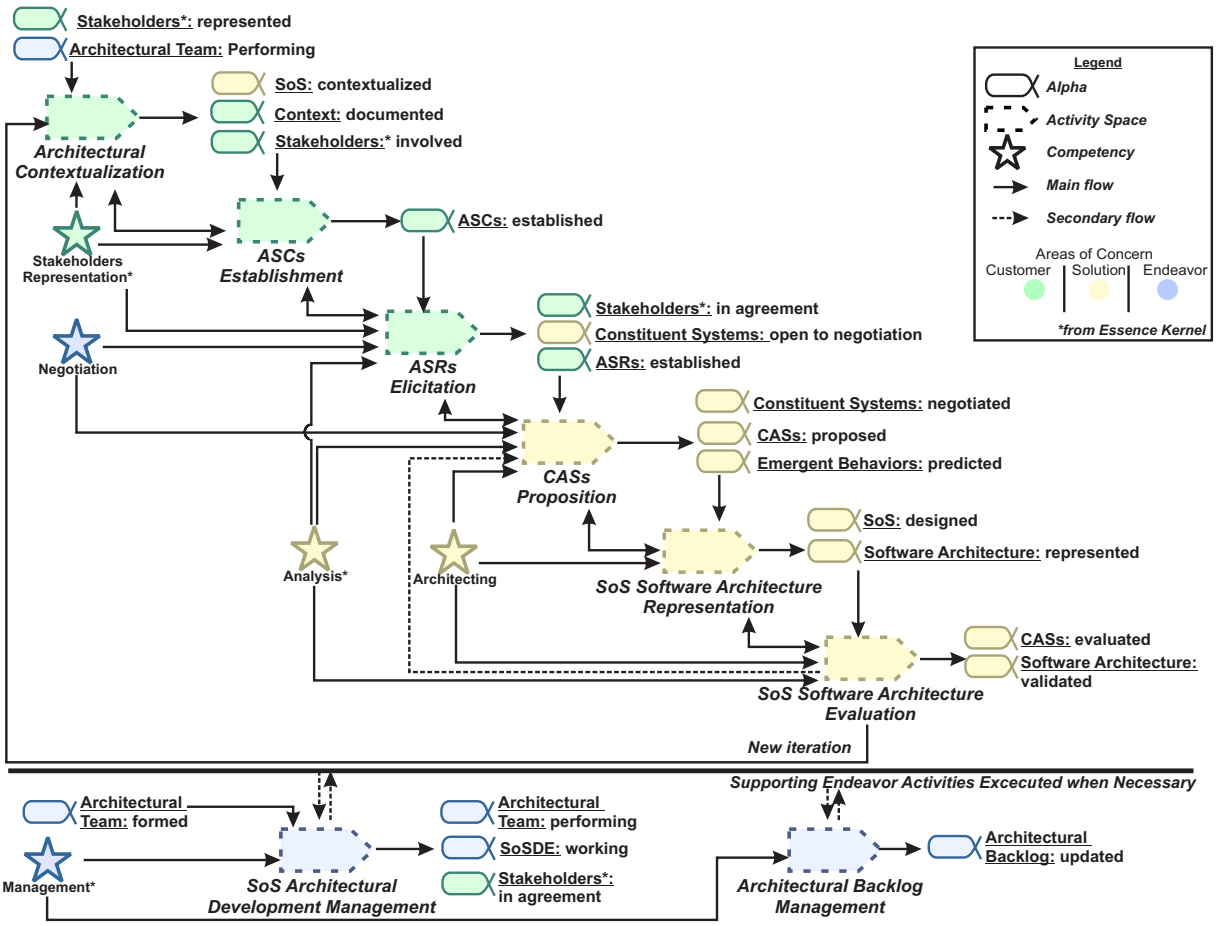


Figure 3.1: Workflow of SOAR Kernel

2012). The Context must be well-defined and documented representing the comprehensive and complex environment of SoS operation and its relation with software architecture.

Stakeholders: This alpha comes from Essence Kernel and represents people, groups, or organizations who affect or are affected by a software system. Stakeholders are a source of requirements for this software system. In SOAR, they must support the architectural team ensuring that an acceptable SoS software architecture is produced.

Architecturally Significant Concerns (ASCs): This alpha represents a specific set of interests pertaining to the development of the SoS software architecture. These interests are related to development, operation, or any other important aspect in the architectural context (ISO/IEC/IEEE, 2011), e.g., architectural patterns or previously agreed design decisions. In this sense, distinction between ASCs and Context is that the former are specifically related to the system and its architecture, while the latter includes elements to support the understanding of the operation environment.

SoS Architecturally Significant Requirements (ASRs): An ASR is any functional or non-functional requirement that is relevant for the SoS software architecture and therefore drives the architectural design. In SoS, ASRs are often related to qual-

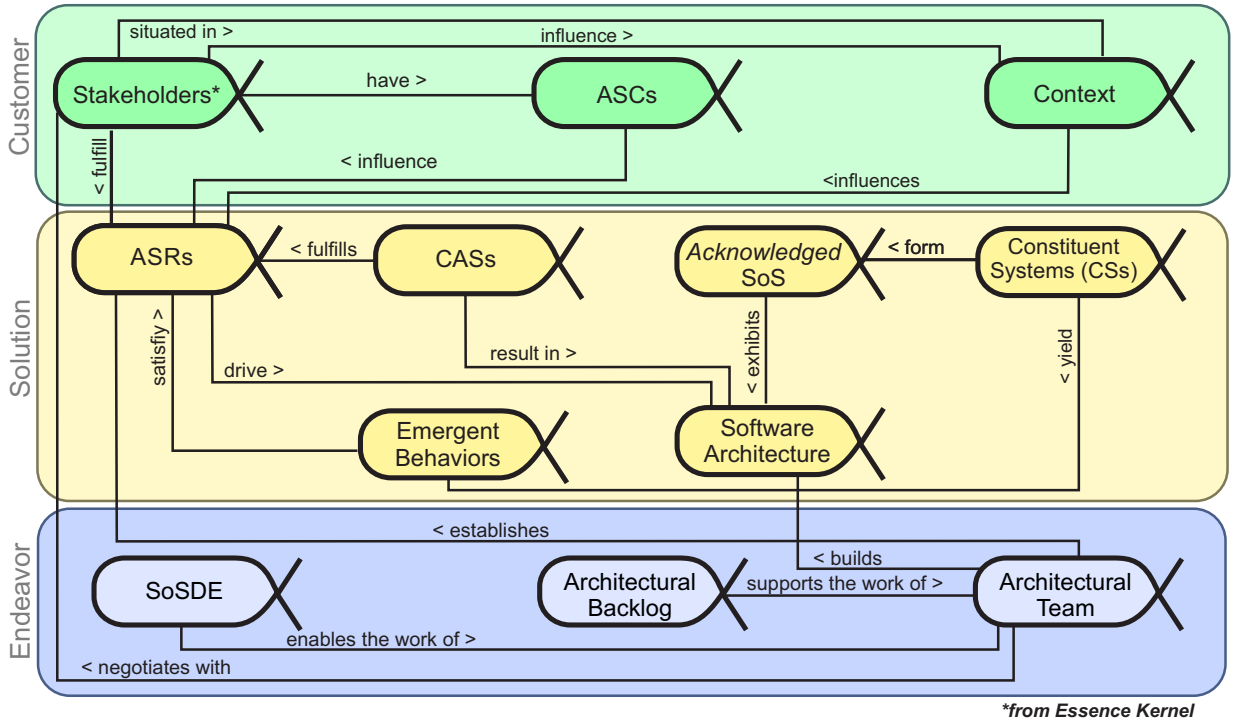


Figure 3.2: Alphas of SOAR Kernel and their relations

ity attributes, constraints, and requirements derived from environmental conditions (i.e., Context) (Chen et al., 2013). These requirements are obtained after agreement through different Stakeholders to be further handled by the architecture. Additionally, quality attributes exert the strongest influence on architectural process and must be considered in the ASRs documentation (Bass et al., 2012; Hofmeister et al., 2007).

Candidate Architectural Solutions (CASs): This alpha represents possible solutions or partial solutions (i.e., fragments of an architecture) proposed to compose the SoS software architecture. Solutions are mainly proposed to meet ASRs reflecting design decisions about the SoS software architecture.

Emergent Behaviors: In an SoS, emergent behaviors result from the collaborative work of constituent systems. Some emergent behaviors can be either foreseen, i.e., they can be determined by specifying interactions among constituent systems or representing interaction patterns (the ways in which they interact), or unforeseen, i.e., they can dynamically and unplanned appear during SoS operation. Both foreseen and unforeseen emergent behaviors may be desirable or even undesirable, so that the result of the interactions among constituent systems within an SoS can be respectively positive or negative over its operation (Holland, 2007). In general, predicted/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.

SoS Software Architecture: It is a software structure (or a set of structures) of SoS comprising constituent systems, their externally visible properties, and relationships among them. An SoS software architecture encompasses concepts, properties, specifications, design principles, and design decisions and patterns of SoS and its environment.

Constituent Systems: they are systems that together contribute to the accomplishment of the SoS mission. Constituents operate independently, having their own missions and resources. In acknowledged SoS, each constituent has its own software architecture that typically is not visible, or accessible to changes at SoS development level. Therefore, any architectural characteristic at constituent systems level must be negotiated with the constituent owners.

Acknowledged System-of-Systems (SoS): It is a complex system resulted from the interoperation of other independent and heterogeneous systems. The collaborative work of these constituent systems yields emergent functionalities to accomplish SoS missions. In acknowledged SoS, 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 a central managed purpose (DoD, 2008).

***SoS Development Environment*:** This alpha expresses the complex development environment surrounding SoS, that is often distributed since several distinct teams can collaborate in the SoS development and evolution. Moreover, developers of constituent systems can influence the SoS development in different ways, e.g., by also being part of SoS teams or merely as constituent systems providers. In this context, the environment where architectural team collaborate and interwork must be managed for each SoS.

Architectural Backlog: It is a transversal knowledge repository that must be used to support the architectural development process. Its function is to encompass any relevant matter related to SoS software architecture not predicted in the regular work products in the project, e.g., considerations for SoS architecture enhancement provided by outsiders from architecting process as constituent systems developers, new ideas for further viability analysis, and registration of changes in the system architecture made during the implementation and not immediately reflected in the architectural description. Due to the transversal approach of this alpha, related work products can be consulted or updated for authorized SoS developers at any time of development process.

Architectural Team: A group of people with strong technical background and actively engaged in the development of the software architecture, making key design decisions. This team plans and performs work needed to create, maintain, represent, simulate, evaluate, and evolve software architectures. Moreover, in small projects this team can be

reduced to one individual, the software architect, who assumes all aforementioned responsibilities (Garland and Anthony, 2003).

3.1.2 SOAR Kernel Activity Spaces: Things to do

In SOAR Kernel, activity spaces express "what must be done" when designing software architectures for acknowledged SoS. Figure 3.3 shows these activity spaces organized by area of concern. A total of eight alphas were proposed in SOAR Kernel that are described as follows:

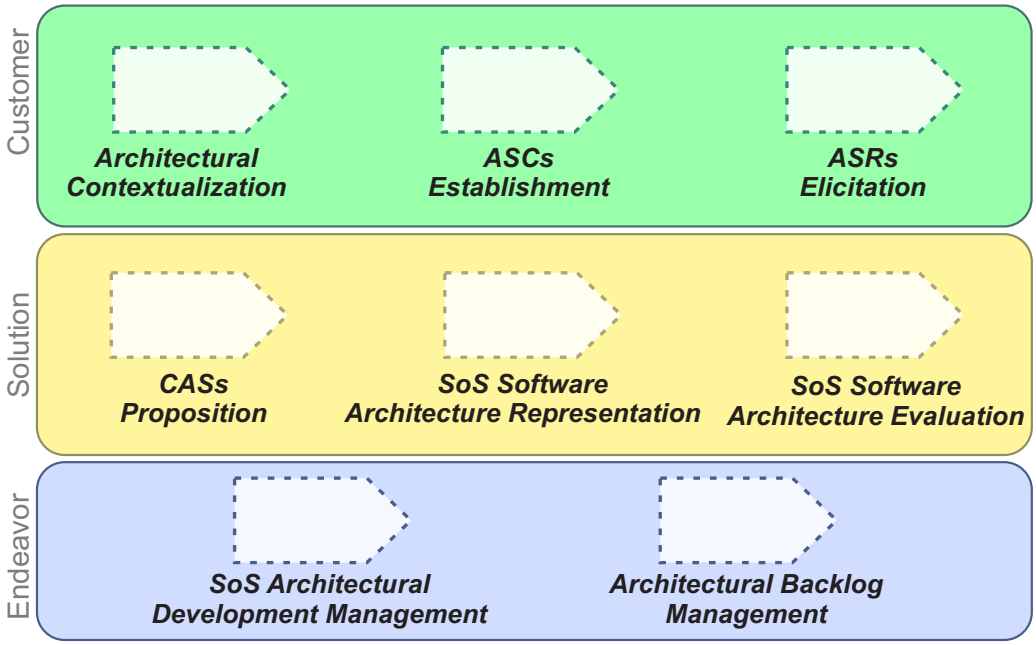


Figure 3.3: Activity spaces of SOAR Kernel

Architectural Contextualization: Due to the SoS multistakeholders environment, there is a high amount of uncertainty that surrounds such system, not only in a technical sense, but also in organizational and business senses. This activity space must have activities to understand, update, and document the SoS Context in each iteration. It must include establishment of SoS global missions and all mission-related information. Moreover, Stakeholders must also be identified and agree with the Context leveraged for such SoS.

ASCs Establishment: ASCs must be documented to express SoS concerns of different Stakeholders. This activity space refers to the identification and understanding of which ASCs are related to and influence the SoS software architecture by considering: (i) context; (ii) different Stakeholders; and (iii) SoS architecture itself in its current state of development.

ASRs Elicitation: The main purpose of this activity space is to define and document problems that SoS software architecture must solve in terms of ASRs. For this, examination of Context and ASCs, and negotiation with Stakeholders to come up with SoS ASRs must be considered. Common ambiguities, different vocabularies, and stakeholders thinking about ASRs must be aligned. Additionally, anticipating changes must help on avoiding restrictive design decisions and support evolution (Sommerville, 2009). Therefore, more evident emergent behaviors can be considered in ASRs elicitation activities as the first step to deal with this issue in the software architecture. Finally, the software architecture can itself be a source of requirements. In this sense, new ASRs resulted from the software architecture must be also identified.

CASs Proposition: In this activity space, CASs are proposed to meet a set of ASRs. The main expected result of this activity space is a new architectural version to be further evaluated. In the context of SoS with emergent behaviors, this activity space must provide means of predicting SoS emergent behaviors.

SoS Software Architecture Representation: 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 software level), representation must focus on software while maintaining the compatibility with more abstract representations.

SoS Software Architecture Evaluation: The main purpose of this activity space is to verify if the proposed CASs are the right ones. The SoS software architecture must be verified against ASRs and ASCs. Although multiple iterations are expected, the result is a validated architecture. If CASs were verified as not adequate, activities of CASs proposition activity space must be performed again. Furthermore, this activity space must also include activities to adequately convey the architecture for developers and stakeholders.

SoS Architectural Development Management: This activity space must support architectural team working with different stakeholders and constituent systems providers. It is related to management of the complex and multi-stakeholders environment typically found in SoS.This activity space must encompass required planning/support for this collaborative work through heterogeneous customers and project teams. It must include not only negotiation of requirements with multiple stakeholders, but also negotiation of capabilities and changes with providers of constituent systems. This scenario is even more complex since SoS clients may simultaneously be constituent systems providers. The man-

agement of this environment must consider that stakeholders can naturally have a more or less self-serving perspectives of participation in the SoS and the collaborative environment must be managed focusing in the SoS missions (Bellomo and Smith, 2008).

Architectural Backlog Management: As a knowledge repository, Architectural Backlog is continuously used throughout diverse architectural development activities.

3.1.3 SOAR Kernel Competencies: Required Skills

SOAR Kernel competencies express "Required Skills" when constructing SoS software architectures. As these competencies describe issues to be considered in human resources, overlaps naturally occur among different competencies. Figure 3.4 shows these competencies organized by area of concern. A total of six competencies were included in SOAR Kernel, two of which are new ones. Competencies already provided by the Essence Kernel are: *Analysis*, *Development*, *Leadership*, *Management*, and *Stakeholders Representation*. Following, we describe these competencies and the new ones proposed in SOAR Kernel:

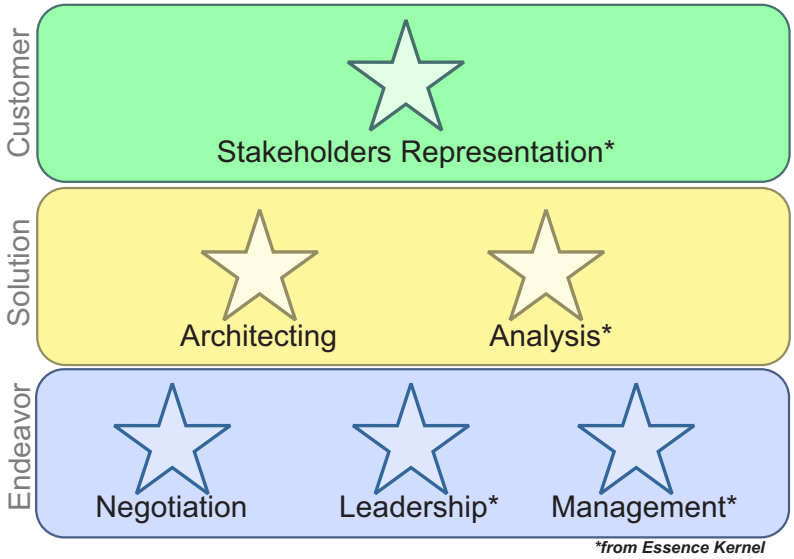


Figure 3.4: Competencies of SOAR Kernel

Analysis: This competency refers to the ability to understand the context and problems, identify solutions, and evaluate their applicability by applying logical thinking.

Leadership: This competency describes the ability of inspiring and motivating a group of people to interact and work achieving project goals.

Management: This competency refers to the administrative and organizational ability that allows to plan, coordinate, and control people's work maximizing the chances of success in a project.

Stakeholders Representation: This competency is the ability to understand, negotiate, and convey opinions and interests of other stakeholders.

Architecting: This competency encapsulates the ability to design and represent effective software architectures, following standards and norms agreed by the architectural team. This architecting competency is about solving complex problems and producing effective software architectures. It requires capability to exploit all knowledge about Context, ASCs, and ASRs and balance them by creating appropriate CASs. Therefore, it is a combination of talent, experience, knowledge, and design skills to develop and maintain the software architecture as expected.

Negotiation: This competency encapsulates the ability to negotiate and reach agreement among different stakeholders in a heterogeneous environment of SoS. It is a combination of technical and personal skills to avoid and manage conflicts reaching best agreements to SoS development. This ability is essential to perform activities in SoS Environment Management activity space.

3.2 Evaluation of SOAR Kernel

As SOAR Kernel is in a higher abstraction level that acts grounding the more concrete SOAR practices, we decided to conduct a survey with experts in correlated fields to evaluate and enhance SOAR Kernel. In summary, a survey is an approach for collecting information to gain insights into a subject under study (Kasunic, 2005). The qualitative survey presented in this section was conducted with a team formed by five experts external to our research group. This survey followed the steps proposed by Kasunic (2005): (i) identify research objectives; (ii) identify and characterize target audience; (iii) design sampling plan; (iv) design and write questionnaire; (v) pilot test questionnaire; (vi) distribute questionnaire; and (vii) analyze results and write report. These steps are described as follows:

1. **Identify the research objectives:** This survey aimed to verify if SOAR Kernel meets the expectations of the SoS community as an approach to support the construction of SoS software architectures. Five research questions (RQs), summarized in Table 3.1, guided this survey.

2. **Identify and characterize target audience:** The survey target audience is represented by potential users of SOAR Kernel, i.e., members of SoS community on both academy and industry. To obtain a sample of this population as representative as possible, this population was sampled considering SoS researchers who have conducted studies on SoS and developers who have constructed SoS. Other aspects such as the level of expertise were also collected to support further analysis of answers.

Table 3.1: Survey research questions

SOAR Kernel	
RQ1	Is SOAR Kernel complete for what it is proposed?
RQ2	Is SOAR Kernel correct, with no wrong or misunderstood statements?
RQ3	Is SOAR Kernel conceptually coherent, with no conflicts or wrong placed elements?
RQ4	Can SOAR Kernel be considered intelligible, well-organized, concise, helpful, and easy to use?

3. **Design sampling plan:** Due to the low availability of experts in SoS software architectures, the sampling strategy was non-probabilistic and for convenience, i.e., to invite a number of experts as largest as possible. For this reason, sample size was limited by the number of individuals who agreed to participate. Experts who participated as survey respondents are involved to development of SoS and software architectures in both academy and industry.

4. **Design and write questionnaire:** Based on the general RQs, the chosen approach for gathering data was to use an online questionnaire[3] combining discursive and non-discursive scale-based questions. Additionally, participants were provided with: (i) guidelines to read documentation and answer questionnaire; (ii) a profile questionnaire to verify the level of expertise of the participants; (iii) a documentation comprising a complete description of SOAR Kernel; and (iv) a support documentation about Essence Language and Essence Kernel.

5. **Pilot test questionnaire:** A initial survey pilot was executed with two researchers from our research group. Data were collected only for verifying errors, average time for response, and possible enhancements in the first version of survey material.

6. **Distribute questionnaire:** After the pilot, an invitation was sent to experts and the survey was sent to whom has agreed to participate. Each participant followed three steps when answering the questionnaire: (i) answering questions about level of expertise;(ii) reading the survey's documentation; and (iii) answering questions about SOAR Kernel.

7. **Analyze results and write report:** For non-dircursive scale-based questions, graphics were associated to a textual discussion to illustrate observed trends in the experts' opinions. Due to the existence of few experts in SoS software architectures (and consequently the low number of survey participants) no statistical test were applied. Furthermore, discursive questions provided insights and open suggestions to

[3]The questionnaires of all surveys of this Thesis are included in Appendix E

enhance SOAR Kernel. The analysis strategy for these questions was the presentation of results through narrative compilations of answers and additional discussions of our observations. Section 3.2.1 presents obtained results and discuss how they help to enhance our proposal.

3.2.1 Analysis and Intepretation of Results

The first part of questionnaire collected the participants' profile through three questions. Figure 3.5 summarizes the level of expertise of the participants in both SoS and software architecture. Regarding the role of each participants, i.e., industry practitioner, academy researcher or both, one participant is an industry practitioner, three are academy researchers, and one is both. Results for each RQ are described as follows.

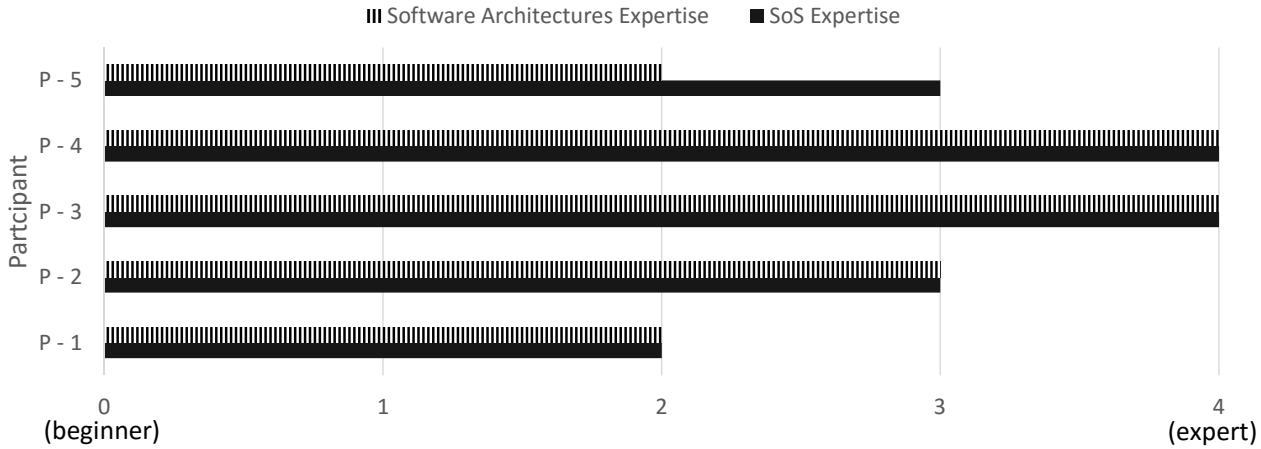


Figure 3.5: Levels of expertise on SOAR Kernel survey

RQ1. This RQ was structured into four questions for the participants, two of them discursive and three non-discursive ones. Figure 3.6 shows results for the first non-discursive subquestion evaluating if SOAR Kernel encompasses all general aspects that are essential when constructing SoS software architectures. Results indicate the general acceptance of SOAR Kernel for this aspect. Additionally, the first discursive question asked the participants for pointing out aspects, concepts, and elements that might be missing. Answers revealed important additions to SOAR: need of encompassing SoS multi-stakeholders environment and the need of explicitly describing the "mission" concept in SOAR Kernel.

Next two non-discursive questions verified if alphas and activity spaces of SOAR Kernel are sufficient to encompass "things to work with" and "what must be done" when constructing SoS software architectures. Figure 3.7 shows results indicating

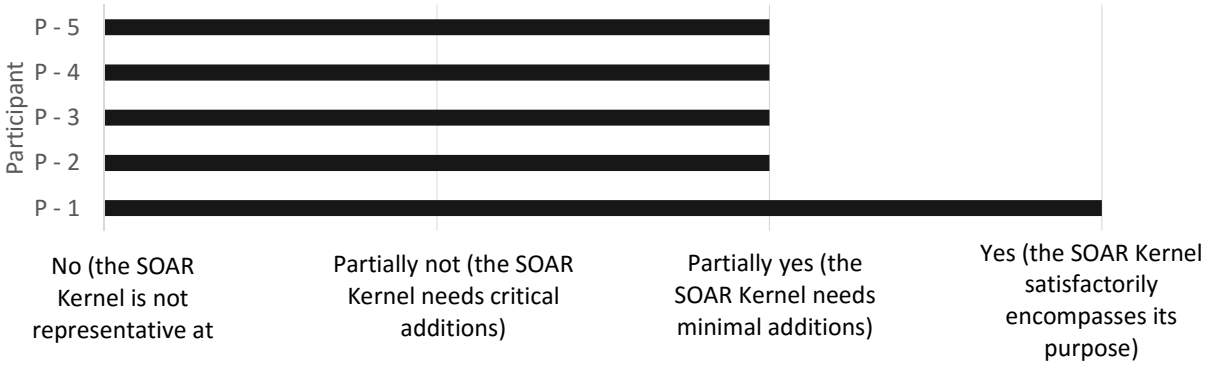


Figure 3.6: Evaluation of SOAR Kernel completeness

a general acceptance of these elements in SOAR Kernel. Additionally, the last discursive question was open to receive insights and suggestions to enhance the sets of alphas and activity spaces in SOAR Kernel. Experts provided important suggestions resulting in some additions to SOAR Kernel: *Distributed Development Environment* and *Emergent Behaviors* as alphas, and *Architectural Backlog Management* as a new activity space. Furthermore, experts pointed that different types of emergent behaviors for SoS were not initially covered. This issue was solved by adding a description of emergent behaviors in the *Emergent Behaviors* new alpha.

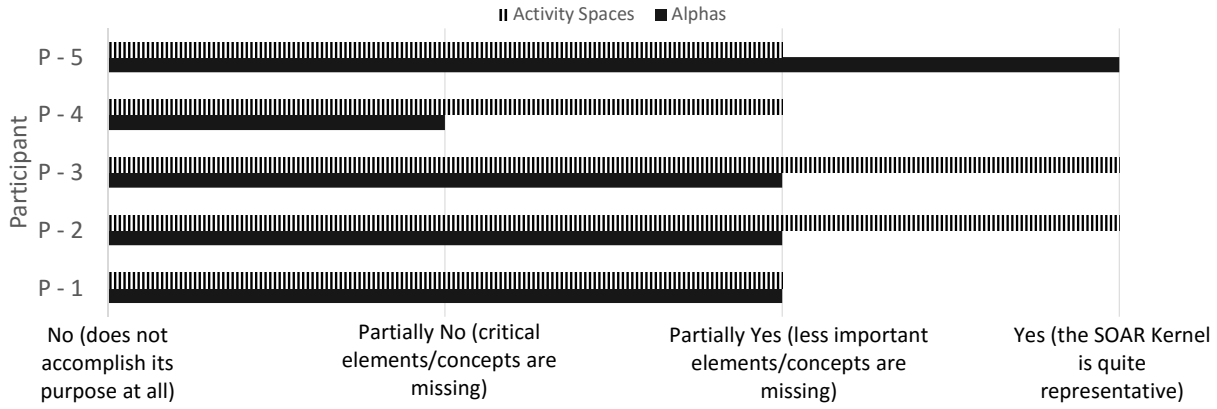


Figure 3.7: Evaluation of completeness of alphas and activity spaces in SOAR Kernel

RQ2. This RQ was decomposed in two questions (a non-discursive and a discursive one) both aiming at evaluating SOAR Kernel in terms of correctness. Figure 3.8 shows results of the non-discursive question indicating the acceptance of SOAR Kernel as correct, but with possibility of corrections. The discursive question asked participants for justifying their grades in the previous question and indicating possible errors for further corrections. Participants' responses helped on finding errors and misunderstandings throughout SOAR. In general, experts found only minor errors in the provided SOAR documentation, such as addition of some possible flows be-

tween activity spaces and correction of names in some elements, i.e., activity spaces, alphas, and alpha states.

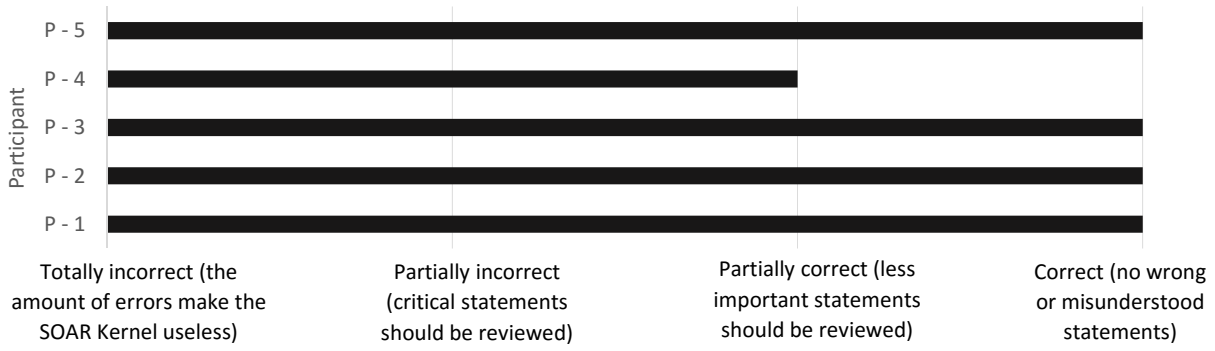


Figure 3.8: Impression level concerning SoS Kernel alphas and activity spaces

RQ3. This RQ was decomposed in two questions (a non-discursive and a discursive one) both aiming at evaluating SOAR Kernel in terms of coherence. Figure 3.9 shows results of non-discursive question indicating the general acceptance of SOAR Kernel as coherent. The discursive question asked participants for justifying their grades in the previous question and indicating problems regarding coherence in SOAR Kernel. In general, participants found only minor coherence problems in the provided SOAR documentation, such as divergences in alpha states between subsequent activity spaces.

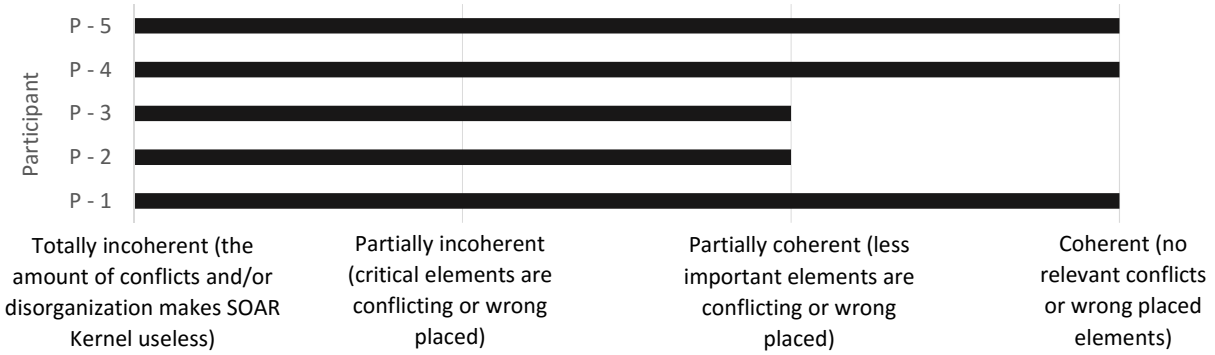


Figure 3.9: Coherence evaluation of SOAR Kernel

RQ4. This RQ was structured in one discursive and three non-discursive questions, each one of non-discursive questions evaluating one aspect of the usability of SOAR Kernel. We considered only usability aspects possible to determine when reading kernel documentation. Figure 3.10 presents results of the non-discursive questions. In general, SOAR Kernel was considered as helpful, well-structured, and intelligible. The discursive question had a complementary purpose asking subjects to point out any

difficulty that they had to understand the kernel and new concepts brought with Essence Language. This question helped on identifying possible improvements in terms of intelligibility. Participants suggested important additions, such as extra diagrams (i.e., the ones not already determined by Essence Approach) to better illustrate SOAR activity spaces. As Essence Language accepts complementary diagrams, the workflow illustrated previously in Figure 3.1 was added to facilitate the comprehension of the kernel approach. Another mentioned point was the difficulty of understanding relationship among different kernels (i.e., Essence Kernel and SOAR Kernel) and how the instantiated process can be executed to the SoS development. These problems were handled in the complete documentation of SOAR with additional explanations, diagrams, and guidelines for use.

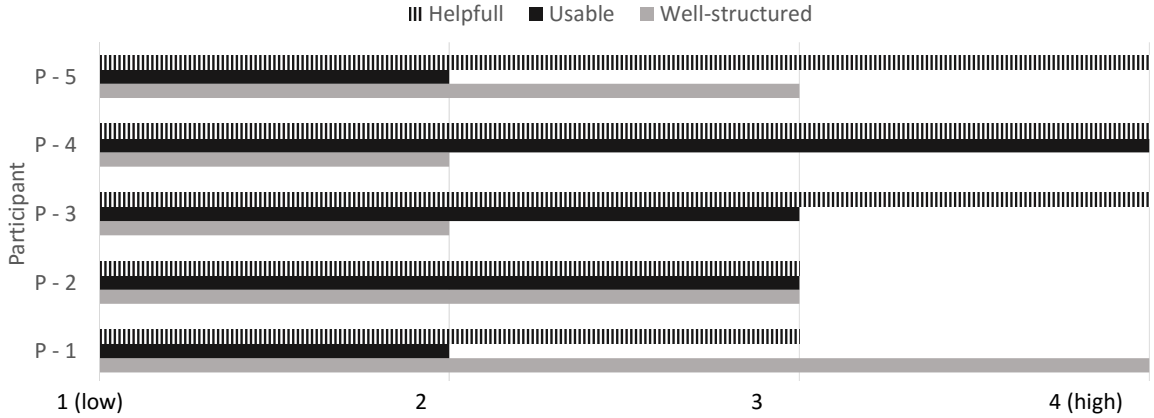


Figure 3.10: Usability evaluation of SoS Kernel

Table 3.2 summarizes results obtained with answers of non-discursive questions. For each RQ, column "Yes" shows percentage of experts favorable to SOAR Kernel in their answers, and column "No" shows the unfavorable ones. Collected answers showed us that in general SOAR Kernel is clear, its elements are described without ambiguities (86.6% of positive feedback for coherence), and provided documentation is enough to enable evaluation team to understand concepts and technical terms (68.8% of positive feedback for usability).

Table 3.2: Survey results of non-discursive questions

Research Question	Non-discursive questions	Yes (%)	No (%)
RQ1	3	73.3	27.6
RQ2	1	93.3	6.7
RQ3	1	86.6	13.4
RQ4	2	68.8	31.2

In addition to proposed questions, a field was left open to suggestions or insights about any issue not properly covered in questions. Concerning these suggestions, initially we

only adopted competencies already provided by Essence Kernel, however, participants suggested specific skills related to SoS software architectures. These suggestions were analyzed by considering the assumptions from SoS literature. With this analysis, was possible to identify new skills that cannot be encompassed by any competency already defined in Essence Kernel. As a result, we conceived new competencies proposed in Section 3.1.3.

3.2.2 Threats to Validity

The main threats to the validity in this survey are related to:

Internal Validity: Internal validity is mainly concerned with unknown factors that may influence the results. In this sense, survey questionnaire itself could induce the results. To mitigate this, a pilot version of questionnaire was applied with two researchers that were asked to pursue for biased questions. This pilot also helped to make adjustments on contents and establish an affordable time to answer the questions.

External Validity: External validity is related to claims for the generality of the presented results. To ensure reliability, none of the participants were members of our research group or of groups related to the authors of SOAR Kernel. Furthermore, answers were received with no identification of their respondents.

Reliability Validity: Reliability validity refers to the possibility of replicating the study. In order to enable this possibility, the survey was designed by following a well defined and accepted methodology (Kasunic, 2005; Shull et al., 2008). Moreover, survey protocol and applied questionnaire were made publicly available.

Construct Validity: Construct validity focuses on correct interpretation and measurement of perceptions, i.e., the relationship between concepts and theories behind the study and what is actually measured and affected. In this perspective, discursive questions could yield different interpretations. To ensure validity in the interpretation, answers and consequent improvements were discussed in our research group responsible for SOAR Kernel.

3.3 Final Remarks

This chapter presented SOAR Kernel that supports construction of acknowledged SoS software architectures. Contributions of this chapter are: (i) a kernel that provides a general grounding when constructing acknowledged SoS software architectures; (ii) a representation of SOAR Kernel in Essence Language built in the EssWork Practice Workbench that enables instantiation of processes for particular projects; (iii) a survey with experts to evaluate SOAR Kernel that indicated that SOAR Kernel is adequate for con-

struction of acknowledged SoS software architectures. Grounded by SOAR Kernel, next three chapters present three practices for, respectively, architectural analysis, architectural synthesis, and architectural evaluation. These practices describe activities and work products to design acknowledged SoS software architectures, offering a lower-level support to complete the goals of SOAR.

Chapter

4

SOAR-A: Architectural Analysis on Acknowledged SoS

This chapter introduces SOAR-A, a SOAR practice that supports the establishment of process instances for architectural analysis of acknowledged SoS software architectures. SOAR-A reflects the state of the art for this issue discussed in Chapter 2. Section 4.1 describes SOAR-A and its elements, i.e., activities and work products. This description refers to the final version of SOAR-A produced after a survey conducted to evaluate it that is presented in Section 4.2. Final remarks are presented in Section 4.3.

4.1 Description of SOAR-A

Architectural analysis is the initial phase, or macro-activity, in architecting processes, in which problems that the architecture must solve are defined as architectural requirements, i.e., ASRs(Hofmeister et al., 2007). In SoS, complexity of this analysis is increased by the scale and heterogeneity of a multi-stakeholders environment, involving multiple organizations, consumers, and providers of constituent systems. SOAR-A is a practice to guide architectural analysis of acknowledged SoS by providing activities and work products that can be adopted by project teams in their own development processes. As a part of SOAR, it is also represented with the Essence language by using Essence Workbench Tool. SOAR Kernel (see Chapter 3) grounds SOAR-A by providing *activity spaces*, *alphas*, and *compe-*

tencies expressing "what must be done". SOAR-A in turn defines essential activities and work products to describe "how to" when eliciting and maintaining ASRs. When using SOAR to build process instances, project teams can assemble the more suitable set of practices based on its development context.

In Essence Language, the main elements of practices are *activities* and *work products*, which are proposed to encompass *alphas* and *activity spaces*. Activities define approaches to accomplish activity spaces by providing more concrete guidelines to reach pre-defined alpha states. A work product is an artifact that concretely represents an alpha, e.g., specific types of document. SOAR-A is in the sense that other activities and work products specific to each SoS development project can be developed and added. Following, activities, alpha states, and work products proposed in SOAR-A are described.

4.1.1 SOAR-A Activities

Figure 4.1 shows the workflow diagram of SOAR-A, which includes a set of activities and related activity spaces inherited from SOAR Kernel. SOAR-A defines a total of eight activities accomplishing a total of five activity spaces from SOAR Kernel: SoS architectural development management, architectural backlog management, architectural contextualization, ASCs establishment, and ASRs elicitation. These activities are organized in a workflow that illustrates both the main flow, with the essential sequence of activities to be followed in every iteration, and a secondary flow, with activities that must be considered when convenient in each development context.

SOAR was conceived to be performed in alignment with other development processes in progress on each SoS development project. These processes are both external sources of information to SOAR and consumers of the architectural information generated by it. In this perspective, the entry-point for SOAR-A is the general SoS context that must be understood at systems engineering level. This context should be externally provided and previously available from architecting process working as an input for architecting process. Furthermore, due to different project scales and organizational situations that each SoS development environment can assume, we do not describe more specific details, such as the number of people to integrate each group, a set of specific roles. In this sense, tasks must be defined in specific process instances, encompassing particularities of each SoS under development. Activities in SOAR-A are described as follows:

Planning Analysis: The way to perform architectural analysis changes on each iteration of architecting process. This activity must establish/update a plan (see the work product *Analysis Plan*) to deal with these changes maintaining an adequate execution of analysis. Architectural team is responsible for this activity and SoS stakeholders can contribute with any information necessary to the definition of plan details. Even with

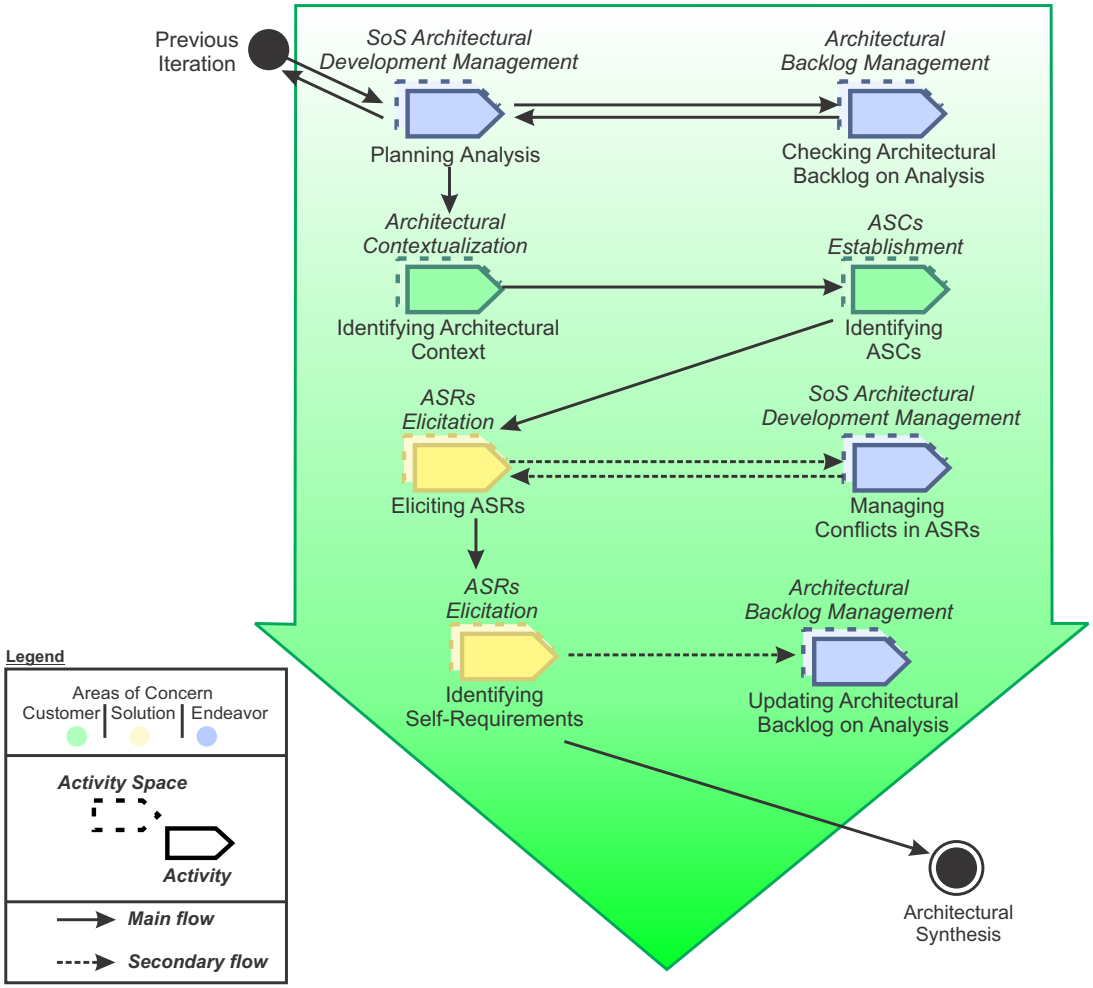


Figure 4.1: SOAR-A activities workflow

the possibility of some deviations during the analysis, this plan must be a reference to be followed as strictly as possible. As the main common issues to be considered in this activity, we point out:

- To identify and list representative Stakeholders that are relevant and agree to participate in the architecting process;
- To identify which stakeholders will contribute to each activity of architectural analysis;
- After the first iteration, reuse previous versions of evaluation plan as an information source to come up with a new plan version;
- To determine of which activities and tasks must be performed on each iteration; and
- To estimate and schedule time and resources required for analysis activities, e.g., time and human.

Checking Architectural Backlog on Analysis: This activity refers to the checking of architectural backlog, which can be used as information source to support architecting activities. As a cross-cutting element, this backlog is checked and updated in all architecting phases, i.e., analysis, synthesis, and evaluation. This backlog can also receive information from other processes in SoS development, e.g., the registry of new ideas for software architecture identified by other teams not responsible for architecture design. The architectural team must identify any backlog information related to architectural analysis, such as the changes on customers needs or potential new ASRs not agreed but registered in previous iterations for further consideration.

Identifying Architectural Context: When developing an SoS, there is a high amount of uncertainty surrounding such system, not only in a technical sense, but also in organizational and business senses. Given the understanding of SoS general context as entry-point, this activity aims at analyzing and identifying all information from this context relevant to software architecture. For example, laws and regulatory entities that may influence the SoS software architecture. Result of this activity must be the description of how the particularities of SoS context are related to its software architecture.

Identifying ASCs: In this activity the ASCs must be leveraged considering the SoS architectural context. It includes identify ASCs, the prioritization of these ASCs to software architecture, obtain agreement among different stakeholders which typically have different interests surrounding SoS. Therefore, this activity must also include tasks for promote a collaborative environment in which multiple stakeholders can interact with architectural team to negotiate ASCs.

Eliciting ASRs: This activity leverages a set of ASRs by considering inherent complexity of multiple interests and constituent systems collaborating with SoS. Two main directions of analysis are proposed, i.e., top-down and bottom-up. The top-down direction derive ASRs from general SoS requirements and their mapping to capabilities of constituents. So that, despite general SoS requirements be out of SOAR scope, it is convenient that these requirements be available when eliciting ASRs. Furthermore, the top-down SoS top-down analysis must also consider: (i) relationship with both global and individual missions; (ii) capabilities of the constituent systems needed to fulfill ASRs and contribute to the accomplishment of global missions; and (iii) predictable emergent behaviors top-down direction as a composition of functionalities available from constituent systems. In addition, eliciting ASRs in a top-down direction must take into account constraints established at SoS level that may affect the architecture and/or have influence over constituent systems.

The bottom-up direction is mainly concerned with understanding interactions among constituent systems that lead to emergent behaviors within the SoS, i.e., how such in-

teractions contribute to the fulfillment of ASRs and accomplishment of global missions. Despite the management of emergent behaviors is often performed in other activities of the architectural development (in particular, simulation and evaluation), ASRs elicitation must consider the assessment of both desirable and undesirable behaviors (either foreseen or unforeseen) that emerge from the interactions among constituent systems and that have influence on ASRs. The bottom-up elicitation must also take into account constraints from constituent systems their influence over the SoS software architecture.

Managing Conflicts in ASRs: ASRs may be conflicting for several reasons, such as: (i) existence of multiple stakeholders; (ii) conflicts in the relationship between constituent systems and SoS due to their managerial independence; (iii) conflicts arisen from the interactions among constituent systems due to their operational independence; and (iv) the fact that a given constituent system might simultaneously belong to more than one SoS. This activity is related to identification of such conflicts, understanding interdependencies among them, and establishing a trade-off to solve these conflicting requirements. Furthermore, in software-intensive SoS, the software typically interacts with other processes (e.g., physical and human). This activity must also understand and manage how these processes influence the SoS software architecture maintain the coherence through these different architectural layers. An example of strategy is the definition of common taxonomies and ontologies to facilitate common understanding and communication among these different processes.

Identifying Self-Requirements: SoS software architecture can itself be a source of ASRs (DoD, 2008). Therefore, this activity concerns the identification of ASRs coming from SoS software architecture, including:

- Analysis of potential constituent systems and their restrictions to identify if new ASRs come from them;
- Analysis of expected leveraged ASRs to identify if they generate new ASRs, e.g., new quality requirements yield from original ASRs; and
- Verification of the SoS software architecture already designed and validated in previous iterations, to identify new ASRs.

Updating Architectural Backlog on Analysis: This activity updates the architectural backlog according to performed architectural analysis, e.g., obsolete information from previous iterations can be removed and new ideas for potential changes in further iterations can be included.

4.1.2 SOAR-A Alpha States and Work Products

Work products are proposed to express *alphas* and provide evidences of progress in alpha states. In process instances, the inclusion of work products is guided by needs of each project. When necessary, work products can be adapted/reused from already existent practices or exclusively conceived for project teams. In this sense, SOAR-A recommends a set of work products for architectural analysis of acknowledged SoS. Based on SOAR Kernel, project teams can not only adopt and implement these work products but also conceive new ones. Following, regarding the alphas from SOAR Kernel employed in SOAR-A, we describe the alpha states variations expected to evidence the well execution of SOAR-A activities. Furthermore, for each of these alphas, we describe the work products recommended by SOAR-A:

Context: To perform SOAR-A, is expected that this alpha be at least[1] in the *provided* state before performing SOAR-A activities. In this state, general context is externally leveraged and documented by systems engineering processes and delivered as a source of information to software architecture process. After executing SOAR-A activities, the expected state to be reached is *documented*, in which a documentation is available and updated by considering all relevant elements of SoS software architecture (e.g., hierarchy, organizational documents, governmental rules, etc.) and the development teams and stakeholders also agree with this documentation. Furthermore, this documentation must to be maintained updated as SoS evolves. Work products proposed to express this alpha is the *Architectural context Documentation*. It can include elements such as:

- *Mission Model*, which comprises the description of both global missions (assigned to the SoS) and individual missions (assigned to constituent systems). Furthermore, this model must also include prioritization of these missions and any other relevant information (e.g., categorizations, associated metrics, and register of sources of external information related to SoS). SoS software architects can use this work product to define architectural models intended to meet mission needs related to capabilities provided by constituent systems (Silva et al., 2015); and

- *Domain Descriptions*, which must represent common concepts to be used through different stakeholders and developers and aims at enhancing communication in SoS development. Domain taxonomy, conceptual model, ontology, and glossary can be included in this work product.

Stakeholders: As an expected input for SOAR-A activities, this alpha must present at least the *represented* state, in which mechanisms for stakeholders participation are

[1]The term “at least” is employed because the alpha states are progressive.

agreed. After executiing SOAR-A activities, the expected state to be reached is *in agreement*, in which stakeholders agree with how their different priorities and perspectives are balanced in architecture to provide a clear direction for architectural team. As a work product for this alpha, we propose the *Stakeholders Map*, which must describe SoS stakeholders and any other information relevant to understand the multi-stakeholders environment of SoS. We recommend that this work product describe at least:

- Stakeholders roles and responsibilities in the architecting process;
- Relationships among stakeholders, including hierarchy and authority levels; and
- Mapping of groups of stakeholders with common interests, e.g., specific communities or organizations and the SoS ASCs related to them.

Architecturally Significant Concerns (ASCs): There is no specific previous state expected to this alpha before performing SOAR-A activities. After executing SOAR-A activities, the expected state to be reached is *established*, in which project teams and stakeholders accept the current set of ASCs as the most adequate to the SoS architectural context. This set of ASCs must be updated through SOAR-A activities during the evolutionary development iterations of SoS life cycle. The work product proposed in SOAR-A to express this alpha is the *List of ASCs*, which registers each concern and any related relevant information of these concerns, such as priority level, stakeholders that are related/interested on each concern, and analysis of impact of each concern in the SoS software architecture.

Architecturally Significant Requirements (ASRs): There is no specific previous state expected to this alpha before performing SOAR-A activities. After executing SOAR-A activities, the expected state to be reached is *elicited*, in which ASRs have been identified and agreed. The work products proposed in SOAR-A to express this alpha is the *ASR Documentation*, which must describes ASRs. As issues to be considered in this work product, we recommend:

- The description and essential information of ASRs agreed between architectural team and stakeholders. This information can include specific glossary, architectural impact, relation among ASRs and ASCs, description of operational scenarios; and
- A quality model, in which quality attributes can be organized, prioritized, and associated to metrics. Since software architecture plays a determinant role in the guarantee of quality for software architectures (Bass et al., 2012), a quality model can be produced to provide a specific view of ASRs that explicitly groups and represents quality attributes, e.g., by using quality attribute scenarios, which are

short descriptions of how a system should respond to some stimulus. These scenarios are extremely useful because they are architecture test cases useful on architectural evaluation activities (Clements et al., 2002). Furthermore, with the maturation of SoS community, general quality models can be proposed encompassing common quality attributes on SoS context, such as interoperability, security, and scalability (Bianchi et al., 2015).

Architectural Team: As an expected input for SOAR-A activities, this alpha presents at least the *formed* state, in which the team has been populated with enough committed people to start the work. After executing SOAR-A activities, the state expected to this alphas before SOAR-A is *performing*, in which the team is working effectively and efficiently. The work product of SOAR-A to express this alpha is the Architectural Team *Work Scheme*, in which all relevant information about the architectural team must be documented, such as coaching plan, the list of individuals and respective skills, roles, hierarchy, etc. This work product must be maintained and updated through all the architecting process, being also present in other phases, i.e., synthesis and evaluation.

SoS Development Environment: There is no specific previous state expected to this alpha before performing SOAR-A activities. After performing SOAR-A activities, the expected state to be reached is *working*, in which the development environment is supported to adequately develop the SoS software architecture. Furthermore, SoS are complex systems that involve a joint work of different companies, demanding for a more precise documentation (Sommerville, 2009). The work product in SOAR-A to express this alpha is the *Analysis Plan*, which documents the required elements to ensure an environment as adequate as possible to elicit ASRs in a distributed perspective of development.

Architectural Backlog: Execution of SOAR-A activities must ensure the maintenance of the *updated* state for this alpha, in which it is continuously reviewed and feed with relevant information regarding the architectural development of the SoS. The work product in SOAR-A to express this alpha is the *Analysis Backlog*, which includes backlog items leveraged on architectural analysis activities. It must include any relevant information of this matter, such as the feedback of problems found during the architectural analysis to enhance elicitation in further iterations or even registration of not agreed ASRs and ASCs to be further considered.

Acknowledged SoS: There is no specific previous state expected to this alpha before performing SOAR-A activities. After executing SOAR-A activities, the expected state to be reached is *analyzed*, in which architectural analysis was performed and ASRs were established expressing problems that software architecture must solve. There is no work product in SOAR-S to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

Constituent Systems: There is no specific previous state expected to this alpha before performing SOAR-A activities. After executing SOAR-A activities, the expected state to be reached is *contextualized*, in which potential constituent systems and their providers/authorities were identified and, if necessary, they are in agreement with ASRs. There is no work product in SOAR-A to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

In order to illustrate how the aforementioned work products are related to activities in SOAR-A, Table 4.1 presents for each activity its related work products and kind of use, i.e., input/output.

Table 4.1: Work products produced/updated in SOAR-A activities

Work Product / Activity	***Architectural Context***	***Analysis Plan***	***Architectural Backlog***	***Stakeholders Map***	***List of ASCs***	***ASRs Documentation***	***Work Scheme***
Planning Analysis	Input	Input/ Output	-	Input	-	-	Input/ Output
Checking Architectural Backlog on Analysis	-	Input	Input	-	-	-	-
Identifying Architectural Context	Input/ Output	Input	-	Output	-	-	-
Identifying ASCs	Input	Input	-	Input	Output	-	-
Eliciting ASRs	Input	Input	-	Input	Input	Input/ Output	-
Managing Conflicts in ASRs	-	Input	-	Input	Input	Input/ Output	-
Checking Self-Requirements	-	Input	-	-	Input	Input/ Output	-
Updating Architectural Backlog on Synthesis	-	Input	Output	-	-	-	-

4.2 Evaluation

Aiming at assessing if SOAR-A can be suitable to support architectural analysis in acknowledged SoS, we conducted a qualitative survey with experts from both academia and industry who have been involved in the development of SoS. As the previous survey conducted for SOAR Kernel (see Section 3.2), this study was also based on the survey steps

proposed by Kasunic (2005) that comprise: (i) identify research objectives; (ii) identify & characterize target audience; (iii) design sampling plan; (iv) design & write questionnaire; (v) pilot test questionnaire; (vi) distribute questionnaire; and (vii) analyze results and write report. These steps are described as follows:

1. **Identify the research objectives:** This survey aimed to verify if SOAR-A meets expectations of SoS community as an approach to support the architectural analysis of SoS. Four research questions (RQs), summarized in Table 4.2, guided this survey.

Table 4.2: Survey research questions

SOAR-A Practice	
RQ1	Is SOAR-A complete for what it is proposed?
RQ2	Is SOAR-A correct, with no wrong or misunderstood statements?
RQ3	Is SOAR-A conceptually coherent, with no conflicts or wrong placed elements?
RQ4	Can SOAR-A be considered intelligible, well-organized, concise, helpful, and easy to use?

2. **Identify and characterize target audience:** The survey target audience is represented by potential SOAR users, i.e., members of SoS community on both academy and industry. To obtain a sample of this population as representative as possible, SoS researchers who have been conducted studies on SoS and developers who have been constructed SoS were considered. The level of expertise were also collected to support further analysis of answers.

3. **Design sampling plan:** Due to the same reasons presented in the survey conducted for SOAR Kernel (see Section 3.2), sample size was limited by the number of individuals that agreed to participate, i.e., a set of five experts.

4. **Design and write questionnaire:** The survey instrument for gathering data was an on-line self-administered questionnaire[2], i.e., a questionnaire designed specifically to be completed by each participant without intervention of the researchers. This questionnaire was produced in the light of the research questions outlined in Table 4.2 and included non-discursive (closed) and discursive (open) questions. The questionnaire focused on evaluating the activities, workflow, and work products proposed in SOAR-A. In closed questions, participants were asked to provide a score to evaluate elements of SOAR-A with respect to the aforementioned dimensions. In open questions, participants provided textual answers justifying their scores as

[2]The questionnaires of all surveys of this Thesis are included in Appendix E.

well as additional comments and improvement suggestions. Additionally, the participants were provided with the following documentation: (i) guidelines to read documentation and answer the questionnaire; (ii) a profile questionnaire to verify the level of expertise of the participants; (iii) the complete description of the SOAR Kernel and SOAR-A; and (iv) a support documentation about the Essence Language, and Essence Kernel.

5. **Pilot test questionnaire:** An initial survey pilot was executed with the participation of one researcher from our research group. Data were collected only for verifying errors, average time for response, and possible enhancements in the first version of survey material.

6. **Distribute questionnaire:** After the pilot, an invitation was sent to participants and the survey was sent to whom has agreed to participate. Each participant followed three steps when answering the questionnaire: (i) answering questions about level of expertise,(ii) reading survey's documentation, and (iii) answering questions about SOAR-A.

7. **Analyze results and write report:** For non-discursive scale-based questions, graphics were associated to a textual discussion to illustrate observed trends in the experts' opinions. Due to the low number of survey participants, no statistical test were applied. Furthermore, discursive questions provided insights and open suggestions to enhance SOAR-A. The analysis strategy for these questions was the presentation of results through narrative compilations of answers and additional discussions of our observations. Section 4.2.1 presents obtained results and discuss how they help to enhance SOAR-A.

4.2.1 Analysis and Interpretation of Results

The first analysis identified the profile of participants with three questions. Two non-discursive questions, presented in Figure 4.2, asked about expertise in both SoS and software architectures. The level of expertise range from zero to three. It indicates that all participants had some level of expertise/knowledge concerning SoS and software architectures. Additionally, a third discursive question asked for the occupation of participants. In this case, the predominance was academic researchers, i.e., only one participant assert to work as industry practitioner. After this profile analysis, we detailed results for each RQ as follows:

RQ1. We conceived three questions for this RQ, two of them non-discursive and a discursive one. Figure 4.3 presents results of non-discursive questions. The first

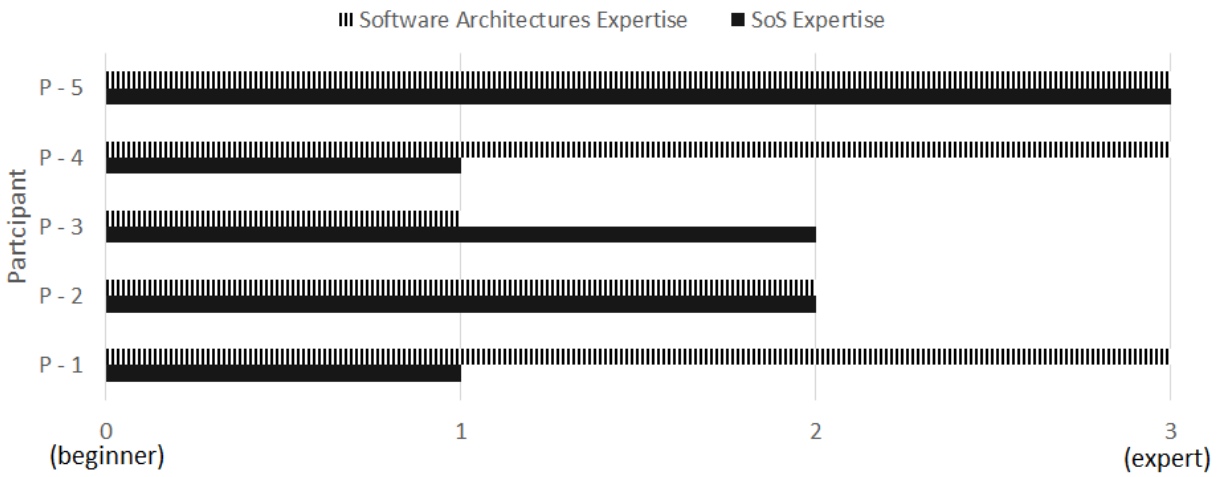


Figure 4.2: Levels of expertise on SOAR-A survey

non-discursive question asked participants for grading work products in terms of completeness, indicating how complete SOAR-A work products are to encompass architectural analysis of acknowledged SoS. The second non-discursive question was about the completeness of SOAR-A activities. Results of both non-discursive questions indicate that participants considered SOAR-A activities and work products as satisfactory in terms of completeness. Finally, in the discursive question, participants were asked to point out any element that could be missing in SOAR-A. In their responses, participants indicated as missing issues: (i) need of a more complete description of some work products (i.e., *Context* and *ASRs*); (ii) a path lacking in the main workflow (i.e., the path from *Checking Architectural Backlog on Analysis* to *Planning Analysis* activity); and (iii) the need of a more complete description of some activities (i.e., the activities *Identifying Self-requirements* and *Planning Analysis*). After analyze these suggestions, we implemented some enhancements to SOAR-A: (i) inclusion of "Misson Model" and "Domain Ontology/Taxonomy" as suggested content to be part of *Context* work product; (ii) the inclusion of lacking path; (iii) enhancements in *Planning Analysis* activity by including a set of common issues to be considered in this activity; and (iv) enhancements in *Identifying Self-requirements* activity by including a set of common sources of self-requirements to be considered in this activity.

RQ2. This RQ was encompassed by two questions, a discursive and a non-discursive one. Figure 4.4 shows the positive results of non-discursive question indicating the general acceptance of SOAR-A as correct. The discursive question asked for indication of possible errors. One participant pointed out some errors: (i) all activities might determine what alpha states are expected as inputs. The "Planning Analysis" activity had no states; (ii) an unnecessary direct flow from *Checking Architectural*

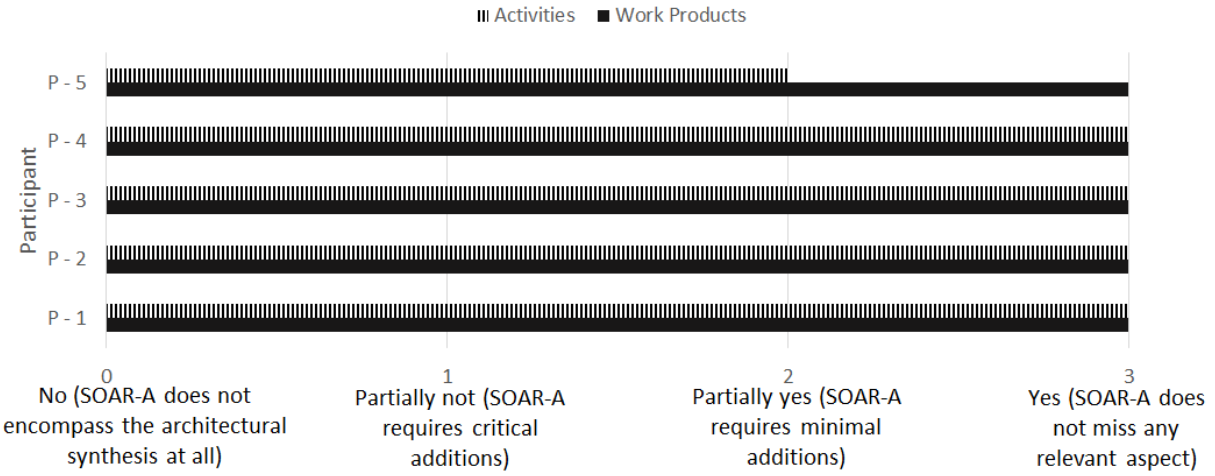


Figure 4.3: SOAR-A Survey: RQ1 Results of Non-discursive Questions

Backlog on Analysis to *Managing Conflicts in ASRs*. These errors were corrected and all alpha states were also revised.

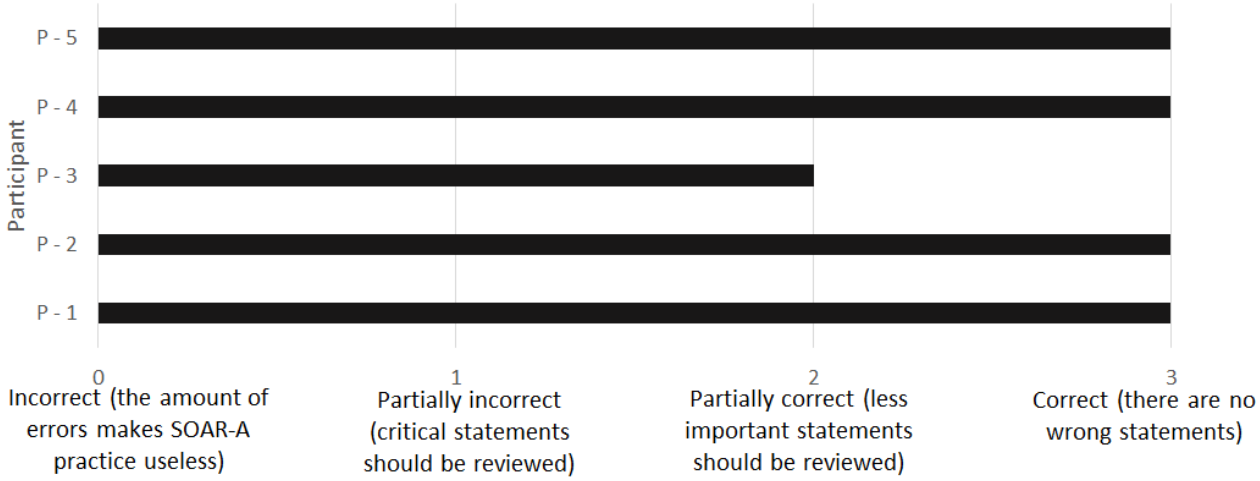


Figure 4.4: SOAR-A survey: RQ2 results of non-discursive questions

RQ3. This RQ was structured into two questions, a discursive and a non-discursive one. The non-discursive question asked the participants to grade SOAR-A in terms of coherence. Figure 4.5 shows results of the non-discursive question indicating the general acceptance of SOAR-A as coherent. The discursive question asked participants for justifying their grades in the previous question and indicating problems regarding coherence. In this question, participants did not point out any suggestion to enhance SOAR-A coherence.

RQ4. This RQ was structured into two questions, a discursive and a non-discursive one. Figure 4.6 shows results of the non-discursive question indicating general acceptance of SOAR-A in terms of clearance and organization. Answers collected in discursive

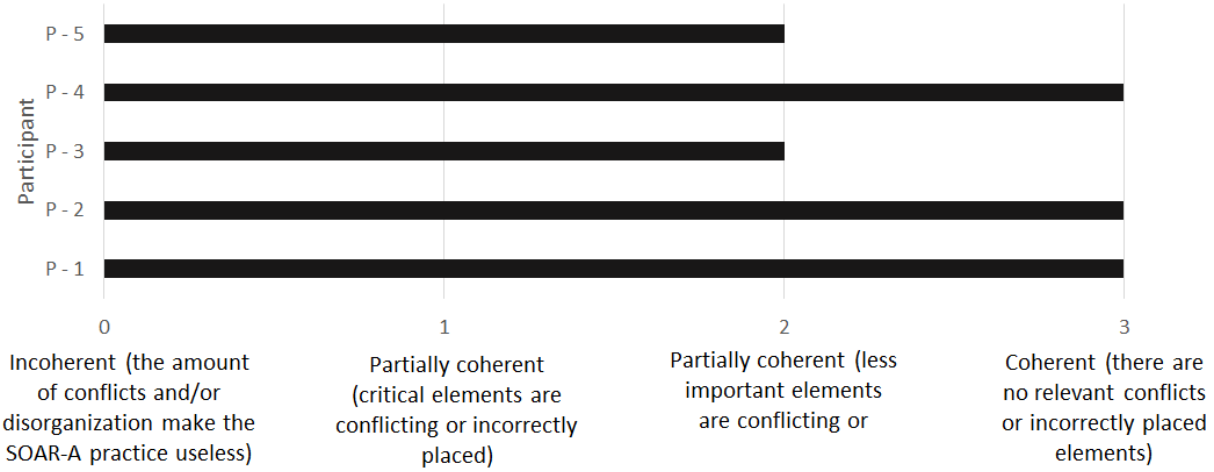


Figure 4.5: SOAR-A Survey: RQ3 results of non-discursive questions

question also provided us with relevant insights for improving: (i) graphical representation of SOAR-A workflow; and (ii) textual clearance of *Managing Conflicts in ASRs* activity description. In order to meet these suggestions, we implemented some changes in SOAR-A: (i) enhancements in the figure of SOAR-A workflow, adding a legend not originally existent as an Essence graphical elements. Additionally, we included legends for all workflows of SOAR; and (ii) examples of common sources of conflicts when establishing ASRs were included in *Managing Conflicts in ASRs* activity.

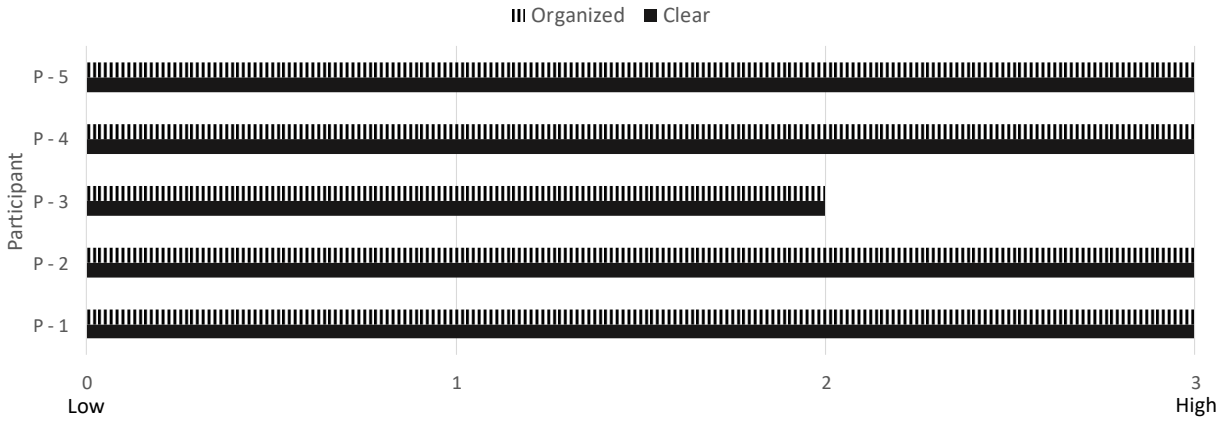


Figure 4.6: SOAR-A Survey: RQ4 results of non-discursive questions

In general, answers were positive with good levels of satisfaction in all RQs. These results represent a good indicative that SOAR-A can support processes of architectural analysis of acknowledged SoS software architectures.

4.2.2 Threats to Valididy

The conducted survey and its results may have been affected by some threats to empirical validity. These threats are discussed as follows.

Internal Validity: To increase validity of the study regarding this concern, we carefully designed, piloted, and iteratively refined the questionnaire to: (i) mitigate risk of ambiguous and poorly phrased questions; (ii) ensure that all participants could properly understand the proposed questions; and (iii) participation was anonymous.

External Validity: As previously mentioned, the sample was limited to five participants and defined by convenience, hampering broader generalizations to target population. Nonetheless, the number of participants of this study still can be accepted since the main goal was to gain insights about SOAR-A and suggestions for improving it.

Reliability Validity: Aiming at mitigate this threat, the conducted survey was designed by following a well defined and accepted methodology (Kasunic, 2005; Shull et al., 2008). Moreover, survey protocol and applied questionnaire were made publicly available.

Construct Validity: Most of bias coming from researchers were mitigated conceiving part of the questions had pre-defined answers (closed questions). Despite discursive (open) questions could yield different interpretations, they helped us to implement important enhancements so SOAR-A described in previous section.

4.3 Final Remarks

This chapter presented SOAR-A, a SOAR practice to support the establishment of process instances for architectural analysis of acknowledged SoS. SOAR-A resulted from an analysis of the state of the art on SoS and, similarly to SOAR Kernel, it applies Essence Standard. The main contributions of this chapter are: (i) SOAR-A and its documentation built in Essence Workbench Tool; and (ii) a survey evaluation with experts on SoS software architectures. Results of this survey shown a good acceptance of SOAR-A. Experts pointed out SOAR-A as adequate, comprehensive to support instantiation of process for architectural analysis of acknowledged SoS. The next chapter presents SOAR-S, a practice of SOAR that supports the establishment of process instances for architectural synthesis of acknowledged SoS.

Chapter

5

SOAR-S: A Practice for Architectural Synthesis on Acknowledged SoS

This chapter introduces SOAR-S, a SOAR practice that supports the establishment of process instances for architectural synthesis of acknowledged SoS. Section 5.1 describes SOAR-S. This description refers to the final version of SOAR-S, produced after two studies conducted to evaluate SOAR-S. Section 5.2 presents an observational study conducted to assess SOAR-S in terms of feasibility. Section 5.3 describes an experiment conducted to evaluate SOAR-S. Final remarks are presented in Section 5.4.

5.1 Description of SOAR-S

Within the construction of software architectures, architectural synthesis proposes architectural solutions to meet the ASRs upon the system (Hofmeister et al., 2007). Despite the relevance of architectural synthesis as a driver for system implementation, existing approaches in the literature do not properly address architectural synthesis in SoS software architectures. Architectural synthesis encompasses "making" of architectural decisions. As previously discussed (see Chapter 2), despite synthesis be present in several architectural design methods, particularities of SoS requires specialized solutions conceived to support its particularities, e.g., evolutionary development and existence of independent architectures from constituent systems.

Due to the independence of constituent systems, SoS software architectures are inherently dynamic. Moreover, emergent behaviors result from collaborative work of constituent systems and these systems can be not subordinated to SoS interests. Therefore, 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. In this context, SOAR-S was conceived to provide guidelines on "how to" perform architectural synthesis for acknowledged SoS.

As a part of SOAR, SOAR-S is also described upon the OMG's Essence Standard and documented in Essence Workbench Tool. Following, activities, alpha states, and work products of SOAR-S are described.

5.1.1 SOAR-S Activities

SOAR-S includes an essential set of activities to be followed when proposing architectural solutions, i.e., CASs, to a set of ASRs in acknowledged SoS. Figure 5.1 shows activities of SOAR-S, their workflow, and correspondent activity spaces inherited from the SOAR Kernel. These activities are described as follows:

Planning Synthesis: This activity establishes/updates a plan for architectural synthesis, dealing with issues concerning execution of next activities of architectural analysis. The main common issues to be considered are:

- Establishment of what techniques, tools, and technologies must be used to support architectural synthesis, e.g., modeling and simulation tools or languages for representation.
- Estimation and scheduling of required resources, e.g., time and human. This scheduling is a reference for the next activities, e.g., when selecting the ASRs to be encompassed in current iteration; and
- After the first iteration, the review of previous iterations in order to maintain architectural synthesis as expected.

Checking Architectural Backlog on Synthesis: As in architectural analysis, the architectural backlog must be also checked on synthesis. It is particularly relevant to verify if there are registered design ideas from external processes or previous architecting iterations registered to be considered in further iterations.

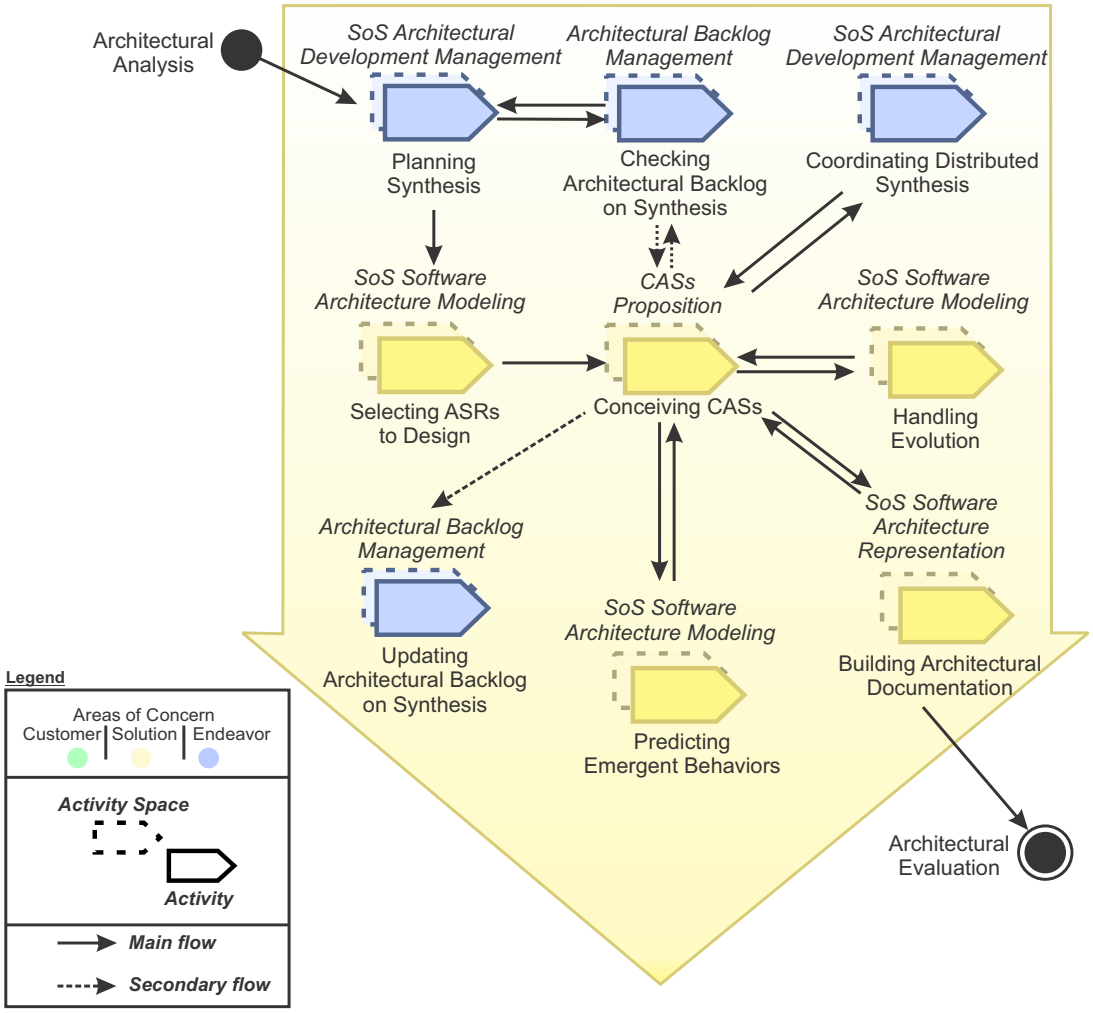


Figure 5.1: SOAR-S activities workflow

Selecting ASRs to Design: This activity is about the decision and establishment of what must be designed in each synthesis iteration based on a set of ASRs to be handled. It defines an architectural milestone in which several factors can influence the number of ASRs to be handled, such as diversity of application domains, architects expertise in these domains, and teams size (Bass et al., 2012).

Conceiving CASs: SoSs are complex and interdisciplinary systems that inherently encompass several types of architectures, or architectural perspectives (e.g., physical and human) (DeLaurentis, 2008). In this activity, a set of CASs is proposed to encompass ASRs under design, thus meeting a set of ASRs and establishing a new architectural version to be further evaluated. Several sources of information must be considered, such as ASCs, context, and the architectural backlog. In this activity, some issues can be considered:

- Different knowledge sources for conceiving CASs, such as background of architectural team on designing similar architectures, frameworks, design checklists, domain decomposition, reference architectures, and architectural patterns (Bass et al., 2012).
- Understanding of how other SoS architectural perspectives influence software architecture to maintain coherence through these different perspectives; and
- Analysis of impact of CASs in terms of risks and implementation costs.

Coordinating Distributed Synthesis: SoS development typically involves different organizations performing a collaborative, distributed development of constituent systems and SoS. In this sense, this activity must support such collaborative work through heterogeneous teams and stakeholders ensuring through negotiation of authority levels and communication strategies the well-execution of architectural synthesis. Although architectures of constituent systems cannot be directly manipulated by the SoS architectural team, architectural decisions at SoS level may demand for agreement at constituents level, such as specific constraints, patterns adoption, architectural frameworks, and reference architectures. These details, which involves decisions at constituent systems level, must be negotiated/agreed between architectural teams of SoS and constituent systems.

Handling Evolution: SoS are inherently evolutionary and their architectures must be developed considering this perspective. Evolvability is the ability to easily accommodate future changes. This attribute is highly required in SoS, since the architecture is constantly evolving. This activity investigates and establishes CASs to maintain SoS evolvability. Since CASs must be established by considering the evolvability concern, this activity dialogues with the *Conceiving CASs* activity influencing in the proposed architectural version.

Predicting Emergent Behaviors: This activity is about predict emergent behaviors to support CASs establishment. Since emergent behaviors of an SoS are not simply a sum of parts (e.g. , constituent systems and their capabilities), each emergent behavior can assume different classifications (Holland, 2007): (i) foreseen, when developers could identify its possibility of occurrence; (ii) unforeseen, when developers could not identify its existence; (iii) desirable, that is expected to occur as a result of SoS operation; and (iv) undesirable, when it should not occur if SoS is operating as planned. Regarding this classification, foreseen and unforeseen emergent behaviors can be desirable or even undesirable. In this activity, efforts must be directed to minimize unforeseen behaviors, e.g., by analyzing architectural models based on the new CASs or using architectural simulation to understand the SoS operation at runtime. Therefore, relevant information can be provided to Conceiving CASs activity about emergent behaviors that can dynamically occur in SoS.

Updating Architectural Backlog on Synthesis: This activity refers to registry and maintain in the architectural backlog. Any additional information regarding the performed synthesis and not registered in other work products can be part of architectural backlog, e.g., ideas to be considered in further iterations.

Building Architectural Documentation: In this activity, SoS software architecture is represented by considering different interests, viewpoints, and particular environments. This documentation must describe a new architectural version that reflects the inclusion of conceived CASs. After architectural evaluation, CASs can be added to a definitive documentation of the SoS software architecture. This architectural description can be made with different representation techniques (informal, semi-formal, and formal) and covering different views (e.g., structural and behavioral) to provide a better understanding of the software architecture to both stakeholders and developers. Some architectural frameworks have been used to reduce efforts of modeling SoS, e.g., Defense Architecture Framework (DoDAF) (DoD, 2010). ADLs can be also used to this representation, such as UML (Object Management Group (OMG), 2015b), SysML (SysML Partners, 2015), and more recently SoSADL (Oquendo, 2016).

5.1.2 SOAR-S Alpha States and Work Products

Work products of SOAR-S express some alphas inherited from SOAR Kernel. Following we present the states variation of these alphas when performing SOAR-S and describe the work products of it:

Architectural Backlog: Execution of SOAR-S activities must ensure maintenance of *updated* state for this alpha, in which it is continuously reviewed and feed with relevant information regarding the architectural design. The work product proposed in SOAR-S to express this alpha is *Synthesis Backlog*, which includes any information relevant to further iterations that were not foreseen in other work products, such as feedback of problems found during the synthesis activities (e.g., inconsistencies in ASRs), considerations for other phases of architecting process, registration of potential changes in ASRs resulting from synthesis conduction, and not agreed design ideas registered for further consideration.

Architecturally Significant Concerns (ASCs): As an expected input for SOAR-S activities, this alpha must be in *established* state, in which ASCs were identified and agreed with stakeholders. ASCs are used as an information source for SOAR-S activities and its state does not changed after architectural synthesis. Because the establishment and documentation of ASCs must be performed during architectural analysis, there is no work product proposed in SOAR-S to express this alpha.

Architecturally Significant Requirements (ASRs): For this alpha, at least the *established* state, in which ASRs were established. ASRs are an input for SOAR-S activ-

ities, previously established during architectural analysis, and does not changed after its execution. If problems in ASRs are identified during SOAR-S, this information must be registered in the architectural backlog.

Candidate Architectural Solutions (CASs): There is no specific previous state expected to this alpha before performing SOAR-S activities. After executing SOAR-S activities, the expected state to be reached is *proposed*, in which a new set of CASs was proposed to meet one or more ASRs. The work product proposed in SOAR-S to express this alpha is the *CASs Documentation*, which includes the CASs representation as a new architectural version and other relevant information, such as priority rank for implementation, lists of alternative CASs, and traceability among CASs, ASRs, stakeholders, and emergent behaviors. Furthermore, this is a support documentation and the representation of CASs into the architecture, considering different views and representation strategies, must be done in *SoS Software Architecture Documentation*, a work product for software architecture alpha.

Software Architecture: There is no specific previous state expected to this alpha before performing SOAR-S activities. After executing SOAR-S activities, the expected state to be reached is *represented*, in which there is an agreement for CASs and they were adequately represented. The work product proposed in SOAR-S to express this alpha is *SoS Software Architecture Documentation*. This documentation must explicit describe SoS software architecture.

Acknowledged SoS: As an expected input for SOAR-S activities, this alpha presents at least the *analyzed* state, in which ASRs were previously established during architectural analysis. After executing SOAR-S activities, the expected state to be reached is *designing*, in which the SoS software architecture has a new version to be evaluated. There is no work product proposed in SOAR-S to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

Constituent Systems: After executing SOAR-S activities, the expected state to be reached is *negotiated*, in which negotiation with the constituent systems providers/authorities were performed and they are in agreement with ASRs, CASs, and what they must offer in terms of capabilities to effectively operate in SoS context. Furthermore, constituent systems boundaries and individual missions must be also know at SoS level. There is no work product proposed in SOAR-S to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

Emergent Behaviors: There is no specific previous state expected to this alpha before performing SOAR-S activities. After executing SOAR-S activities, the expected state to be reached is *predicted*, in which analysis and prediction strategies were followed to leverage both desired and undesired behaviors. There is no work product proposed

in SOAR-S to directly express this alpha; however, project teams can schedule and add additional documentation if necessary.

Architectural Team: Execution of SOAR-S activities must ensure the maintenance of the *performing* state for this alpha, in which architectural team is working effectively and efficiently. The work product proposed in SOAR-S to express this alpha is the *Work Scheme*, a work product present in all design phases, i.e., analysis, synthesis, and evaluation.

Development Environment: Execution of SOAR-S activities must ensure the maintenance of the *working* state, in which the distributed environment adequately works to develop the SoS software architecture. The work product proposed in SOAR-S to express this alpha is the *Synthesis Plan*, which documents required elements to ensure an environment as adequate as possible to perform architectural synthesis activities.

Stakeholders: Execution of SOAR-S activities must ensure the maintenance of the *in agreement* state for this alpha, in which stakeholders agree with how their different priorities and perspectives are balanced in the architecture to provide a clear direction for the architectural team. There is no work product proposed in SOAR-S to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

Furthermore, in order to illustrate how the aforementioned work products are related to activities in SOAR-A, Table 4.1 presents for each activity its related work products and respective kind of use (input/output).

In order to illustrate how aforementioned work products can relate to activities of SOAR-S, Table 5.1 presents for each activity its related work products and kinds of use, i.e., input/output.

5.2 Verifying the Applicability of SOAR-S: First Study

Since not only SOAR-S but also Essence Standard are new approaches in software engineering, it is essential to verify if is it possible to generate process instances from SOAR-S. In this context, the main goal of this study was to produce a proof of concept about the feasibility of applying SOAR-S to generate process instances adequate to architectural synthesis of acknowledged SoS software architectures. Results provided more confidence to further conduction of the experiment described in Section 5.3.

This study was conducted with six graduate students from the University of São Paulo (USP) during the Fall 2015 semester. Our study has focused on using SOAR-S to conceive process instances. Additionally, feedback on the participants experience in using SOAR-S was collected and used to enhance SOAR-S. This study was carried out based on

Table 5.1: Work products produced/updated in SOAR-S activities

Work Product / Activity	***Synthesis Plan***	***Arch. Backlog***	***CASs Documentation***	***SoS Software Architecture Documentation***	***Work Scheme***
Planning Synthesis	Output	-	-	-	-
Checking Architectural Backlog on Synthesis	Input	Input	-	-	-
Selecting ASRs to Design	Input	-	Input	Input	-
Conceiving Candidate Architectural Solutions (CASs)	Input	-	Input/ Output	-	-
Coordinating Distributed Synthesis	Input	-	-	-	Input/ Output
Handling Evolution	Input	-	Input/ Output	-	-
Predicting Emergent Behaviors	Input	-	-	-	-
Updating Architectural Backlog on Synthesis	Input	Output	Input	-	-
Building Architectural Documentation	Input	-	Input	Output	-

the process for software engineering experiments proposed by Wohlin et al. (2012) that comprises five main steps: scoping, planning, operation, analysis and interpretation, and presentation.

5.2.1 Scope and Planning of the Study

The objective of this study was outlined by using the Goal, Question, Metric (GQM) technique (Basili et al., 1999). According to GQM, the study had the *objective* of analyzing SOAR-S *for the purpose* of evaluation *with respect to* applicability and generality in producing process instances *from the point of view of* software engineering students *in the context* of a course of Experimental Software Engineering of the Graduation Program at USP. Based on this goal, we established two research questions (RQ):

- RQ1: Can SOAR-S support the production of process instances of architectural synthesis for acknowledged SoS software architectures?

- RQ2: Can SOAR-S support the production of process instances for different application domains?

The study was performed in the context of an Experimental Software Engineering course of graduation program at USP. Students (hereafter referred to as subjects) were chosen by convenience, as they had knowledge on software engineering and represented a sample from possible SoS developers. Our planned strategy to answer the RQs was ask these subjects to built process instances for two different acknowledged SoS by using SOAR-S support. We first built descriptions of two different acknowledged SoS and their development scenarios to be used in instantiation activities: a flood monitoring SoS and a Global Earth Observation SoS. By considering these descriptions, we also produced two process instances for architectural synthesis with SOAR-S support (hereafter referred to as reference instances). These reference instances express what we expect from using SOAR-S support and were used as models for evaluating the instances generated by the subjects. In this perspective, it was considered the level of conformance of subjects' instances with these reference instances. Furthermore, two hypotheses were defined in our study, one for each research question. These hypotheses are described as follows:

1. **Applicability:** Null hypothesis, H_{A0}: It is not possible to generate process instances to design acknowledged SoS with the support of SOAR-S.

   Alternative hypothesis, H_{A1}: It is possible to generate process instances to design acknowledged SoS with the support of SOAR-S.

2. **Generality of Domain:** Null hypothesis, H_{G0}: It is not possible to apply SOAR-S for generating process instances to design acknowledged SoS when considering different application domains.

   Alternative hypothesis, H_{G1}: It is possible to apply SOAR-S for generating process instances to design acknowledged SoS when considering different application domains.

To test our hypothesis, we proposed two metrics (i.e., independent variables) ranging from zero to one:

- **Conformance Factor (CF):** When building a process instance using SOAR-S, subjects analyze the SoS description and select which SOAR-S elements, i.e., activities and work products, they believe should be used. We thus evaluated which of

these elements were correctly selected by comparing subjects selections against selections of our reference instances. Based on this evaluation, this metric measures the conformance of subjects' instances raised to each SOAR-S element. CF is computed (in its non-normalized form), for each SOAR-S activity or work product, as the percentage of process instances produced by subjects in which the selection is equal to the selection from reference instances. In this case, the higher the percentage, more equal process instances produced by subjects are to reference instances.

- **Variation Factor (VF):** In this study, subjects produced process instances for two different application domains. For each of these domains, we calculated an average CF based on CFs of all activities and work products of SOAR-S reached in each domain. Given these average CFs, this metric measures the variation of them. VF is computed as the module of the difference between these two average CFs. In this case, the lower the VF, the higher the independence of process instances of specific application domains (i.e., the CFs raised in different application domains are similar).

Materials used during this study were: (i) initial version of SOAR-S described in Essence Language; (ii) descriptions of two SoS in different application domains (i.e., GEO and flood monitoring, named GEOSS and FMSoS); and (iii) a form to be filled by subjects with their process instances. We structured this form into three parts:

- **Part I:** personal level of knowledge in main issues of the study, i.e., software architecture and software engineering processes;
- **Part II:** guidelines for creating process instances with SOAR-S support. For this, we provided a textual field to be filled with the instance description in terms of activities and work products. Therefore, it was mandatory to provide at least these elements in process instances;
- **Part III:** fields to be filled by each subject with elements of SOAR-S must be included in its process instance; and
- **Part IV:** personal impressions from subjects regarding utilization of SOAR-S.

5.2.2 Study Operation

Before conduction with real subjects, we performed a pilot with two participants from our research group to verify the study conformance with established planning. After this pilot, we established the steps presented in Figure 5.2 that were followed in the study operation. Subjects first received a training on SOAR-S and how to explore its

representation in *EssWork Practice Workbench*. Next, were divided into two groups: Group I, which was introduced to GEO SoS; and Group II, which was introduced to Flood Monitoring SoS. Both groups received descriptions of respective SoS of training and the SOAR-S represented in *EssWork Practice Workbench*. After training sessions, two groups were asked to produce process instances based on received documentation. There was no time limit to produce the process instances and the subjects spent an average time of one hour to produce the instances.

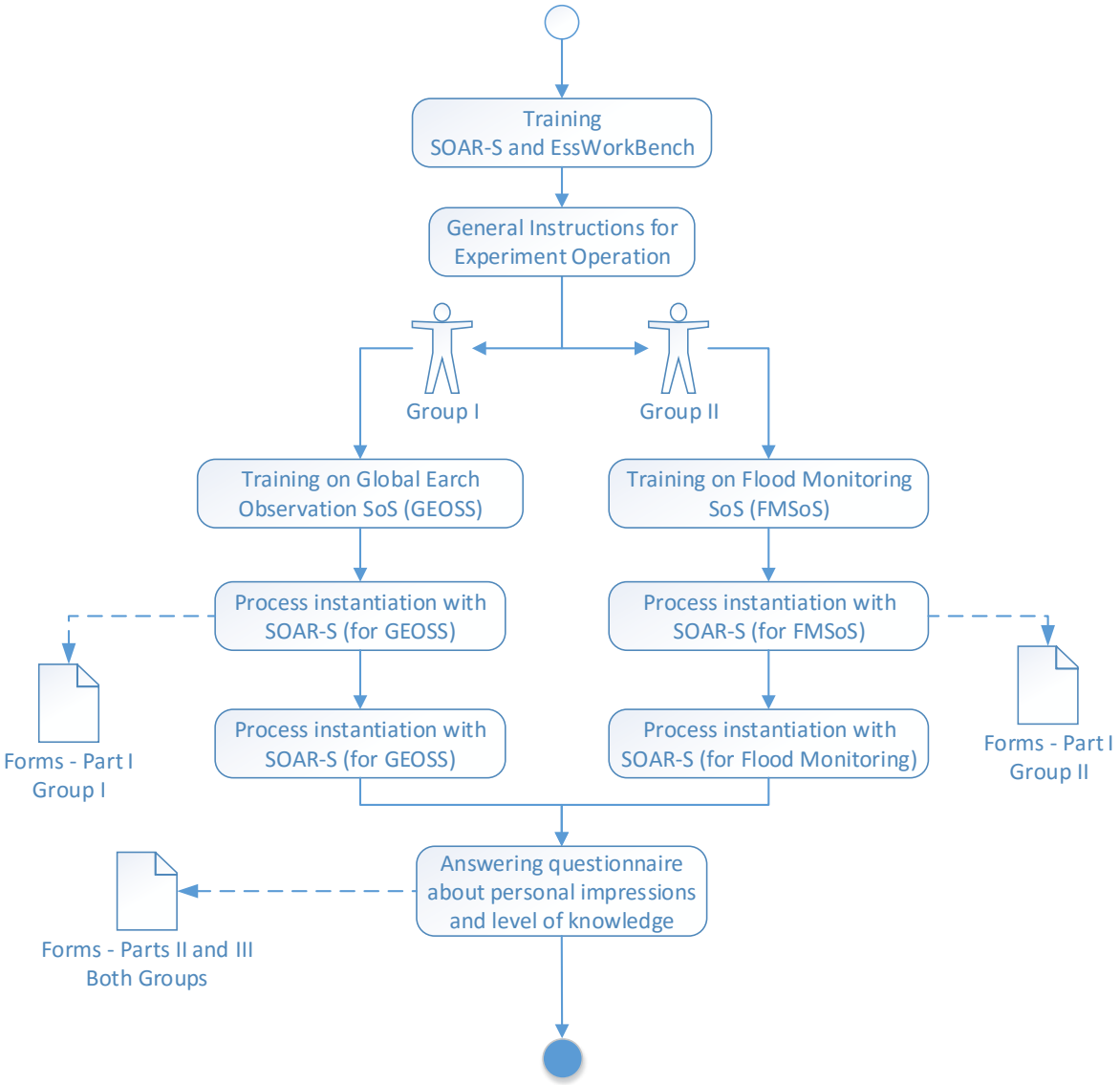


Figure 5.2: Study operation

5.2.3 Analysis and Interpretation of Results

A summary of obtained results is shown in Table 5.2. For each element of SOAR-S (i.e., activities and work products), we present average results achieved by each group in relation to pre-conceived reference instances. We observed in both groups that instances generated by subjects achieved a high degree of conformance (i.e., excepting the *Coordinating Distributed Synthesis* in Group II activity, all other evaluated elements reached at least 58.3% of average CF). Furthermore, average CFs reached by the two groups was similar, reaching a VF of 8.97%. In this context, we observed that was possible to use SOAR-S to generate instances of process for architectural synthesis of acknowledged SoS.

Table 5.2: First SOAR-S study: summary of results

Activity	**Group I - GEOSS (%)**	**Group II - FMSoS (%)**
Planning Synthesis	100.0	100.0
Checking Architectural Backlog	100.0	66.7
Selecting ASRs to design	66.7	100.0
Conceiving CASs	100.0	66.7
Coordinating Distributed Synthesis	66.7	33.3
Predicting Emergent Behaviors	100.0	100.0
Building Architectural Documentation	100.0	66.7
Updating Architectural Backlog	100.0	100.0
Alpha	**Group I - GEOSS (%)**	**Group II - FMSoS (%)**
Synthesis Plan	66.7	83.3
Architectural Backlog	83.3	58.3
CASs Documentation	83.3	58.3
Work Scheme	66.7	100.0
Software Architecture Documentation	83.3	66.7
Average CF	**85.9**	**76.9**
VF=9		

Subjects were also asked for evaluate SOAR-S representation and SoS descriptions by providing personal impressions, additional comments, and enhancement suggestions. This information was extracted in Part II of form, which was built with non-discursive questions and open discursive ones. Non-discursive questions were proposed ranging from 1 (very bad) to 4 (excellent) and evaluation followed three main perspectives: coherence, cleanness, and organization. Figure 5.3 summarizes the results about impressions on SOAR-S representation. Not excellent averages of coherence and completeness provided indicatives to review SOAR-S description to enhance the representation in these aspects. Figure 5.4 summarizes results about SoS descriptions, in which despite similarity in the

results between the two application domains, low averages in coherence and cleanness, help on review documentation of SoS descriptions not only to further replications of this study, but also for reuse of this documentation in the experiment presented in Section 5.3.

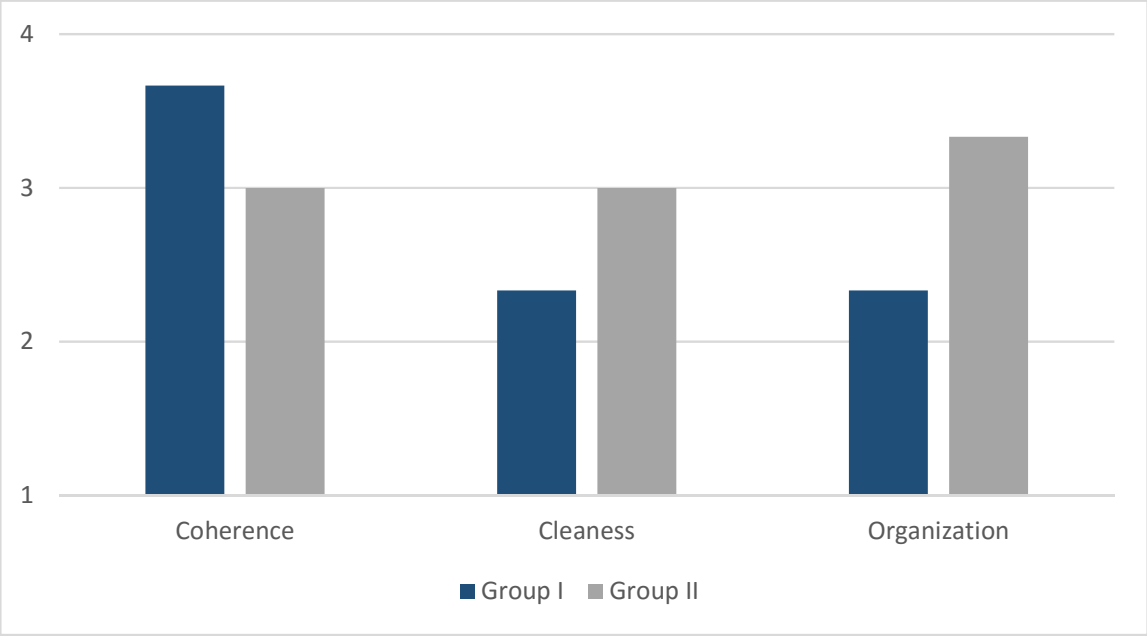


Figure 5.3: Impressions of SOAR-S representation.

Regarding discursive questions, qualitative data provided us with lessons to enhance SOAR-S. The main suggestions from subjects and further incorporated into respective documents are: (i) in SOAR-S, inclusion of more details about relationships among activities; (ii) also in SOAR-S, inclusion of more details of which information are essential in work products for Architectural Backlog and ASRs as means of enhancing guidelines concerning their conception/updating; (iii) in GEOSS SoS description, inclusion of more details about involved stakeholders; and (iv) creation of *Handling Evolution* activity and inclusion in SOAR-S.

With this study, we consider that results indicate that SOAR-S can be an adequate, comprehensive practice to support production of process instances for architectural synthesis. Results also contributed to improve SOAR-S and to verify its applicability and independence in terms of application domains.

5.2.4 Threats to Validity

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

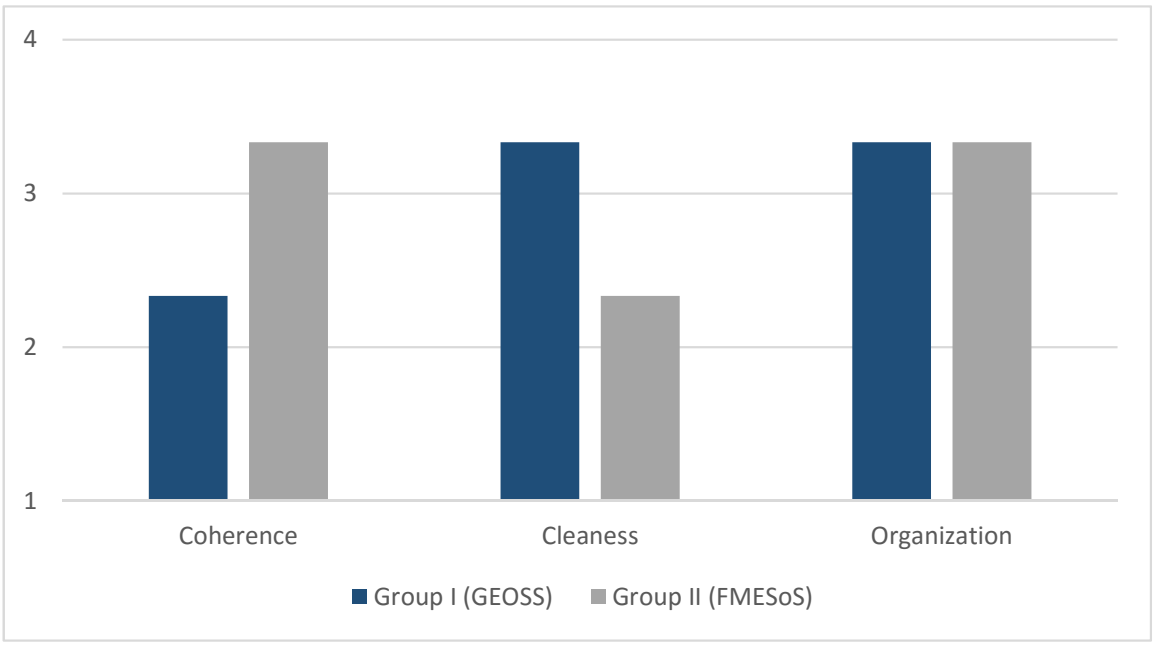


Figure 5.4: Impressions of SoS description.

Internal Validity. 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. We believe that the number of participants can be accepted since our main goal was to observe the results of using SOAR-S and gain insights and suggestions for improving it.

Construct Validity. We attempted to mitigate most of bias coming from the subjects by structuring a significant part of the form to drive the use of SOAR-S in a particular SoS context.

5.3 Evaluating SOAR-S: Second Study

In order to evaluate SOAR-S, we performed a controlled experiment with SoS researchers from academia and industry. This experimental study aimed at checking if SOAR-S could support the establishment of design processes for SoS software architectures better than those currently developed in an *ad hoc* manner, i.e., without considering any specific support for process establishment. As the study described in the previous section, this experiment was also based on the systematic process proposed by Wohlin et al. (2012).

5.3.1 Scope and Planning the Experiment

Following the GQM strategy (Basili et al., 1999), this experiment had the *objective* of analyzing SOAR-S support for generate process instances *for the purpose* of evaluation *with respect to* a set of process requirements *from the point of view of* software engineering researchers *in the context* of SoS community. Based on this goal, we established one research question (RQs):

- RQ1: Can process instances established with SOAR-S support encompass process requirements of SoS software architectures better than those currently established in an *ad hoc* manner?

In this experiment, we defined one hypothesis that is described as follows:

1. Null hypothesis, H_{A0}: There is no difference in conformance of instances generated using SOAR-S and instances generated in an *ad hoc* manner, i.e., H_{A0}: AF(SOAR-S) = AF(Control).

   Alternative hypothesis, H_{A1}: AF(SOAR-S) > AF(Control).

In this experiment, input for each subject was a document containing an overview of a specific acknowledged SoS and its development environment. Expected output was the proposition made by the subjects of activities, work products, and a workflow to perform architectural synthesis of such SoS. Therefore, each set of activities and work products might represent an instance of a process for architectural synthesis of a specific SoS.

Our experiment was performed with a group of seven researchers from academia and industry. Researchers (hereafter referred to as subjects) were chosen by convenience. We provided two descriptions of different acknowledged SoS in different application domains. In order to evaluate process instances generated by subjects, we invited two SoS experts to perform this evaluation. Based on a list of process requirements for SoS software architectures leveraged in an SLR (See Section 2.2.3), these experts initially identified what requirements were applicable as evaluation criteria to our experiment context, i.e., architectural synthesis for SoS software architecture. Final requirements set to be employed by the experts comprised 13 requirements summarized in Table 5.3. Given this set, we proposed a metric to test our hypothesis (i.e., independent variables):

- **Adequacy Factor (AF):** this metric measures how adequate the process instances established by subjects are to encompass the process requirements summarized in Table 5.3. Two experts who performed this evaluation are external from our research group and provided a consensual judgment about how many process requirements

each process instance encompassed. Therefore, AF of each process instance is the number of requirements that this instance encompassed. In this case, the higher the AF, better is the process instance.

Table 5.3: Process requirements for architectural synthesis

Requirement	Description
PR1	Consider in the architectural design what is necessary to handle the distribution of constituent systems. In case of geographically dispersed stakeholders, provide communication means to allow the architectural design.
PR2	Support prediction analysis and adequate representation of desired and undesired emergent behaviors at any stage of the architecting process.
PR3	Provide means to establish traceability among SoS missions, functionalities, emergent behaviors, and capabilities from constituent systems.
PR4	Provide means to support the architectural evolution in accordance with to SoS development.
PR5	Continuously develop, monitor, update, and refine architectural decisions and respective SoS software architecture.
PR6	Maintain the management of complex range of stakeholders ensuring their involvement in the architectural design during SoS life cycle.
PR7	Establish an architecture documentation that registers the SoS software architecture and its evolution.
PR8	Deal with quality attributes (e.g., interoperability, connectivity, and performance), providing means to earlier verify these attributes in the architecting process.
PR9	Support inclusion of self-managed constituent systems handling issues in SoS software architecture generated by these constituents, e.g., different organizations and own interests involved, development teams, and different stages of development.
PR10	Include means to handle the lack of detailed information about the internal architecture of constituents.
PR11	Manage operational impact of constituent systems, which have individual capability of operation and self-regulation. Support dynamic reconfiguration/participation of constituent systems in the SoS, facilitating both operation and evolutionary changes.
PR12	Provide means to monitor and receive continuous feedback from SoS operations and deal with deviations and changes in the operation of constituent systems.
PR13	Consider impacts and relevance of software in SoS and the relation of software with other architectural layers, e.g., physical and human.

The materials used during this experimental study were: (i) initial version of SOAR-S described in Essence Language with *EssWork Practice Workbench*; (ii) descriptions of development of two SoS projects in different application domains (i.e., GEO and flood monitoring), and (iii) a form to be filled by the subjects with a complete set of guidelines to create partial process instances to architectural synthesis based on SOAR-S description. We structured this form into three parts:

- **Part I:** personal level of knowledge in both software architecture and software engineering processes;
- **Part II:** guidelines to perform experiment activities, including creation of process instances for architectural synthesis in the context of provided SoS descriptions; and
- **Part III:** personal impressions regarding utilization of SOAR-S description and the SoS descriptions.

5.3.2 Experiment Operation

Figure 5.5 presents steps followed in the experiment operation that was conducted in a single day. Initially, subjects received general instructions about experiment documents, form, and activities to be performed. Answers of level of expertise were then collected on online forms (Part I of form previously described), doubts and misunderstandings on experiment were also clarified. After, subjects were split into two groups, each one following different sequence of activities in experiment execution.

Subjects of Group I were firstly introduced to GEOSS and trained on how to create process instances with SOAR-S support. Then, they were asked to build process instances for architectural synthesis by considering the description of a GEOSS project. After to build a process instance with SOAR-S support, the Group I was introduced to a FMSoS project and asked to build process instances to it without SOAR-S support. As shown in Figure 5.5, the Group II performed the same activities but through an inverse sequence of support, i.e., they first built instances ad hoc and after they built instances with SOAR-S support. SoS project descriptions were also switched, they first built instances for FMSoS and after for GEOSS. Furthermore, no limit of time was imposed during the instantiating activities. Before experiment conduction, we performed a pilot experiment with two participants from our research group, each one following the activities sequence proposed to one of planned groups, to verify study conformance with experiment planning and make adjustments if necessary.

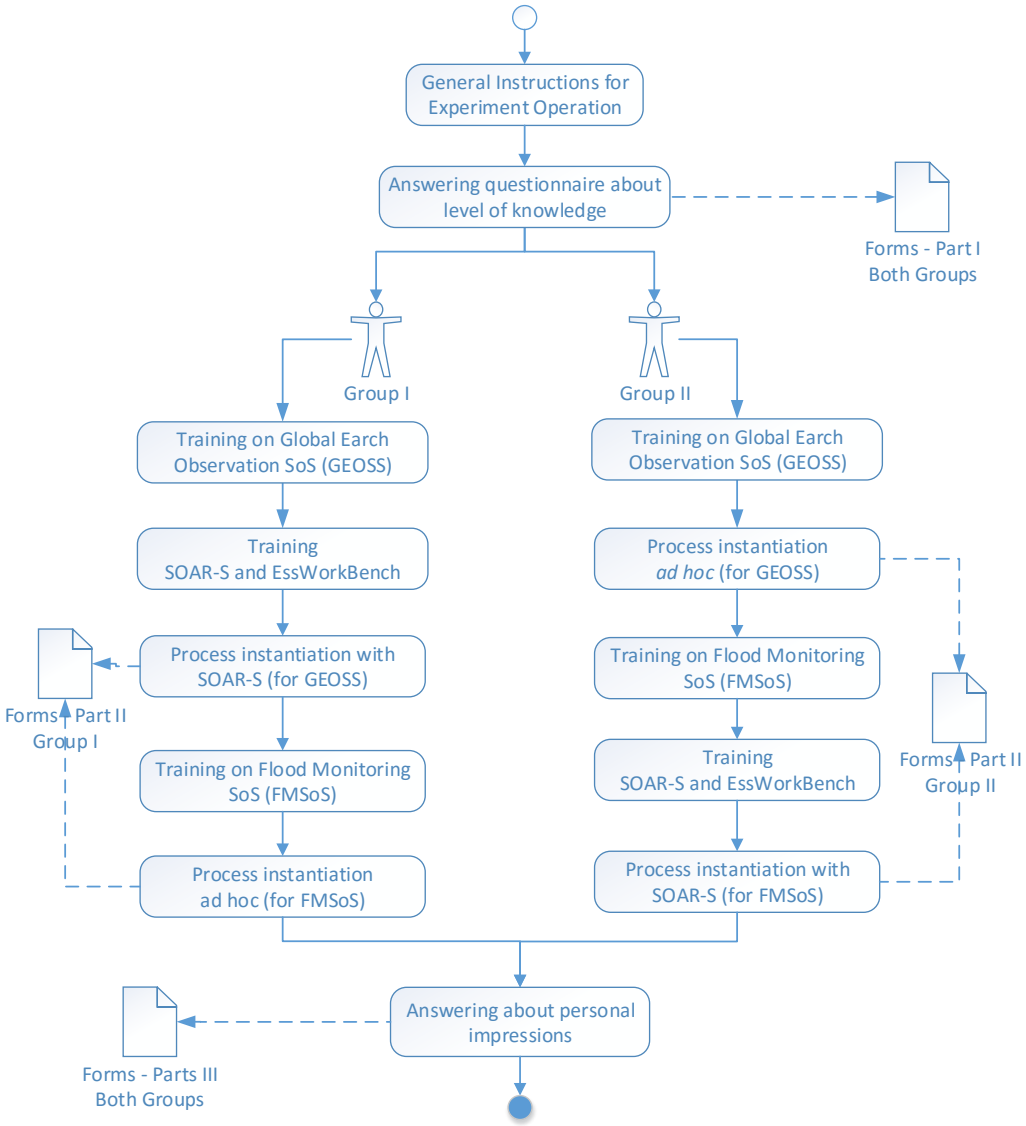


Figure 5.5: Experiment operation of second experiment

5.3.3 Analysis and Interpretation of Results

A summary of results obtained by the subjects' scores in the proposed metric (AF) is shown in Table 5.4. As previously described, each subject produced two process instances with different level of support, i.e., *ad hoc* and with SOAR-S. In this sense, each line presents individual results of a subject with the scores raised on its process instances. Furthermore, results are divided by groups, group I with three subjects, and group II with four subjects. Figure 5.6 shows box plots for the evaluated metric considering two treatments, i.e., *ad hoc* and SOAR-S.

We first analyzed results concerning individual level of knowledge. Answers revealed that all subjects had a knowledge compatible with experiment goals with no significant

differences between subjects. Next, we individually compared scores raised by subjects in their process instances generated during the experiment. In this analysis, we could notice some important trends:(i) the sequence of treatments does not influenced in best results of SOAR-S instances. Results of group I were similar to group II indicating that there was no difference from subjects learning in group II, in which the *ad hoc* was the first treatment; (ii) mean values of raised scores shown in Figure 5.6 indicate that, when supported by SOAR-S, subjects produced their process instances with higher AC than with an *ad hoc* manner; and (iii) we also could not observe significant differences in results of two different SoS project contexts. Group I built process instances for GEOSS with SOAR-S support and group II make it for FMSoS with similar performance. Due to the low number of subjects, we could not apply statistical test being restrict to observe trends in results.

Table 5.4: Second SOAR-S study: subjects individual scores on each instantiation activity

	Group I	
Subject	**AF - (SOAR-S/GEOSS)**	**AF - (*Ad hoc*/FMSoS)**
S1	12	8
S2	8	4
S3	11	6
	Group II	
Subject	**AF - (SOAR-S/FMSoS)**	**AF - (*Ad hoc*/GEOSS)**
S4	11	4
S5	10	5
S6	10	5
S7	6	3

5.3.4 Discussion and Threats to Valididy

In general, results were favorable to use SOAR-S instead to author architectural processes in an *ad hoc* manner. Results obtained in this experiment shown trends that SOAR-S can provide a suitable support to authoring design processes for SoS software architectures. In complex project environments typical of SoS, it is not possible to admit unique solutions in terms of processes and compare two or more different solutions is challenging as well. Therefore, we pursued instead for ways to support authoring of these processes meeting specific demands from SoS context. In first study described in the previous section, we try to verify if SOAR-S is applicable for this kind of support. In a second study, we evaluated how better process instances can be when supported by our approach in contrast with *ad hoc* perspective currently practiced in real SoS environments. The base of comparison

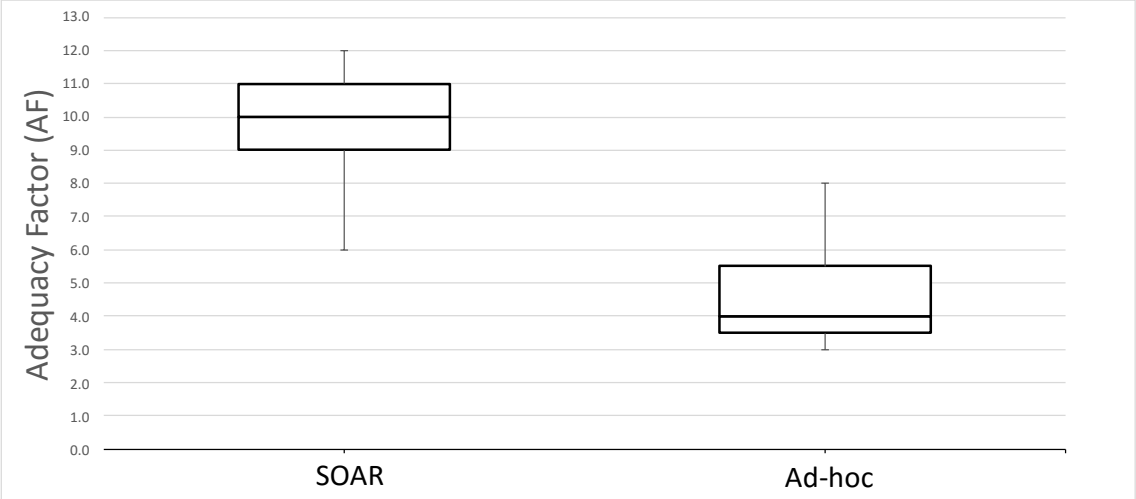


Figure 5.6: Impressions of SoS description.

is a list of process requirements carefully leveraged as a result of an SLR and agreed by experts who effectively performed evaluation. We consider that case studies in such complex projects are a very difficult and cost expensive choice that demands for more reliability from approaches to be evaluated. Therefore, we believe our results are valid as indicators of trends that enhance the reliability of future case studies in real SoS projects.

We were aware of threats to the validity of our experiment. Following, we briefly discuss these limitations and our efforts to mitigate them.

Internal Validity. To avoid problems of internal validity, we carefully designed, piloted, and iteratively refined form and documentation provided to subjects. Additionally, we decided to perform the experiment in a single day, kept two groups separated during conduction, prevented communication among subjects, and made the participation voluntary and anonymous.

External Validity. We believe that the number of participants can be accepted since the experiment context demands for subjects with too specific knowledge. Since real population is naturally small, we consider that even a limited number of subjects from this population can produce important results indicating relevant trends.

Construct Validity. To mitigate most of bias coming from evaluation of process instances we combine two different sources of knowledge, agreement of two researchers and the list of process requirements produced from related literature on architecting SoS software architectures. Furthermore, as we decided to perform the experiment in a single day, parts of design process had to be simplified or omitted, which could result in a threat to construction validity. Nevertheless, the architectural synthesis is a macro-activity of

the design process relevant to be investigated, even using a simplified version of SOAR, and able to be evaluated in a SoS project context.

Conclusion Validity: A main threat of our study is related to not be possible to provide evidences with statistical significance to refuse null hypothesis. For reducing possible issues associated with conclusion validity, our main goal was restricted to observe trends in the results of using SOAR-S.

5.4 Final Remarks

This chapter presented SOAR-S, a SOAR practice to support the establishment of processes for architectural synthesis of acknowledged SoS. As other SOAR practices, SOAR-S is also represented in Essence Language and extends SOAR Kernel. Contributions of this chapter are: (i) SOAR-S to guide architectural synthesis on acknowledged SoS; (ii) a SOAR-S documentation built in Essence Workbench Tool that enables instantiation of processes for particular projects; (iii) a study with six SoS researchers from academia and industry to verify if SOAR-S is capable to support process instantiation; and (iv) an experimental study with seven SoS researchers. This experimental study aimed at checking if instances produced with SOAR-S can be better than those ones developed in an *ad hoc* manner. Results from two studies indicate that SOAR-S is adequate to architectural synthesis in acknowledged SoS. Next chapter presents SOAR-E, the subsequent practice of SOAR to the establishment of processes for architectural evaluation of acknowledged SoS.

Chapter

6

SOAR-E: A Practice for Architectural Evaluation on Acknowledged SoS

This chapter introduces the SOAR-E, a SOAR practice to the establishment of processes for architectural evaluation of acknowledged SoS. Section 6.1 describes SOAR-E. This description refers to the final version of SOAR-E, produced after a survey conducted for evaluating it that is presented in Section 6.2. Final remarks are presented in Section 5.4.

6.1 Description of SOAR-E

Due to the relevance of software architectures on determining system structure and quality, their evaluation is essential to avoid further failures in any project of software-intensive system. Therefore, evaluating software architectures is an essential activity in development processes, even more in large complex systems (Kazman et al., 2012). In SOAR-E, evaluating a SoS software architecture is to determine the degree in which new candidate architectures (i.e., CASs) satisfy requirements (i.e., ASRs) and if proposed changes can cause architectural degradation regarding ASRs. As previously discussed (see Chapter 2), SoS differ from monolithic systems since they involve a set of characteristics, in which a global architecture contains constituent systems that have own architectures not mandatorily visible at SoS level. SOAR-E includes activities and work products to the

architectural evaluation of acknowledged SoS, taking advantage of concepts and elements already defined in SOAR Kernel (see Chapter 3).

In software development processes, the main two opportunities to perform architectural evaluation are before and after implementation (Clements et al., 2002). Evaluating before implementation is relevant to reduce cost to solve problems in design decisions after implementation (Shanmugapriya and Suresh, 2012). Evaluate after implementation is relevant to verify if the as-built SoS conforms with the as-designed one (Shanmugapriya and Suresh, 2012). In SOAR, both these evaluations are supported by SOAR-E. As a part of SOAR, SOAR-E is also described upon OMG's Essence Standard and documented in Essence Workbench Tool. Activities, alpha states, and work products of SOAR-E are described as follows.

6.1.1 SOAR-E Activities

Figure 6.1 shows SOAR-E general workflow, its activities, and activity spaces inherited from SOAR Kernel. Activities of SOAR-E are described as follows:

Evaluating deviations between current architecture, i.e., last validated version, and architecture really implemented in SoS. This evaluation is called late evaluation and intends to identify problems between architecting and development processes (Shanmugapriya and Suresh, 2012). This evaluation is even more relevant in SoS, since architectures of constituents are sometimes not accessible to SoS architectural teams and characteristics agreed with constituent owners must be verified after their implementation.

Planning Evaluation: As in other SOAR practices (i.e., SOAR-A and SOAR-S), strategies established in SOAR-E be also reviewed and updated on each design iteration. In this activity, the architectural team must establish/update a plan for architectural evaluation activities according to development stage. Following, we list some possible issues and related tasks to be considered in this activity:

- Determine which strategies, such as techniques, tools, and technologies, must be used for evaluation. Bosch (2000) classifies these strategies in four main groups: (i) experience-based, which depends on previous experience and domain knowledge of evaluators; (ii) simulation-based, in which simulation techniques can be employed to emulate the SoS generated by an architecture. Simulation-based can help on checking emergent behaviors and other aspects not statically verifiable; (iii) mathematical modeling, which employs mathematical models to come up with mathematical proofs for verify quality requirements; and (iv) scenario-based, in which evaluation is based on scenarios built to describe a complete intended use of the system under evaluation. In general, these scenarios are related to required quality attributes and

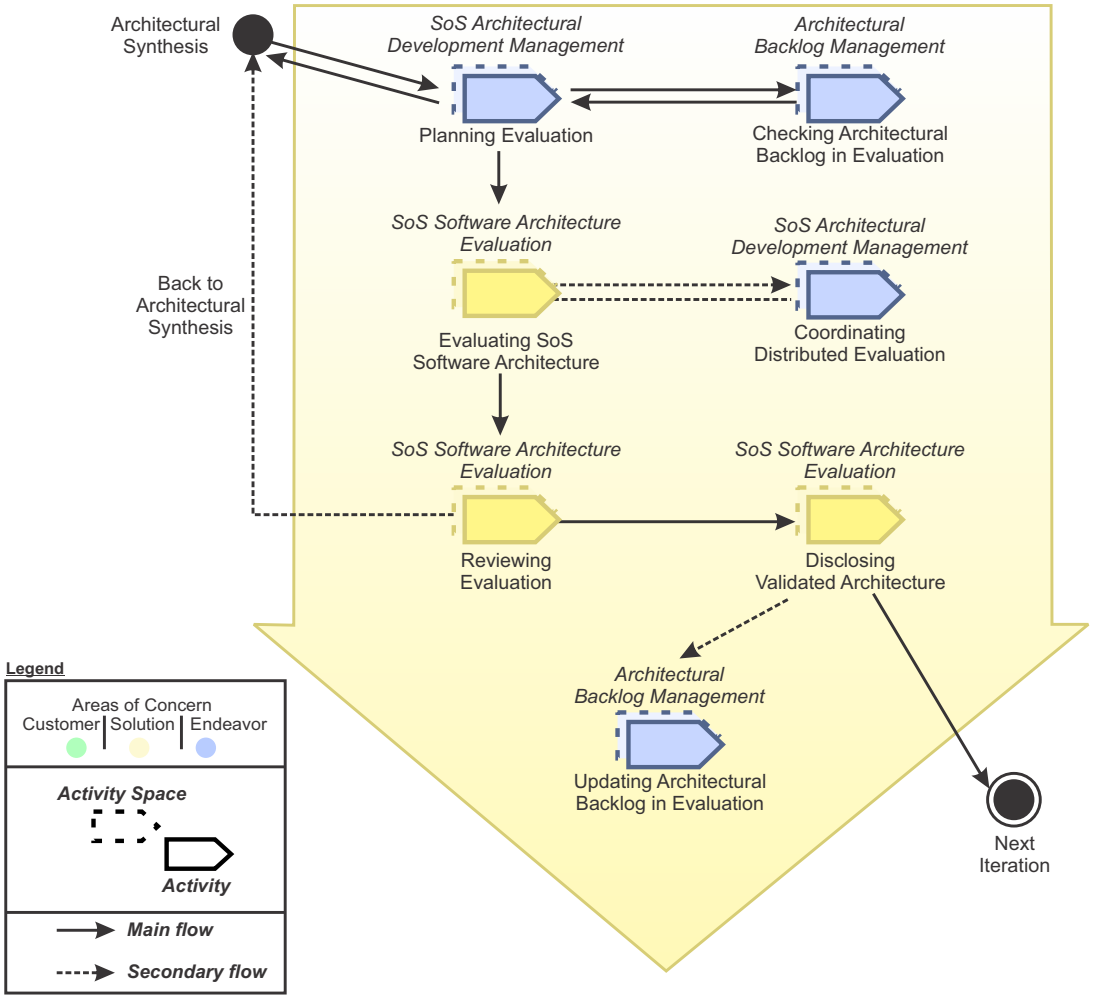


Figure 6.1: SOAR-E activities workflow

help on identifying architectural risks and their potential impacts in system operation (Kazman et al., 1998). Scenario-based is specially useful in SoS, because it is a way to evaluate SoS focusing on global scenarios of operation, in which complete information of constituent systems architectures can be not available. In this case, evaluation can use information of expected behaviors agreed with constituent systems providers;

- Analyze previous versions of evaluation plan, after first iteration, to come up with a new plan determining which activities of SOAR-E must be performed;

- Determine disclosing and copyright rules, i.e., who must have access to the information report; and

- Estimate resources necessary to conduct evaluation activities. For example, analyze the relevance of the new CASs proposed in synthesis in terms of size, complexity, number of alternative CASs, and impact in whole SoS software architecture. This

analysis must guide how deep evaluation shall be and how many resources it demands. Regarding human resources, evaluation of software architectures typically includes the participation of more people who did not participated in the conception of CASs (Bass et al., 2012). Depending on each development project, different structures can be considered to organize people to be engaged as evaluators and their specific roles. We recommend three main groups based on ATAM (Kazman et al., 1998)[1]: (i) an evaluation team with the role of ensuring that evaluation occurs as planned. It can be formed by members of architectural team or external ones; (ii) decision makers team must be formed by people who have authority to mandate changes in the architecture; and (iii) a group of SoS stakeholders, which is a more generic group formed by selected stakeholders that must determinate how satisfactory is the way how CASs encompass ASRs. Developers of constituent systems represent a relevant type of stakeholders, since they can help the evaluation through the perspective of who must reflect architectural decisions at SoS level to lower level of constituent systems.

Once evaluation strategies are planned, this activity must also encompass the preparation of an environment that allows evaluation in accordance with established plan. This environment involves several issues that must demand for preparatory tasks, such as:

- Confirm who performs the evaluation, contacting potential participants and ensuring participation of who are essential;
- Negotiate and establish with participants an agreed schedule of activities in accordance with evaluation plan;
- Produce any additional documentation to follow the evaluation strategies; and
- Prepare evaluation participants with all information needed to perform evaluation. It may include workshops, presentations, and coaching surrounding SoS software architecture.

Checking Architectural Backlog on Evaluation: In architectural evaluation, the architectural backlog must be checked to supply the planning activity with any information potentially relevant to perform architectural evaluation. Some examples are: (i) reporting problems or process enhancements ideas registered during evaluation in previous iterations; (ii) and external information from other processes, e.g., problems identified with independent evaluations performed by constituent systems owners, general systems

[1]Some of these recommendations are based on ATAM in light of SoS challenges, since it is a mature evaluation method for software architectures already employed in some SoS projects (Chigani and Balci, 2012; Kazman et al., 2012).

engineering processes of SoS can register on architectural backlog information of new risks and quality attributes also relevant to software architecture.

Evaluating SoS Software Architecture: In this activity, architectural evaluation must be conducted in accordance with evaluation plan. Inputs of this activity are evaluation plan, CASs, related documentation from analysis and synthesis, and supporting documentation produced in the Planning Evaluation activity. The main output must be an evaluation report describing findings, such as problems, strengths, and weaknesses of CASs and, in case of different alternative CASs, results comparing them. This activity can demand more or less efforts according to several issues, such as the strategies proposed to evaluate the architecture, size of CASs, and SoS level of maturity. Information produced in this activity must feed not only SoS implementation but also further iterations of architecting process. The main issues recommended to be considered in this activity are:

- Identify problems that new CASs can bring to the SoS;
- Check if emergent behaviors predicted in synthesis are truly accomplished by the software architecture;
- Identify potential sources of architectural degradation, in which changes in architecture cause its degradation regarding already raised ASRs;
- Propose design solutions to meet identified problems and new enhancing opportunities. Evaluators must also indicate if propositions must be directly applied to implementation or if they must be registered to be better explored in architectural design; and
- Analyze architectural risks, identifying which CASs may lead to undesirable consequences for SoS missions.

Coordinating Distributed Evaluation: This activity is necessary to manage the participation of heterogeneous and distributed groups of stakeholders. It must occur in parallel to *Evaluating SoS Software Architecture* activity, encompassing tasks to support this collaborative work. For example, making adaptations in evaluation plan, managing communication in the distributed work, and organizing of evaluation meetings.

Reviewing Evaluation: This activity encompasses both report of the evaluation and establishment of an upshot of the evaluation. Based on results of *Evaluating SoS Software Architecture* activity, architectural team and stakeholders with decision authority must agree about the feasibility of implementing a new architectural version. If there is a consensus that the SoS software architecture resultant of applying CASs is unfeasible,

architectural team must to go back to synthesis to come up with new CASs. In this case, review evaluation activity must also include preparatory tasks to this backtrack. For example, changing time schedule in the development project, estimating additional resources needed to redone activities, and establishing agreement between architectural team and decision makers concerning new efforts. In case of approval for implementation, the *Disclosing Validated Architecture* activity must be normally followed. Reviewing evaluation must also include production of feedback information including all justifications of what was deliberated. In case of partial approval, architectural team must deliberate if approved CASs are enough and consistent to deliver an architectural version to be implemented. Furthermore, all relevant information related to evaluation must be reported as an output of *Reviewing Evaluation* activity. Following, we recommend some issues to be considered when authoring tasks to accomplish this reporting on each SoS project:

- Produce an evaluation report including all relevant information generated during the evaluation;
- Structure documentation by considering the different target audiences in accordance with copyright and privacy rules; and
- Perform meetings and presentations to transmit results to different audiences.

Updating Architectural Backlog on Evaluation: This activity registers any relevant information leveraged in evaluation and not included in other work products of evaluation activities For example, new design ideas, considerations for other architecting stages in future iterations (i.e., architectural analysis or architectural synthesis), enhancement suggestions for architectural evaluation strategies in future iterations, and description of possibly future trade-offs.

Disclosing Validated Architecture: When an evaluated set of CASs is approved, the representation of the current SoS software architecture must be updated and disclosed as a new architectural version with new CASs to be implemented. Documentation of SoS software architecture can be built through all architecting process, specially during architectural synthesis activities. However, only after evaluation, an approved set of CASs can be documented and disclosed in the ongoing SoS software architecture. In this context, this activity updates the SoS software architecture documentation to reflect these CASs. Furthermore, this activity is also a way to convey the updated SoS software architecture reaching all interested stakeholders. Different disclosure strategies can be considered for different stakeholders, such as presentations, meetings, and coaching on SoS software architecture. If necessary, access can be also limited, e.g., providers of constituent systems can receive information with focus on specific parts of the SoS software architecture.

6.1.2 SOAR-E Alpha States and Work Products

In architecting process, evaluation activities must generate different work products and change alpha states of this process. Work products of SOAR-E express some alphas inherited from SOAR Kernel. Following we present the states variation of these alphas when performing SOAR-E and describe the work products of it:

Architectural Backlog: Execution of SOAR-E activities must ensure maintenance of *updated* state for this alpha, in which it is consulted in *Checking Architectural Backlog on Evaluation* activity and fulfilled in *Updating Architectural Backlog on Evaluation* activity with any relevant information unforeseen in other work products of SOAR-E. The work product that express this alpha is the *Evaluation Backlog*, which must document backlog information from *Updating Architectural Backlog on Evaluation* activity.

Architecturally Significant Concerns (ASCs): ASCs are a source of information for SOAR-E activities. The required initial state for this alpha is *established*, in which a set of ASCs is identified and agreed in the current development stage. After evaluation activities, this alpha must reach the *fulfilled* state only if all ASCs related to proposed CASs were satisfied, i.e., the architectural decisions employed to address ASRs were verified as effective ones also satisfying ASCs. In case of partially addressing them, the ASCs maintain the state *established*. A possible work product associated to this alpha is a *List of concerns*. It must be generated in analysis phase, providing description to the ASCs. SOAR-A practice already describes guidelines concerning this work product (see Chapter 4).

Architecturally Significant Requirements (ASRs): Since ASRs must be confronted to the CASs, this alpha is an expected input for SOAR-E activities. It must present at least the *established* state, in which ASRs were established and agreed, expressing problems that software architecture must solve. The work product associated to this alpha is the ARSs documentation. It must be generated during the architectural analysis, providing description to the ASRs. SOAR-A practice already describes guidelines concerning this work product (see Chapter 4).

CASs: CASs must be produced during architectural synthesis (see more information about CASs proposition in Chapter 5). In SOAR-E, this alpha must present at least the *proposed* state, in which a new set of CASs was proposed and agreed to meet one or more ASRs. After execution of SOAR-E activities, the expected state to be reached is *evaluated*, in which there is an agreement regarding the acceptance of CASs. The main condition to reach this state is the conclusion of *Evaluating SoS Software Architecture* activity. The work product associated to this alpha is a CASs description to be incorporated to the description of SoS software architecture.

SoS Development Environment: Execution of SOAR-E activities must ensure maintenance of the *working* state, in which the distributed environment is supported to adequately work to architectural development of SoS. The work product of SOAR-E in which this state maintenance can be verified is the *Evaluation Plan*, which documents a planning to evaluate SoS software architecture.

Acknowledged SoS: Before architectural evaluation, this alpha must present at least the *designing* state, in which a set of CASs were proposed to ASRs as an increment to the SoS software architecture. If this set of CASs had success during architectural evaluation, the expected state to be reached is *designed*, in which these CASs were approved to be further implemented. In case of failure, *designing* state is maintained and SOAR-E activities must support backtracking to new architectural synthesis to conceive new CASs. There is no work product in SOAR-S to directly express this alpha, but project teams can schedule and add additional documentation if necessary.

Constituent Systems: The execution of SOAR-E activities must ensure the maintenance of the *negotiated* state for this alpha, in which constituent systems providers/authorities agreed with the ASRs and respective CASs proposed to the SoS software architecture. There is no work product in SOAR-E to directly express this alpha.

Emergent Behaviors: As an expected input to SOAR-E activities, this alpha must present at least the *predicted* state, in which both desired and undesired behaviors are predicted during architectural synthesis. After execution of SOAR-E activities, the expected state to be reached is *checked*, in which the evaluation strategies are applied confirming the emergent behaviors of SoS as in accordance with the ASRs. There is no work product in SOAR-E to directly express this alpha.

Software Architecture: The previous state required to this alpha is the *represented* state, in which there is an agreement about proposed CASs and they are adequately persisted in architectural representation. After execution of SOAR-E activities, two different states can be reached according to evaluation results: (i) *validated* state, if CASs are totally or partially approved for implementation; and (ii) *ASRs selected* state if they are not approved. In this scenario, a redesign is necessary and the architecting process must back to synthesis. Regarding work products, a previous *SoS Software Architecture Documentation* is expected as an input already produced during architectural synthesis (see Chapter 5). If necessary, this documentation can be updated during evaluation, for example, when architectural team discards some alternative CASs after evaluation. Furthermore, *Evaluation Report* is a second work product related to this alpha that must be produced/updated during evaluation activities. For main content of this work product, we recommend the following points:

- Organize report documentation by considering two main levels of target audiences, i.e., stakeholders at both levels SoS and constituent systems;
- Describe architectural risks. This description can include risk themes that express systemic weaknesses in the architecture or even in the architecting process (Kazman et al., 1998);
- Indicate new opportunities for enhancing the SoS software architecture; and
- Support testing activities with relevant information, e.g., describe critical quality scenarios, if scenarios were used to describe ASRs, or key ASRs to be explored in testing activities.

Architectural Team: Execution of SOAR-E activities must ensure the maintenance of the *performing* state for this alpha, in which the architectural team must be working in architectural evaluation. The work product proposed to expresses this alpha is the *Work Scheme*. It is a work product common to all architecting phases, i.e., analysis, synthesis, and evaluation. In this sense, it is already described in SOAR-A practice (see Chapter 4).

Stakeholders: Execution of SOAR-E activities must ensure the maintenance of the *in agreement* state for this alphas. If validated, stakeholders must agree with the introduction of CASs. There are no work products in SOAR-E to directly express these alpha. However, work products that support the understanding of stakeholders can be previously conceived during architectural analysis and used as information source for evaluation activities (see work products proposed to these alphas in Chapter 4).

In order to illustrate how aforementioned work products can relate to activities of SOAR-E, Table 6.1 presents for each activity its related work products and kinds of use, i.e., input/output.

6.2 Evaluation

Following the same strategy employed to evaluate of SOAR Kernel and SOAR-A, we evaluated SOAR-E by conducting a survey with experts from both academia and industry who have been involved in SoS development. Similar to the surveys described in previous chapters (see the survey conducted for SOAR Kernel in Section 3.2 and the one conducted for SOAR-A in Section 4.2), this study was also based on the survey steps proposed by Kasunic (2005) that comprises: (i) identify research objectives; (ii) identify & characterize target audience; (iii) design sampling plan; (iv) design & write questionnaire; (v) pilot test questionnaire; (vi) distribute questionnaire; and (vii) analyze results and write report. These steps are described as follows:

Table 6.1: Work products produced/updated in SOAR-E activities

Work Product / Activity	***Evaluation Plan***	***Arch. Backlog***	***Evaluation Report***	***SoS SA Doc.***	***Work Scheme***
Planning Evaluation	Output	Input	Input	-	Input/ Output
Checking Architectural Backlog on Evaluation	Input	Input	-	-	-
Evaluating SoS Software Architecture	Input	-	Input/ Output	Input	-
Coordinating Distributed Evaluation	Input	-	-	-	Input/ Output
Reviewing Evaluation	Input	-	Input/ Output	Input	-
Updating Architectural Backlog on Evaluation	Input	Output	Input	-	-
Disclosing Validated Architecture	Input	-	-	Output	-

1. **Identify research objectives:** This survey aimed to verify if SOAR-E meets expectations of the SoS community as a supporting approach to architectural evaluation on SoS. Four research questions (RQs), summarized in Table 6.2, guided this survey.

Table 6.2: Survey research questions

SOAR-E Practice	
RQ1	Does SOAR-E encompass all common issues related to architectural evaluation on SoS software architectures?
RQ2	Is SOAR-E correct, with no wrong or misunderstood statements?
RQ3	Is SOAR-E conceptually coherent, with no conflicts or wrong placed elements?
RQ4	Can SOAR-E be considered intelligible and easy to use?

2. **Identify and characterize target audience:** The target audience is represented by potential SOAR users. We sampled this population by considering SoS researchers who have conducted studies and/or development projects on SoS and

developers who have constructed SoS. The level of expertise, was also collected to support further analysis of answers.

3. **Design sampling plan:** Considering the same reasons presented in the survey conducted for SOAR Kernel (see Section 3.2), sample size was limited by individuals who agreed to participate, i.e., a set of seven experts.

4. **Design and write questionnaire:** By following the same strategy of previous surveys, we gathered answers with an on-line self-administered questionnaire[2]. The questionnaire was guided by research questions presented in Table 6.2 and included non-discursive (closed) and discursive (open) questions. In closed questions, participants were asked to provide a score to evaluate elements of SOAR-E. In open questions, participants provided textual answers justifying their scores as well as additional comments and improvement suggestions. Additionally, participants were provided with following documentation[3]: (i) guidelines to read documentation and answer questionnaire; (ii) a profile questionnaire to verify the level of expertise of the participants; (iii) complete description of the SOAR-E; and (iv) a support documentation about both Essence Language and Essence Kernel.

5. **Pilot test questionnaire:** A pilot survey was executed with participation of one researcher from our research group. Data were collected only to verify if average time for response is affordable, possible errors and enhancement in first version of survey material.

6. **Distribute questionnaire:** A list of experts were invited to participate and survey was sent to whom has agreed to collaborate. Then, three steps were proposed to participants: (i) answering questions about level of expertise;(ii) reading survey's documentation; and (iii) answering questions about SOAR-E.

7. **Analyze results and write report:** For scale-based questions, graphics were associated to a textual discussion to illustrate observed trends. Due to the limited number of survey participants, no statistical test were applied. Discursive questions provided insights to enhance SOAR-E. Following section presents obtained results and discusses how they help to enhance this practice.

6.2.1 Analysis and Interpretation of Results

To analyze the survey results, we first identified personal profile of participants. Figure 6.2 shows level of expertise in a scale ranging from zero (beginner) to three (expert) for both

[2]see the questionnaires of all surveys of this Thesis in Appendix E.
[3]Questionnaires of all surveys of this thesis are included in Appendix E.

SoS and software architectures. Collected information indicates that all participants have at least a level of knowledge on SoS and software architectures. We also asked for their occupation area. In this case, the predominance was of academic researchers and only two participants work as industry practitioner.

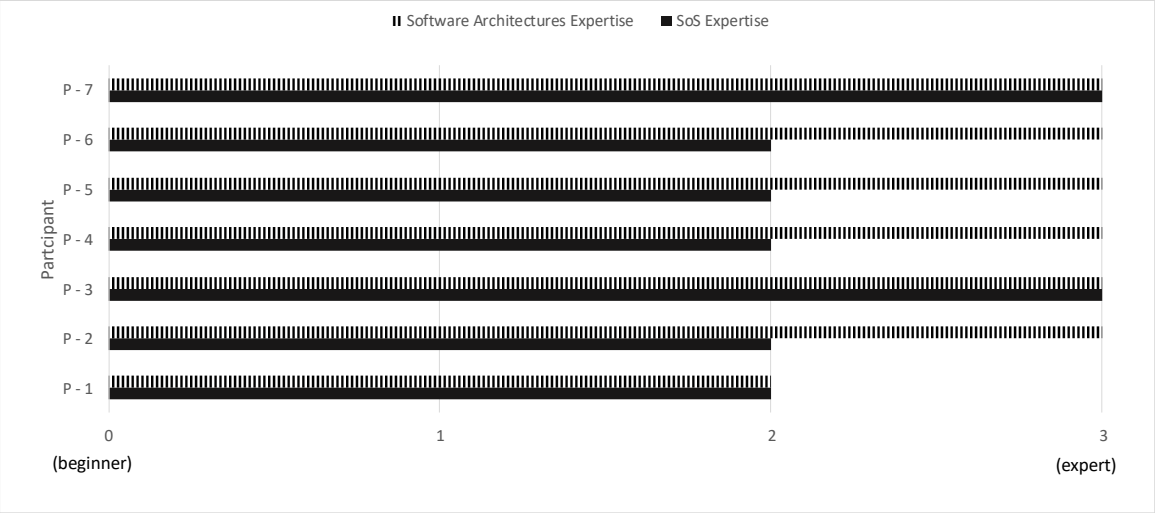


Figure 6.2: Levels of expertise on SOAR-E Survey

Regarding previously established RQs, results for each one are detailed as follows.

RQ1. Three questions were related to this RQ, two of them non-discursive and a discursive one. Figure 6.3 summarizes results of non-discursive questions. The first question asked for participants how complete work products of SOAR-E are to the architectural evaluation of acknowledged SoS. In second non-discursive question, participants graded the completeness of the activities contained in SOAR-E. A discursive question asked participants to point out what is missing in SOAR-E. Answers help us to enhance SOAR-E, in particular, we could provide in "planning evaluation" activity more information about possible tasks to be considered in a process instance to evaluate SoS software architectures.

RQ2. This RQ had two associated questions, a discursive and a non-discursive one. Figure 6.4 shows scales pointed out by participants concerning how correct SOAR-E is in terms of correctness. In general, participants considered SOAR-E as correct. Furthermore, errors pointed out by some participants in discursive question were only minor errors in some figures and typographical errors.

RQ3. This RQ was structured into two questions (a discursive and a non-discursive one) about the SOAR-E coherence. The non-discursive question asked the participants

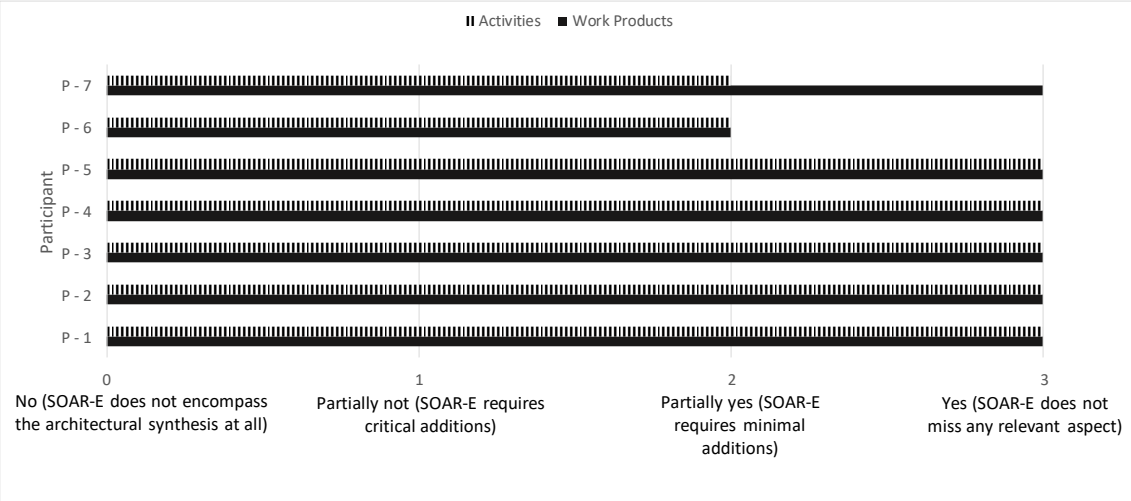


Figure 6.3: SOAR-E Survey: RQ1 results of non-discursive questions

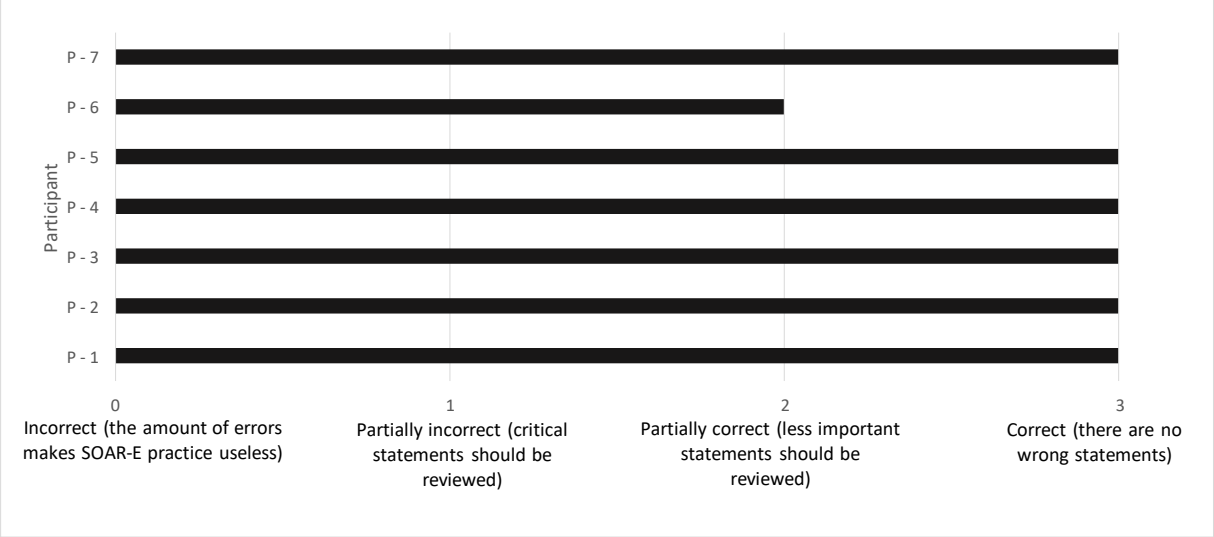


Figure 6.4: SOAR-E Survey: RQ2 results of non-discursive questions

for grading SOAR-E general coherence. Figure 6.5 shows results of the non-discursive question, indicating that participants judged SOAR-E as coherent. However, in the discursive question, participants indicated possibilities of coherence corrections. In particular, the description of some work products had to be better aligned with the checklists of alpha states described in SOAR Kernel.

RQ4. This RQ had two associated questions, a non-discursive and a discursive one. Figure 6.6 shows results of the non-discursive question that evaluated SOAR-E description in two dimensions: clearance and organization. Results indicated general acceptance of SOAR-E as clear and well-organized, however, we did not obtain to-

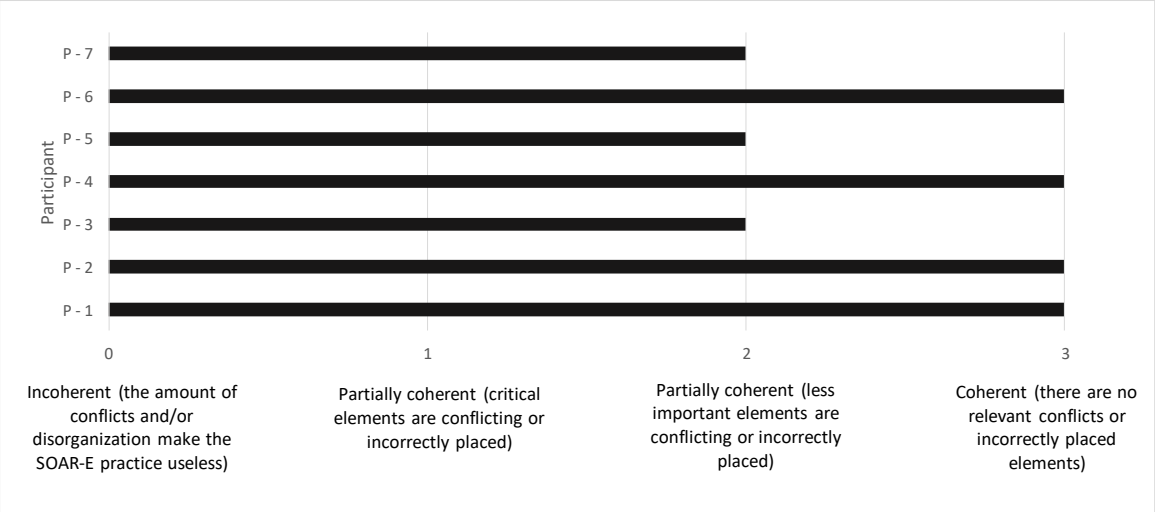


Figure 6.5: SOAR-E Survey: RQ3 Results of non-discursive questions

tal satisfaction from participants. Analyzing discursive question, we identified some possibilities of enhancements in description of activities and work products with increment of suggestions of issues to be considered as specific tasks in process instances. We also verified that despite facility to understand specific elements and concepts of the Essence Language, as some participants had their first experience with this language during the survey, they reported some extra effort to understand SOAR-E.

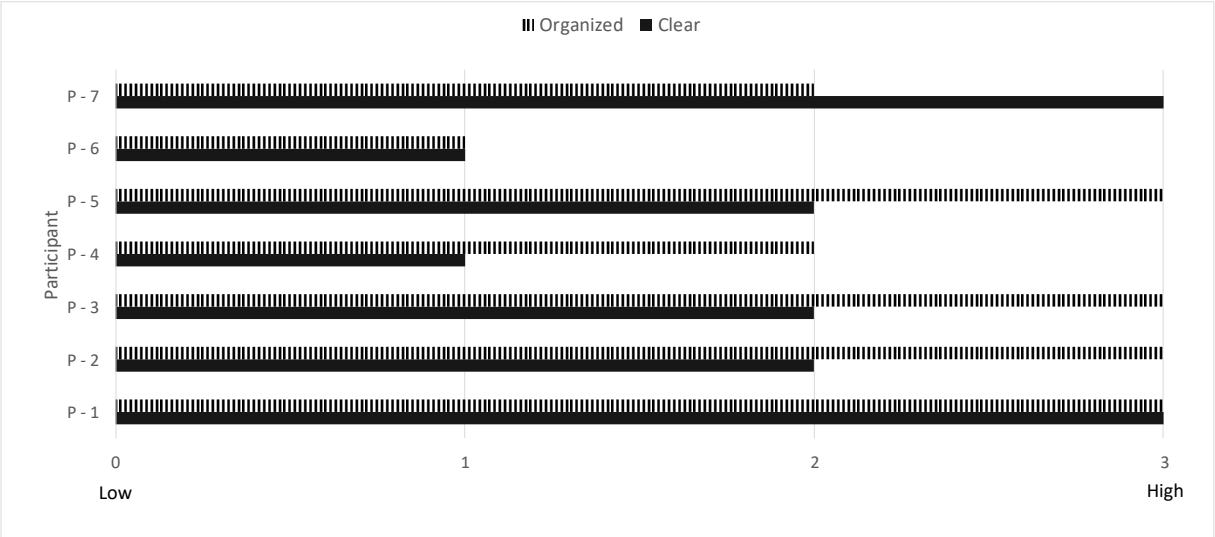


Figure 6.6: SOAR-A Survey: RQ4 results of non-discursive questions

The positive evaluations showed us that SOAR-E is described without severe problems and its representation in Essence Language was enough clear to enable participants to

understand its content. These results represent a good indicative that SOAR-E can be used to its application purposes. Regarding possible threats to validity, as this survey followed the same methodology of the SOAR-A's survey, we consider same limitation previously described for this survey (see the discussion of these threts in Section 4.2.2).

6.3 Final Remarks

Architectural evaluation is quite relevant for any software-intensive system, predicting impact of design decisions, avoiding later problems, and reducing budget. This chapter presented SOAR-E, a SOAR practice to the establishment of processes for architectural evaluation of acknowledged SoS. It resulted from an analysis of state of the art in related literature in conjunction with experience of experts in SoS and software architectures. Contributions of this chapter are: (i) a practice named SOAR-E that encompasses architectural evaluation of acknowledged SoS; (ii) SOAR-E documentation, built in Essence language and available to use in Essence Workbench Tool; and (iii) a survey with experts of SoS community SOAR-E which revealed good acceptance of SOAR-E. Next chapter concludes this thesis, summarizing main contributions, general limitations, and prospecting future work.

Chapter

7

Conclusions

SoS have been shown to be a suitable system class to develop software-intensive systems in several and even critical application domains. Construction of software-intensive systems under SoS inherent characteristics can result in systems with higher autonomy, dynamicity, and operational flexibility to accommodate changes. However, these characteristics also bring several challenges for engineers, architects, and developers to create complex systems formed by other independent constituent systems. Main examples of these challenges include: (i) enable an evolutionary development in a system formed by other independent systems; (ii) tackle changes/reconfiguration during SoS operation to accommodate changes in both SoS and their constituent systems; and (iii) deal with large-scale global missions involving multiple organizations and interests. In this scenario, SoS development has arisen as an important, new research area, with contributions from different research communities, e.g., systems engineering and software engineering. The importance of developing novel theories and technologies to SoS development has been highlighted in roadmaps targeting year 2020 and beyond (Cordis, 2012; Nielsen et al., 2015).

From another perspective, software architectures have been considered an essential element to produce high-quality systems (Kruchten et al., 2006; Shaw and Clements, 2006). Decisions made at the architectural level directly interfere with the achievement of systems missions. In this sense, software architectures have been also considered an essential element to the success of SoS (Brondum and Liming, 2010; Jamshidi, 2008b; Maier, 1998;

Schaefer, 2005). The architecture of an SoS usually involves a number of complex constituent systems, different technologies, and several development teams applying different approaches to develop these constituents. Design processes for such architecture involve the conception of operations, functions, behaviors, internal and external relationships, and dependencies regarding SoS and their constituents (DoD, 2008). Despite the existence of some initiatives to support design of SoS, most of these software architectures are still designed using ad hoc processes.

Regarding this lack, this thesis focus on guiding the authoring of architectural design processes for acknowledged SoS. Our main goal is to support process managers to create instances of their architecting processes. Contributions of our work include: (i) a conceptual model for characterizing software-intensive SoS; (ii) a list of process requirements for developing SoS software architectures; and (iii) a general process for designing acknowledged SoS software architectures. Main contributions of this thesis is revisited in Section 7.1. Section 7.2 summarizes limitations of the work, how these limitations can be overcome, and directions for further research.

7.1 Revisiting the Thesis Contributions

In this thesis we presented SOAR, a high level process to support architecting process instantiation for acknowledged SoS software architectures. Since SOAR is focused on acknowledged SoS, it deals with particular challenges of these category, such as the need of negotiation with constituent systems providers and the low control of their individual architectures. In this context, Figure 7.1 depicts the research goals and main thesis contributions, including SOAR main elements and how they are related to our research. SOAR includes SOAR Kernel and three practices, i.e., SOAR-A, SOAR-S, and SOAR-E. These contributions are fivefold, each one summarized in the following.

A conceptual model and a framework for software-intensive SoS. As discussed in Section 2.1.1, there is an absence of a consensual understanding about what defines a software-intensive SoS. Regarding this lack, we established a common understanding about software-intensive SoS that supports the analysis and classification of this class of systems. With this model, it is possible to analyze and classify a given system, clearing if it is a SiSoS or not and to which category of SoS it belongs, i.e., virtual, collaborative, acknowledged, or directed. In order to build this model, we considered an extension of a standard for software-intensive systems (ISO/IEC/IEEE, 2011) aligned with inherent SoS characteristics found in the existing literature (DoD, 2008; Firesmith, 2010; Jamshidi, 2008a; Maier, 1998; Sauser et al., 2010). Moreover, we also provided in Section 2.2.2 a framework for the characterization of research into SoS Software Architec-

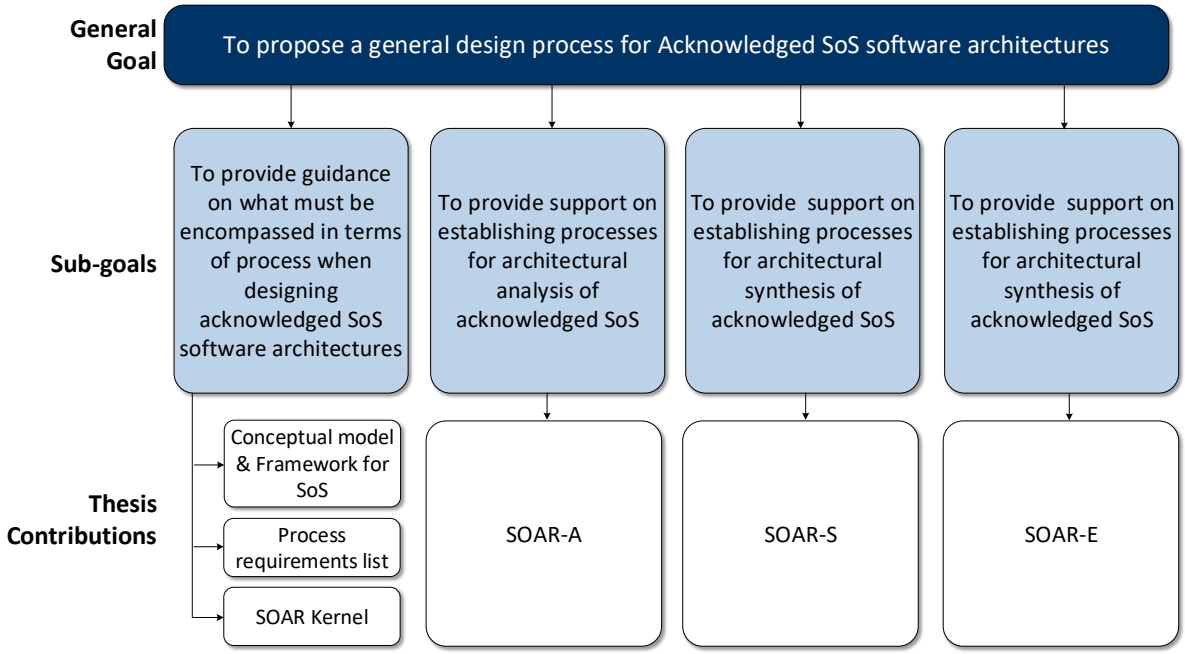


Figure 7.1: Main thesis contributions and correlation to research goals

tures. This framework is the result of an iterative process of classification of the primary studies found in our SM presented in the same section. It presents the most common and apparently stable elements/classes for the primary studies found and can be useful for the understanding of a given study or a set of studies. We expect that both this both contributions can help SoS community to align its knowledge about SoS software architectures.

A list of process requirements for designing SoS software architectures. As described in Section 2.2.3, the second contribution of this work is a list of requirements to express what any architecting processes should satisfy to adequately support the design of SoS software architectures. Resulting from a systematic literature review that investigated the state of the art on architecture design processes for SoS. This list is useful to evaluate process instances for architecting SoS.

SOAR Kernel. The third contribution of this work, described in Chapter 3, is SOAR Kernel. This kernel includes alphas, activity spaces, and competencies that describe what must be done in design process for acknowledged SoS software architectures. In development scenarios with mature, well-established organizational processes, it is possible to enhance them by verifying their alignment to SOAR Kernel. Furthermore, this kernel was also conceived to be a conceptual basis to build the SOAR practices. SOAR Kernel was evaluated in a survey conducted with experts in SoS and software architecture. Results indicate that SOAR Kernel is adequate to its purposes, being a relevant contribution to SoS community.

SOAR-A. Chapter 4 describes SOAR-A, a SOAR practice proposed to support instantiation of processes for architectural analysis of acknowledged SoS. Similarly to evaluation strategy adopted with SOAR Kernel, we also evaluated SOAR-A with a survey involving experts. Results presented positive evaluations, indicating that SOAR-A can be used to support process authoring on architectural analysis of acknowledged SoS.

SOAR-S. Chapter 5 describes SOAR-S, a SOAR practice conceived to support instantiation of processes for architectural synthesis of acknowledged SoS. Aiming at evaluating SOAR-S, we conducted two studies. The first one evaluated if process instances could be generated from SOAR-S. Results were positive, showing that is possible to use SOAR-S to generate process instances. In the second study, we conducted an experiment confronting process instances generated with SOAR against the ones generated in an ad hoc manner. As a comparison criteria, we asked experts to apply the aforementioned list of requirements to evaluate process instances generated with SOAR-E support against ad hoc ones. Results shown that process instances generated with SOAR-S are significantly better than ad hoc instances.

SOAR-E. Chapter 6 describes SOAR-E, a SOAR practice to support instantiation of processes for architectural evaluation of acknowledged SoS. Similarly to evaluation strategy adopted in SOAR Kernel and SOAR-A, SOAR-E was evaluated with a survey involving experts. Its evaluation was positive, indicating that SOAR-E can be used to support such instantiation.

It is undeniable that each SoS has an unique development context with exclusive demands for its architecting process. In this perspective, SOAR is a flexible approach to support the establishment of process instances that consider particularities of each project and, at the same time, are aligned with a basis that explores common challenges of designing acknowledged SoS software architectures. In summary, achievements of this thesis contribute to the areas of Software Architecture and SoS, as they advance the current state of the art on the architectural design processes of acknowledged SoS software architectures.

7.2 Limitations and Future Work

During the development of this thesis, we identified limitations and also new opportunities of research to bring new contributions to the areas of SoS and Software Architecture. We summarize them as follows:

Evaluate the use of SOAR in real-world SoS. Due to the size and complexity to create a process instance including all design activities, we individually evaluated SOAR main elements (i.e., SOAR Kernel, SOAR-A, SOAR-S, and SOAR-E) by using surveys and

experimental studies. This set of evaluations allow us to understand SOAR limitations, perform enhancements, and verify tendencies of SOAR acceptance in the SoS community. As a next step, we envision conduction of case studies in industry scenario. Based on such evaluation and use, we expect to disseminate SOAR, enhance it with users feedbacks, and make it a relevant resource to support process authoring for acknowledged SoS software architectures.

Extension of SOAR to cover other SoS categories. SOAR was conceived with focus on acknowledged SoS. In this sense, we also consider to enrich SOAR by proposing variations that encompass other SoS categories, i.e., virtual, collaborative, and directed. For example, we can consider an extension of SOAR for collaborative SoS, in which constituent systems voluntarily collaborate to form a SoS according to common interests. In this case, we must investigate how to change SOAR structure according to specific challenges of this category, such as the additional complexity of voluntary constituents, which can unexpectedly stop their collaboration in the SoS.

Specialization of SOAR for specific application domains. After SOAR acceptance by the SoS community, we envision the development of specialized versions focused on specific application domains, in which SoS is typically employed, e.g., traffic control, military defense, global Earth observation, flood monitoring, and smart cities. For this, we must explore and include common architectural solutions, each one presenting sets of design solutions more adequate for each application domain. For example, we can consider in these specializations: (i) consensual quality models providing an essential set of quality attributes, associated metrics, and guidelines for use; (ii) reference architectures by considering particularities of specific application domains; and (iii) product line architectures, for application domains that demand constituents produced in a product line perspective. Therefore, our focus at this point will be to explore and mature solutions for common problems in specific application domains, complementing SOAR with a set of compatible, reusable architectural solutions.

References

Abdalla, G.; Damasceno, C. D. N.; Guessi, M.; Oquendo, F.; Nakagawa, E. Y. A systematic literature review on knowledge representation approaches for systems-of-systems. In: *Proceedings of the 10th Brazilian Symposium on Components, Architectures and Reuse Software*, Belo Horizonte, Brazil, 2015, p. 70–79.

Acheson, P. Methodology for object-oriented system architecture development. In: *Proceedings of the 4th Annual IEEE Systems Conference (SysCon 2010)*, USA: IEEE, 2010, p. 643–646.

Acheson, P.; Pape, L.; Dagli, C.; Kilicay-Ergin, N.; Columbi, J.; Haris, K. Understanding system of systems development using an agent- based wave model. *Procedia Computer Science*, v. 12, p. 21–30, 2012.

Ackermann, C.; Lindvall, M.; Cleaveland, R. Towards behavioral reflexion models. In: *Proceedings of the 20th IEEE Int. Symposium on Software Reliability Engineering (ISSRE 2009)*, USA: IEEE, 2009.

Agusdinata, D. B.; DeLaurentis, D. Specification of system-of-systems for policymaking in the energy sector. *Integrated Assessment Journal*, v. 8, n. 2, 2008.

Aleti, A.; Buhnova, B.; Grunske, L.; Koziolek, A.; Meedeniya, I. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, v. 39, n. 5, p. 658–683, 2013.

Alwakeel, S. S.; Alhalabi, B.; Alwakeel, M. M. Perspectives on system of systems for pilgrimage ritual guidance and management. In: *Proceedings of the 8th Annual IEEE Systems Conference (SysCon 2014)*, USA: IEEE, 2014, p. 425–430.

America, P.; Rommes, E.; Obbink, H. Multi-view variation modeling for scenario analysis. *Software Product-Family Engineering*, v. 3014, p. 44–65, 2004.

Andrews, Z.; Payne, R.; Romanovsky, A.; Didier, A.; Mota, A. Model-based development of fault tolerant systems of systems. In: *Proceedings of the 7th Annual IEEE Systems Conference (SysCon 2013)*, USA: IEEE, 2013, p. 356–363.

Aoyama, M.; Tanabe, H. A design methodology for real-time distributed software architecture based on the behavioral properties and its application to advanced automotive software. In: *Proceedings of the 18th Asia Pacific Software Engineering Conference (APSEC)*, Ho Chi Minh, Vietname, 2011, p. 211–218.

Basili, V. R.; Shull, F.; Lanubile, F. Building knowledge through families of experiments. *IEEE Transaction on Software Engineering*, v. 25, n. 4, p. 456–473, 1999.

Bass, L.; Clements, P.; Kazman, R. *Software Architecture in practice*. SEI Series in Software Engineering, 3 ed. Addison-Wesley, 2012.

Bass, L.; Kazman, R. *Architecture-based development*. Technical Report CMU/SEI-99-TR-007, SEI, Pittsburgh, USA, 1999.

Batista, T. Challenges for SoS architecture description. In: *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems (SESoS 2013)*, New York, NY, USA: ACM, 2013, p. 35–37.

Belloir, N.; Chiprianov, V.; Ahmad, M.; Munier, M.; Gallon, L.; Bruel, J.-M. Using relax operators into an mde security requirement elicitation process for systems of systems. In: *Proceedings of the European Conference on Software Architecture Workshops*, Vienna, Austria, 2014, p. 32:1–32:4.

Bellomo, S.; Smith, J. D. Attributes of effective configuration management for systems of systems. In: *Proceedings of the 2nd Annual IEEE Systems Conference*, Piscataway, NJ, USA: IEEE, 2008, p. 1–8.

Bengtsson, P.; Lassing, N.; Bosch, J.; van Vliet, H. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, v. 69, n. 1-2, p. 129–147, 2004.

Bhasin, K.; Hayden, J. Architecting communication network of networks for space system of systems. In: *Proceedings of the IEEE International Conference on System of Systems Engineering*, 2008a.

Bhasin, K.; Hayden, J. Architecting communication network of networks for space system of systems. In: *Proceedings of the IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2008b, p. 1–7.

Bianchi, T.; Santos, D. S.; Felizardo, K. R. Quality attributes of systems-of-systems: A systematic literature review. In: *Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems*, Proceedings of the SESoS '15, Piscataway, NJ, USA: IEEE Press, 2015, p. 23–30 (*Proceedings of the SESoS '15*,). Disponível em `http://dl.acm.org/citation.cfm?id=2821418.2821425`

Biolchini, J.; Mian, P. G.; Natali, A. C. C.; Travassos, G. H. *Systematic review in Software Engineering.* Technical Report, Systems Engineering and Computer Science Department, Federal University of Rio de Janeiro, Brazil, 2005.

Boardman, J.; Sauser, B. System of systems – The meaning of *of*. In: *Proceedings of the IEEE/SMC International Conference on Systems of Systems Engineering*, Los Angeles, USA: IEEE Computer Society, 2006.

Bodeau, D. System-of-systems security engineering. In: *Proceedings of the 10th Annual Computer Security Applications Conference*, USA: IEEE, 1994, p. 228–235.

Boehm, B.; Lane, J. 21st century processes for acquiring 21st century software-intensive systems of systems. *Journal of Defense Software Engineering*, v. 19, n. 5, p. 4–9, 2006.

Bonilla, E.; Britton, J.; Gordon, M.; Scheldt, M.; Williams, R. Automated generation of integrated architectures and end-to-end network models. In: *Proceedings of the IEEE Aerospace Conference*, 2005, p. 1363–1369.

Bosch, J. *Design and use of software architectures: Adopting and evolving a product-line approach.* 1 ed. New York, NY, USA: Addison-Wesley, 2000.

Bowen, R.; Sahin, F. A net-centric xml based system of systems architecture for human tracking. In: *Proceedings of the 5th International Conference on System of Systems Engineering*, Loughborough, England, 2010, p. 1–6.

Bowen, R.; Sahin, F. Net-centric system of systems framework for human detection. In: *Proceedings of the 8th Conference on System of Systems Engineering*, USA: IEEE, 2013, p. 255–260.

Breivold, H. P.; Crnkovic, I.; Larsson, M. A systematic review of software architecture evolution research. *Information and Software Technology*, v. 54, n. 1, p. 16–40, 2012.

Brereton, P.; Kitchenham, B. A.; Budgen, D.; Turner, M.; Khalil, M. Lessons from applying the systematic literature review process within the Software Engineering domain. *Journal of Systems and Software*, v. 80, n. 4, p. 571–583, 2007.

Briggs, M.; Baird, D.; Ogg, W.; Aafloy, S.; El-Osery, A.; Wedeward, K. An adaptable outdoor robotic platform: architecture, communications, and control. In: *Proceedings of the IEEE/SMC International Conference on System of Systems Engineering*, USA: IEEE, 2006.

Brondum, J.; Liming, Z. Towards an architectural viewpoint for systems of software intensive systems. In: *Proceedings of the ICSE Workshop on Sharing and Reusing Architectural Knowledge*, Cape Town, South Africa, 2010, p. 60–63.

Bryans, J.; Payne, R.; Holt, J.; Perry, S. Semi-formal and formal interface specification for system of systems architecture. In: *Proceedings of the IEEE International Systems Conference (SysCon 2013)*, USA: IEEE, 2013, p. 612–619.

Bull, P.; Grigg, A.; Guan, L.; Phillips, I. A quality of service framework for adaptive and dependable large scale system-of-systems. In: *Proceedings of the 5th IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2010, p. 1–6.

Butterfield, M.; Pearlman, J.; Vickroy, S. A system-of-systems engineering geoss: Architectural approach. *Systems Journal, IEEE*, v. 2, n. 3, p. 321–332, 2008.

Caffall, D.; Michael, J. Architectural framework for a system-of-systems. In: *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, USA: IEEE, 2005, p. 1876–1881.

Calinescu, R. Resource-definition policies for autonomic computing. In: *Proceedings of the 5th Conference on Autonomic and Autonomous Systems*, USA: IEEE, 2009, p. 111–116.

Calinescu, R.; Kwiatkowska, M. Software Engineering techniques for the development of systems of systems. In: Choppy, C.; Sokolsky, O., eds. *Proceedings of the 15th Monterey Workshop Foundations of Computer Software: Future Trends and Techniques for Development*, v. 6028 de *Lecture Notes in Computer Science*, Germany: Springer Berlin Heidelberg, p. 59–82, 2010.

Carbon, R.; Johann, G.; Muthig, D.; Naab, M. A method for collaborative development of systems of systems in the office domain. In: *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008)*, USA: IEEE, 2008, p. 339–345.

Chen, H.-M.; Kazman, R. Architecting ultra-large-scale green information systems. In: *Proceedings of the 1st International Workshop on Green and Sustainable Software*, Piscataway, NJ, USA, 2012, p. 69–75.

Chen, L.; Ali Babar, M.; Nuseibeh, B. Characterizing architecturally significant requirements. *IEEE Software*, v. 30, n. 2, p. 38–45, 2013.

Chen, L.; Babar, M. A.; Zhang, H. Towards an evidence-based understanding of electronic data sources. In: *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering*, Swinton, United Kingdom: British Computer Society, 2010, p. 135–138.

Chen, P. Architecture-based interoperability evaluation in evolutions of networked enterprises. In: Bussler, C.; Haller, A., eds. *Proceedings of the Business Process Management Workshops*, v. 3812 de *Lecture Notes in Computer Science*, Germany: Springer Berlin Heidelberg, p. 293–304, 2006.

Chen, P.; Han, J. Facilitating system-of-systems evolution with architecture support. In: *Proceedings of the 4th International Workshop on Principles of Software Evolution*, New York, NY, USA: ACM, 2001, p. 130–133.

Chigani, A.; Balci, O. The process of architecting for software/system engineering. *International Journal of System of Systems Engineering*, v. 3, n. 1, p. 1–23, 2012.

Christian, E. GEOSS architecture principles and the GEOSS clearinghouse. *IEEE Systems Journal*, v. 2, n. 3, p. 333–337, 2008.

Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. *Documenting software architectures: Views and beyond.* SEI Series in Software Engineering, 2 ed. Addison-Wesley Professional, 2010.

Clements, P.; Kazman, R.; Klein, M. *Evaluating software architectures: Methods and case studies.* The SEI Series in Software Engineering. Boston, MA: Addison-Wesley, 2002.

Cloutier, R.; Griego, R. Applying object oriented systems engineering to complex systems. In: *Proceedings of the 2nd Annual IEEE Systems Conference*, USA: IEEE, 2008, p. 1–6.

Cook, T.; Drusinksy, D.; Shing, M.-T. Specification, validation and run-time monitoring of SOA based system-of-systems temporal behaviors. In: *Proceedings of the 2nd IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2007, p. 1–6.

Cordis Directions in systems of systems engineering. [*On-line*], *World Wide Web*, available in: `http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/report_system_of_system.pdf` (Access in 03/18/2016), 2012.

Dagli, C.; Ergin, N.; Enke, D. Gosavi, A.; Qin, R.; Colombi, J.; Rebovich, R. Giammarco, K.; Acheson, P.; Haris, K.; Pape, L. *An advanced computational approach to system of systems analysis & architecting using agent-based behavioral model.* Technical Report 021-2, Systems Engineering Research Center(SERC), 2013.

Dahmann, J.; Baldwin, K. Understanding the current state of us defense systems of systems and the implications for systems engineering. In: *Proceedings of the 2nd Annual IEEE Systems Conference*, Montreal, Canada, 2008, p. 1–7.

Dahmann, J.; Rebovich, G.; Lowry, R.; Lane, J.; Baldwin, K. An implementers' view of systems engineering for systems of systems. In: *Proceedings of the IEEE International Systems Conference*, Montreal, QC, 2011, p. 212–217.

Dandashi, F.; Hause, M. C. Uaf for system of systems modeling. In: *Proceedings of the 10th System of Systems Engineering Conference*, 2015, p. 199–204.

Degrossi, L. C.; Amaral, G. G.; Vasconcelos, E. S. M.; Albuquerque, J. P.; Ueyama, J. Using wireless sensor networks in the sensor web for flood monitoring. In: *Proceedings of the 10th International ISCRAM Conference*, 2013.

DeLaurentis, D. Understanding transportation as a system of systems design problem. In: *Proceedings of the 43rd AIAA Aerospace Sciences Meeting*, Nevada, USA, 2005a.

DeLaurentis, D. A. A taxonomy-based perspective for systems of systems design methods. In: *Proceedings of the IEEE international conference on systems, man and cybernetics*, IEEE, 2005b, p. 86–91.

DeLaurentis, D. A. Appropriate modeling and analysis for systems of systems: Case study synopses using a taxonomy. In: *Proceedings of the IEEE International Conference on System of Systems Engineering*, 2008, p. 1–6.

Demchak, B.; Ermagan, V.; Farcas, E.; Huang, T.; Kruger, I.; Menarini, M. A rich services approach to CoCoME. In: Rausch, A.; Reussner, R.; Mirandola, R.; Plášil, F., eds. *The Common Component Modeling Example: Comparing software component models*, v. 5153 de *Lecture Notes in Computer Science*, Germany: Springer Berlin Heidelberg, p. 85–115, 2008.

Dickerson, C.; Valerdi, R. Using relational model transformations to reduce complexity in SoS requirements traceability: Preliminary investigation. In: *Proceedings of the 5th IEEE International Conference on System of Systems Engineering (SoSE 2010)*, USA: IEEE, 2010, p. 1–6.

Dieste, O.; GrimÃ¡n, A.; Juristo, N. Developing search strategies for detecting relevant experiments. In: *Empirical Software Engineering*, v. 14, p. 513–539, 2009.

Dikel, D. M.; Kane, D.; Wilson, J. R. *Software architecture: Organizational principles and patterns*. 1 ed. Prentice Hall, 2001.

DoD The dodaf architecture framework version 2.02. Available at: `http://dodcio.defense.gov/` (Access in 06/03/2016), 2010.

DoD, D. *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, 2008.

Domerçant, J.; Mavris, D. Measuring the architectural complexity of military systems-of-systems. In: *Proceedings of the IEEE Aerospace Conference*, USA: IEEE, 2011, p. 1–16.

Dvorak, D.; Indictor, M.; Ingham, M.; Rasmussen, R.D.and Stringfellow, M. A unifying framework for systems modeling, control systems design,and system operation. In: *Proceedings of the 4th IEEE International Conference on Systems, Man and Cybernetics*, 2005, p. 3648–3653.

Dybå, T.; Dingsøyr, T.; Hanssen, G. K. Applying systematic reviews to diverse study types: An experience report. In: *Proceedings of the 1st International Symposium on Empirical Software Engineering and Maintenance (ESEM 2007)*, USA: IEEE, 2007, p. 225–234.

Eaton, R.; Katupitiya, J.; Siew, K.; Dang, K. Precision guidance of agricultural tractors for autonomous farming. In: *Proceedings of the 2nd Annual IEEE Systems Conference*, USA: IEEE, 2008, p. 1–8.

Eeles, P. Understanding architectural assets. In: *Proceedings of the 7th Working IEEE/I-FIP Conference on Software Architecture (WICSA'08)*, Vancouver, Canada, 2008, p. 267–270.

Ermagan, V.; Kruger, I.; Menarini, M. Aspect-oriented modeling approach to define routing in enterprise service bus architectures. In: *Proceedings of the International*

Workshop on Models in Software Engineering, New York, NY, USA: ACM, 2008, p. 15–20.

Farcas, C.; Farcas, E.; Krueger, I.; Menarini, M. Addressing the integration challenge for avionics and automotive systems: From components to rich services. *Proceedings of the IEEE*, v. 98, n. 4, p. 562–583, 2010.

Farroha, D.; Farroha, B. Agile development for system of systems: Cyber security integration into information repositories architecture. In: *Proceedings of the 5th Annual IEEE Systems Conference*, USA: IEEE, 2011, p. 182–188.

Ferguson, R.; Peterson, B.; Thompson, H. System software framework for system of systems avionics. In: *Proceedings of the 24th Digital Avionics Systems Conference (DASC 2005)*, USA: IEEE, 2005, p. 1–10.

Firesmith, D. *Profiling systems using the defining characterstics of systems of systems (SoS)*. Technical Report CMU/SEI-2010-TN-001, Software Engineering Institute (SEI), Carnegie Mellon University, 2010.

Gagliardi, M.; Wood, W. G.; Klein, J.; Morley, J. A uniform approach for system of systems architecture evaluation. *CrossTalk - The Journal of Defense Software Engineering*, v. 22, p. 12–15, 2009.

Gamble, M.; Gamble, F. Reasoning about hybrid system of systems designs. In: *Proceedings of the 7th International Conference on Composition-Based Software Systems (ICCBSS 2008)*, USA: IEEE, 2008, p. 154–163.

Garland, J.; Anthony, R. *Large-scale software architecture: A practical guide using uml*. West Sussex, England: John Wiley & Sons Ltd., 2003.

Ge, B.; Hipel, K. W.; Yang, K.; Chen, Y. A data-centric capability-focused approach for system-of-systems architecture modeling and analysis. *Systems Engineering Journal*, v. 16, n. 3, p. 363–377, 2013.

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

Gonçalves, M. B.; Oquendo, F.; Nakagawa, E. Y. A meta-process to construct SoS software architectures. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, USA: ACM, 2015, p. 1411–1416.

Gorlick, M. M.; Strasser, K.; Taylor, R. N. COAST: An architectural style for decentralized on-demand tailored services. In: *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, USA: IEEE, 2012, p. 71–80.

Greaves, M.; Stavridou-Coleman, V.; Laddaga, R. Dependable agent systems. *IEEE Intelligent Systems*, v. 19, n. 5, p. 20–23, 2004.

Griendling, K.; Mavris, D. N. A process for systems-of-systems architecting. In: *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Reston, VA, USA: American Institute of Aeronautics and Astronautics, 2010.

Guessi, M.; Neto, V. V. G.; Bianchi, T.; Felizardo, K. R.; Oquendo, F.; Nakagawa, E. Y. A Systematic Literature Review on the Description of Software Architectures for Systems of Systems. In: *Proc. of the Symposium on Applied Computing ACM (SAC 2015)*, Salamanca, Spain, (acccepted), 2015, p. 1–8.

Haley, C.; Nuseibeh, B. Bridging requirements and architecture for systems of systems. In: *Proceedings of the International Symposium on Information Technology*, USA: IEEE, 2008, p. 1–8.

Hata, Y.; Kamozaki, Y.; Sawayama, T.; Taniguchi, K.; Nakajima, H. A heart pulse monitoring system by air pressure and ultrasonic sensor systems. In: *Proceedings of the 1st IEEE International Conference on System of Systems Engineering*, San Antonio, TX, USA, 2007.

Hershey, P.; Rao, S.; Silio, C.; Narayan, A. System of systems to provide Quality of Service monitoring, management and response in cloud computing environments. In: *Proceedings of the 7th IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2012, p. 314–320.

Hodges, R.; Cloutier, R.; Bone, M.; Korfiatis, P. Singleton to sandwich chunking into buslets for better system development. In: *Proceedings of the 6th IEEE International Conference on System of Systems Engineering (SoSE 2011)*, USA: IEEE, 2011, p. 125–130.

Hofmeister, C.; Kruchten, P.; Nord, R. L.; Obbink, H.; Ran, A.; America, P. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, v. 80, n. 1, p. 106–126, 2007.

Hofmeister, C.; Nord, R.; Soni, D. *Applied software architecture.* Addison-Wesley Professional, 2000.

Holl, G.; Grunbacher, P.; Elsner, C.; Klambauer, T. Supporting awareness during collaborative and distributed configuration of multi product lines. In: *Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, USA: IEEE, 2012, p. 137–147.

Holland, O. T. Taxonomy for the modeling and simulation of emergent behavior systems. In: *Proceedings of the Spring Simulation Multiconference*, Society for Computer Simulation International, 2007, p. 28–35.

Horita, F. E. A.; Fava, M. C.; Mendiondo, E. M.; Rotava, J.; Souza, V. C.; Ueyama, J.; Albuquerque, J. P. Agora-geodash: A geosensor dashboard for real-time flood risk monitoring , university park, usa,. In: *11th International ISCRAM Conference*, 2014.

Hughes, D.; Ueyama, J.; Mendiondo, E.; Matthys, N.; Horré, W.; Michaels, S.; Huygens, C.; Joosen, W.; Man, K. L.; Guan, S.-U. A middleware platform to support river monitoring using wireless sensor networks. *Journal of the Brazilian Computer Society*, v. 17, p. 85–102, 2011.

Iacobucci, J.; Mavris, D. A method for the generation and evaluation of architecture alternatives on the cloud. In: *Proceedings of the 6th International Conference on System of Systems Engineering*, USA, 2011, p. 137–142.

IEEE Computer Society Guide to the Software Engineering Body of Knowledge (SWEBOK Version 3). Online, `http://www.swebok.org` - Accessed in January 4th 2015, 2014.

ISO/IEC *Unified Modeling Language (UML) Version 1.4.2(ISO/IEC 19501:2005(E)).* Standard 19501/2005, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC), 2005.

ISO/IEC *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* Technical Report 25010/2011, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC), 2011.

ISO/IEC/IEEE *Recommended Practice for Architectural Description of Software-Intensive Systems (ISO/IEC/IEEE 42010).* Standard 42010/2011, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE), 2011.

Jackson, M. M.; FernÃ¡ndez, M. M.; McVittie, T. I.; Sindiy, O. V. *Architecting the human space flight program with systems modeling language (sysml).* Technical Report, American Institute of Aeronautics and Astronautics, online,`http://hdl.handle.net/2014/42571r` - Accessed in July 4th 2016, 2012.

Jacobson, I. Software Development Moves from a Craft to an Engineering Discipline Using the Essence Standard. Online,`https://www.ivarjacobson.com/publications/case-studies/asian-telecommunications-equipment-vendor` - Accessed in July 4th 2016, 2015.

Jacobson, I.; Ng, P.-W.; McMahon, P. E.; Spence, I.; Lidman, S. *The essence of software engineering: Applying the semat kernel.* Addison-Wesley, 2013.

Jamshidi, M. System of systems engineering - new challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine*, v. 23, n. 5, p. 4–19, 2008a.

Jamshidi, M. *System of systems engineering: Innovations for the twenty-first century.* 1 ed. Wiley & Sons, 2008b.

Johnson, M. A. *System of systems engineering: Innovations for the twenty-first century*, cap. 18 - System-of-systems Standards. In: (Jamshidi, 2008b), p. 451–461, 2008.

Jones-Wyatt, E.; Domercant, J.; Mavris, D. A reliability-based measurement of interoperability for systems of systems. In: *Proceedings of the 7th Annual IEEE Systems Conference (SysCon 2013)*, USA: IEEE, 2013, p. 408–413.

Josuttis, N. *Soa in practice.* 1st ed. O'Reilly, 2007.

Kaiser, G.; Parekh, J.; Gross, P.; Valetto, G. Kinesthetics eXtreme: An external infrastructure for monitoring distributed legacy systems. In: *Proceedings of the Autonomic Computing Workshop*, USA: IEEE, 2003, p. 22–30.

Kasunic, M. *Designing an effective survey.* Technical Report CMU/SEI-2005-HB-004, 2005.

Kazman, R.; Gagliardi, M.; Wood, W. Scaling up software architecture analysis. *Journal of Systems and Software*, v. 85, n. 7, p. 1511–1519, 2012.

Kazman, R.; Klein, M.; Barbacci, M.; Longstaff, T.; Lipson, H.; Carriere, J. The architecture tradeoff analysis method. In: *Proceedings of the 4th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98)*, Monterey, CA, USA, 1998, p. 68–78.

Kewley Jr., R.; Andreas, T. A systems engineering process for development of federated simulations. In: *Proceedings of the Spring Simulation Multiconference*, San Diego, CA, USA: Society for Computer Simulation Int., 2009, p. 1–8.

Khalsa, S.; Nativi, S.; Geller, G. The geoss interoperability process pilot project (ip3). *IEEE Transactions on Geoscience and Remote Sensing*, v. 47, n. 1, p. 80–91, 2009.

Kitchenham, B. *Procedures for performing systematic reviews.* Technical Report, Keele University and NICTA, 2004.

Kitchenham, B.; Brereton, O. P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in Software Engineering: A systematic literature review. *Information and Software Technology*, v. 51, n. 1, p. 7–15, 2009.

Kitchenham, B.; Charters, S. *Guidelines for performing systematic literature reviews in Software Engineering.* Technical Report EBSE 2007-001, Keele University / Durham University, 2007.

Klein, J.; van Vliet, H. A systematic review of system-of-systems architecture research. In: *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, Vancouver, Canada, 2013, p. 13–22.

Kruchten, P. The 4+ 1 view model of architecture. *Software, IEEE*, v. 12, n. 6, p. 42–50, 1995.

Kruchten, P. *The rational unified process: An introduction.* The Addison-Wesley Object Technology Series, 3 ed. Addison-Wesley, 2003.

Kruchten, P.; Obbink, H.; Stafford, J. The past, present, and future for software architecture. *IEEE Softw.*, v. 23, n. 2, p. 22–30, 2006.

Kruger, I.; Meisinger, M.; Menarini, M.; Pasco, S. Rapid systems of systems integration - Combining an architecture-centric approach with enterprise service bus infrastructure. In: *Proceedings of the IEEE International Conference on Information Reuse and Integration*, USA: IEEE, 2006, p. 51–56.

Lane, J.; Valerdi, R. Synthesizing sos concepts for use in cost estimation. In: *Proceedings of the 1st IEEE International Conference o Systems, Man and Cybernetics*, 2005, p. 993–998.

Lane, J.; Valerdi, R. Synthesizing sos concepts for use in cost modeling. *Systems Engengineering*, v. 10, n. 4, p. 297–308, 2007.

Lewis, G.; Morris, E.; Simanta, S.; Smith, D. Service orientation and systems of systems. *IEEE Software*, v. 28, n. 1, p. 58–63, 2011.

Liang, S. X.; Luqi, V. B. Quantifiable architecting of dependable systems of embedded systems. *SIGSOFT Software Engineering Notes*, v. 28, n. 6, p. 7–7, 2003.

Lindvall, M.; Ackermann, C.; Stratton, W.; Sibol, D.; Ray, A.; Yonkwa, L.; Kresser, J.; Godfrey, S.; Knodel, J. Using sequence diagrams to detect communication problems between systems. In: *Proceedings of the IEEE Aerospace Conference (AERO 2008)*, USA: IEEE, 2008, p. 1–11.

Loiret, F.; Rouvoy, R.; Seinturier, L.; Merle, P. Software engineering of component-based systems-of-systems: A reference framework. In: *Proceedings of the 14th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2011)*, New York, NY, USA: ACM, 2011, p. 61–66.

Lytra, I.; Zdun, U. Supporting architectural decision making for systems-of-systems design under uncertainty. In: *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems*, New York, NY, USA: ACM, 2013, p. 43–46.

Maier, M. W. Architecting principles for systems-of-systems. *Systems Engineering*, v. 1, n. 4, p. 267–284, 1998.

McDonough, A. Munich Re and ESSENCE â€“ Kernel and Language for Software Engineering Methods: A Case Study . Online,`http://www.omg.org/news/whitepapers/` - Accessed in July 4th 2016, 2014.

Menon, C.; Kelly, T. Eliciting software safety requirements in complex systems. In: *Proceedings of the 4th Annual IEEE Systems Conference*, USA: IEEE, 2010, p. 616–621.

Mensing, B.; Goltz, U.; Aniculfesei, A.; Herold, S.; Gärtner, S.; Schneider, K. Towards integrated rule-driven software development for IT ecosystems. In: *Proceedings of the 6th IEEE International Conference on Digital Ecosystems Technologies*, USA: IEEE, 2012, p. 1–6.

Michael, J.; Riehle, R.; Shing, M.-T. The verification and validation of software architecture for systems of systems. In: *Proceedings of the IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2009, p. 1–6.

Mittal, S.; Risco Martin, J. Model-driven systems engineering for netcentric system of systems with DEVS unified process. In: *Proceedings of the Winter Simulation Conference*, USA: IEEE, 2013, p. 1140–1151.

Naegle, B. Developing performance-based requirements for open architecture design. In: *Proceedings of the 2nd IEEE International Conference on System of Systems Engineering (SoSE 2007)*, USA: IEEE, 2007, p. 1–6.

Nakagawa, E.; Oquendo, F. Perspectives and challenges of reference architectures in multi software product line. In: *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, New York, NY, USA: ACM, 2013, p. 100–103.

Nakagawa, E. Y.; GonÃ§alves, M. B.; Guessi, M.; Oliveira, L.; Oquendo, F. The state of the art and future perspectives in systems of systems software architectures. In: *Proceedings of the 1st Software Engineering System of Systems Workshop*, 2013.

Nativi, S.; Bigagli, L.; Mazzetti, P.; Boldrini, E.; Papeschi, F. GI-Cat: A mediation solution for building a clearinghouse catalog service. In: *Proceedings of the International Advanced Geographic Information Systems & Web Services*, USA: IEEE, 2009, p. 68–74.

Nguyen, Q. T.; Bouju, A.; Estraillier, P. Multi-agent architecture with space-time components for the simulation of urban transportation systems. *Procedia - Social and Behavioral Sciences*, v. 54, p. 365–374, 2012.

Nielsen, C. B.; Larsen, P. G. Extending VDM-RT to enable the formal modelling of system of systems. In: *Proceedings of the 7th International Conference on System of Systems Engineering*, 2012, p. 457–462.

Nielsen, C. B.; Larsen, P. G.; Fitzgerald, J.; Woodcock, J.; Peleska, J. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Computing Surveys*, v. 48, n. 2, p. 18:1–18:41, 2015.

Obbink, H.; Müller, J. K.; America, P.; van Ommering, R.; Muller, G.; van der Sterren, W.; Wijnstra, J. G. Copa: a component-oriented platform architecting method for families of software-intensive electronic products (tutorial). In: *Proceedings of 1st Software Product Line Conference*, Denver, CO, USA, 2000.

Object Management Group (OMG) Essence - kernel and language for software engineering methods (essence) v 1.1. Online, available in: `http://www.omg.org/spec/Essence/1.0` Accessed in February 8th 2015, 2014.

Object Management Group (OMG) Software and Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Online, `http://www.omg.org/spec/Essence/1.1/` - Accessed in July 8th 2015, 2015a.

Object Management Group (OMG) Unified Modeling Language (UML). Online, `http://www.omg.org/spec/UML/2.4.1/` - Accessed in July 9th 2015, 2015b.

Oliveira, L. B. R.; Nakagawa, E. Y. A service-oriented reference architecture for software testing domain. In: Crnkovic, I.; Gruhn, V.; Book, M., eds. *Proceedings of the 5th European Conference on Software Architecture*, v. 6903 de *Lecture Notes in Computer Science*, Germany: Springer Berlin Heidelberg, p. 405–421, 2011.

Oliveira, M.; Pereira, J. Extensible virtual environment systems using system of systems engineering approach. In: *Proceedings of the 17th International Conference on Artificial Reality and Telexistence (ICAT 2007)*, USA: IEEE, 2007, p. 89–96.

Oquendo, F. Formally describing the software architecture of systems-of-systems with sosadl. In: *11th System of Systems Engineering Conference, SoSE 2016, Kongsberg, Norway, June 12-16, 2016*, 2016, p. 1–6.
Disponível em `http://dx.doi.org/10.1109/SYSOSE.2016.7542926`

Papatheocharous, E.; Axelsson, J.; Andrersson, J. Issues and challenges in ecosystems for federated embedded systems. In: *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems*, New York, NY, USA: ACM, 2013, p. 21–24.

Papazoglou, M.; Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, v. 16, n. 3, p. 389–415, 2007.

Parker, J. M. Applying a system of systems approach for improved transportation. *S.A.P.I.EN.S [Online]*, online,`http://sapiens.revues.org/1011` - Accessed in July 4th 2016, 2008.

Payne, R.; Bryans, J.; Fitzgerald, J.; Riddle, S. Interface specification for system-of-systems architectures. In: *Proceedings of the 7th International Conference on System of Systems Engineering*, Genoa, Italy, 2012, p. 567–572.

Pérez, J.; Díaz, J.; Garbajosa, J.; Yagüe, A.; Gonzalez, E.; Lopez-Perea, M. Large-scale smart grids as system of systems. In: *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems*, Montpellier, France, 2013, p. 38–42.

Perrochon, L.; Mann, W. Inferred designs. *IEEE Software*, v. 16, n. 5, p. 46–51, 1999.

Perry, D. Issues in process architecture. In: *Proceedings of the 9th Software Process Workshop*, Arlie, VA, USA, 1994, p. 138–140.

Petcu, V.; Petrescu, A. Systems of systems applications for telemedicine. In: *Proceedings of the 9th Roedunet International Conference*, Sibiu, Romania, 2010, p. 208–211.

Petersen, K.; Feldt, R.; Shahid, M.; Mattsson, M. Systematic mapping studies in Software Engineering. In: *Proceedings of the 12th Conference on Evaluation and Assessment in Software Engineering (EASE 2008)*, Swinton, England, UK: British Computer Society, 2008, p. 1–10.

Ploom, T.; Glaser, A.; Scheit, S. Platform based approach for automation of workflows in a system of systems. In: *Proceedings of the 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, USA: IEEE, 2013, p. 12–21.

Pressman, R. S.; Maxim, B. R. *Software engineering: A practitioner's approach.* 8th ed. McGraw-Hill Higher Education, 2015.

Ramos, M.; Masiero, P.; Braga, R.; Penteado, R. Reengineering legacy systems towards system of systems development. In: *Proceedings of the 13th IEEE International Conference on Information Reuse and Integration (IRI 2012)*, USA: IEEE, 2012, p. 624–630.

Ran, A. *Ares conceptual framework for software architecture.* Technical Report, Nokia Research Center, 2000.

Rossak, W.; Zemel, T.; Kirova, V.; Jololian, L. Two-level process model for integrated system development. In: *Proceedings of the Tutorial and Workshop on Systems Engineering of Computer-Based Systems*, USA, 1994, p. 90–96.

Rothenhaus, K.; Michael, J.; Shing, M.-T. Architectural patterns and auto-fusion process for automated multisensor fusion in SOA system-of-systems. *IEEE Systems Journal*, v. 3, n. 3, p. 304–316, 2009.

Sage, A.; Biemer, S. Processes for system family architecting, design, and integration. *IEEE Systems Journal*, v. 1, n. 1, p. 5–16, 2007.

Sahin, F.; Jamshidi, M.; Sridhar, P. A discrete event XML based simulation framework for system of systems architectures. In: *Proceedings of the 2nd IEEE International Conference on System of Systems Engineering*, USA: IEEE, 2007, p. 1–7.

Sanduka, I.; Obermaisser, R. Model-based development of systems-of-systems with real-time requirements. In: *Proceedings of the 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, p. 188–194.

Sauser, B.; Boardman, J.; Verma, D. Systomics: Toward a biology of system of systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, v. 40, n. 4, p. 803–814, 2010.

Schaefer, R. Systems of systems and coordinated atomic actions. *SIGSOFT Software Engineering Notes*, v. 30, n. 1, p. 6–11, 2005.

Schroh, D.; Bozowsky, N.; Savigny, M.; Wright, W. nCompass service oriented architecture for tacit collaboration services. In: *Proceedings of the 13th Information Visualisation Int. Conference (IV 2009)*, USA: IEEE, 2009, p. 433–442.

Schuitemaker, K.; Braakhuis, J. G.; Rajabalinejad, M. A model based safety architecture framework for dutch high speed train lines. In: *Proceedings of the 10th System of Systems Engineering Conference*, 2015, p. 24–29.

SEI Software Engineering Institute (SEI) – Software Architecture Glossary. Online, `http://www.sei.cmu.edu/architecture/start/glossary/` - Accessed in January 4th 2015, 2015.

Selberg, S.; Austin, M. Toward an evolutionary system of systems architecture. In: *Proceedings of the 18th Annual International Symposium of the Int. Council on Systems Engineering*, 2008, p. 1–14.

Shanmugapriya, P.; Suresh, R. M. Software architecture evaluation methods - a survey. *International Journal of Computer Applications*, v. 49, n. 16, p. 19–26, 2012.

Sharawi, A.; Sala-Diakanda, S. N.; Dalton, A.; Quijada, S.; Yousef, N.; Rabelo, L.; Sepulveda, J. A distributed simulation approach for modeling and analyzing systems of systems. In: *Proceedings of the Winter Simulation Conference*, USA, 2006, p. 1028–1035.

Shaw, M.; Clements, P. The golden age of software architecture. *IEEE Software*, v. 23, n. 2, p. 31–39, 2006.

Shing, M.-T.; Drusinsky, D.; Cook, T. Quality assurance of the timing properties of real-time, reactive system-of-systems. In: *Proceedings of the 1st IEEE International Conference on System of Systems Engineering*, 2006, p. 1–6.

Shull, F.; Singer, J.; Sjøberg, D. *Guide to advanced empirical software engineering.* 1 ed. Springer-Verlag London, 2008.

Silva, E.; Batista, T.; Oquendo, F. A mission-oriented approach for designing system-of-systems. In: *Proceedings of the 10th System of Systems Engineering Conference*, USA, 2015, p. 346–351.

Simanta, S.; Morris, E.; Lewi, G. A.; Smith, D. B. Engineering lessons for systems of systems learned from service-oriented systems. In: *Proceedings of the 4th Annual IEEE Systems Conference (SysCon 2010)*, USA: IEEE, 2010, p. 634–639.

Sloane, E.; Beck, R.; Metzger, S. AGSOA - Agile governance for service oriented architecture (SOA) systems: A methodology to deliver 21st Century military net-centric systems of systems. In: *Proceedings of the 2nd Annual IEEE Systems Conference*, USA: IEEE, 2008, p. 1–4.

Sloane, E.; Way, T.; Gehlot, V.; Beck, R. Conceptual SOS model and simulation systems for a next generation national healthcare information network (NHIN-2): Creating a net-centric, extensible, context aware, dynamic discovery framework for robust, secure, flexible, safe, and reliable healthcare. In: *Proceedings of the 1st Annual IEEE Systems Conference*, USA: IEEE, 2007, p. 1–6.

Sommerville, I. *Software engineering*. 9th ed. Addison-Wesley Longman Publishing Co., Inc., 2009.

Squair, M. Safety, software architecture and MIL-STD-1760. In: *Proceedings of the 11th Australian Workshop on Safety Critical Systems and Software*, Darlinghurst, Australia: Australian Computer Society, Inc., 2006, p. 93–112.

Stratton, W.; Sibol, D.; Lindvall, M.; Ackermann, C.; Godfrey, S. Developing an approach for analyzing and verifying system communication. In: *Proceedings of the IEEE Aerospace Conference*, USA: IEEE, 2009, p. 1–13.

SysML Partners Unified Modeling Language (UML). Online, `http://www.omgsysml.org/` - Accessed in July 7th 2016, 2015.

Tianfield, H. Fundamentals and architectures of complex distributed systems. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, USA: IEEE, 2008, p. 2471–2475.

Tsai, W.; Fan, C.; Chen, Y.; Paul, R. DDSOS: A dynamic distributed service-oriented simulation framework. In: *Proceedings of the 39th Annual Simulation Symposium (ANSS 2006)*, 2006, p. 1–8.

Ueyama, J.; Hughes, D. R.; Matthys, N.; Horré, W.; Joosen, W.; Huygens, C.; Michaels, S. An event-based component model for wireless sensor networks: A case study for river monitoring. In: *Proceedings of the 28th Brazilian Symposium on Computer Networks and Distributed Systems*, Porto Alegre, RS, Brazil: SBC, 2010, p. 997–1004.

Valerdi, R.; Ross, A.; Rhodes, D. A. A framework for evolving system of systems engineering. *The Journal of Defense Software Engineering*, p. 28–30, available at: `http://www.crosstalkonline.org/storage/issue-archives/2007/200710/200710-Valerdi.pdf` (Access in 07/01/2016), 2007.

Vila, V. Data fusion enabled networks. In: *Proceedings of the 10th International Conference on Information Fusion*, USA: IEEE, 2007, p. 1–7.

Wang, R.; Dagli, C. Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model-driven systems development process. *Systems Engineering*, v. 14, n. 4, p. 383–409, 2011.

Wessel, J.; Meyer, B. Assessing system software performance in complex system of systems environments. In: *Proceedings of the Military Communications Conference*, USA: IEEE, 2010, p. 2310–2315.

Wester-Ebbinghaus, M.; Moldt, D.; Kohler-Bubmeier, M. From multi-agent to multi-organization systems: Utilizing middleware approaches. In: Artikis, A.; Picard, G.; Vercouter, L., eds. *Proceedings of the 9th International Workshop on Engineering Societies in the Agents World*, v. 5485 de *Lecture Notes in Computer Science*, Germany: Springer Berlin Heidelberg, p. 46–65, 2009.

Wohlin, C. Guidelines for snowballing in systematic literature studies and a replication in Software Engineering. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA, 2014.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering*. Berlin, Germany: Springer, 2012.

Xia, X.; Wu, J.; Liu, C.; Xu, L. A model-driven approach for evaluating system of systems. In: *Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*, USA: IEEE, 2013, p. 56–64.

You-Sheng, Z.; Yu-Yun, H. Architecture-based software process model. *ACM SIGSOFT Software Engineering Notes*, v. 28, n. 2, p. 1–5, 2003.

Zadeh, L. Fuzzy sets. *Information and Control*, v. 8, n. 3, p. 338–353, 1965.

Zhang, H.; Babar, M. A.; Tell, P. Identifying relevant studies in Software Engineering. *Information and Software Technology*, v. 53, n. 6, p. 625–637, 2011.

Zhou, B.; Dvoryanchikova, A.; Lobov, A.; Lastra, J. Modeling system of systems: A generic method based on system characteristics and interface. In: *Proceedings of the 9th IEEE International Conference on Industrial Informatics*, Caparica, Lisbon, Portugal, 2011, p. 361–368.

Zhou, J.; De Roure, D. Floodnet: Coupling adaptive sampling with energy aware routing in a flood warning system. *Journal of Computer Science and Technology*, v. 22, n. 1, p. 121–130, 2007.

Zhou, Y.; Zhang, H.; Huang, X.; Yang, S.; Babar, M. A.; Tang, H. Quality assessment of systematic reviews in Software Engineering: A tertiary study. In: *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA: ACM, 2015.

Zhu, L.; Staples, M.; Jeffery, R. Scaling up software architecture evaluation processes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 5007 LNCS, p. 112–122, 2008.

Appendix

A

Systematic Mapping on SoS Software Architectures: Study Protocol and List of Included Primary Studies

This appendix presents the research protocol and the list of included studies of the SM whose results are summarized and discussed in Section 2.2.

An SM provides mechanisms to identify and aggregate research evidence (Petersen et al., 2008). It is a more open form of SLR providing an overview of a research area to assess the amount of existent evidence on a topic of interest (Petersen et al., 2008). An individual piece of evidence, for instance, a case study or an empirical study considered in SLRs and SMs, is known as primary study, while SLRs and SMs are known as secondary studies (Kitchenham and Charters, 2007). In other words, the term "primary study" refers to an individual publication or a study that aggregates research evidence.

SM builds a classification scheme and organizes a research field of interest within the categories of the scheme. As a result, the coverage of the research field can be determined and the scheme can also be used to answer specific research questions (Petersen et al., 2008). Therefore, SM provides an overview of a research area by identifying and quantifying the related available research. In general, an SM begins with a planning phase, which includes formulation of research questions and definition of inclusion and exclusion criteria. The data extraction activity for an SM is broad and the analysis of a mapping

does not include the use of in-depth analysis techniques, such as meta-analysis, but rather totals and summaries. Graphic representations also can be used to summarize the data (Kitchenham and Charters, 2007).

Secondary studies (SLRs and SMs) (Kitchenham and Charters, 2007; Petersen et al., 2008) have been provided with methodological and structured processes to identify and aggregate research evidence. They have been increasingly applied and advocated as a suitable research strategy in the Software Engineering area (Brereton et al., 2007; Kitchenham and Charters, 2007; Petersen et al., 2008). Indeed, for a given software engineering problem, secondary studies are adequate strategies for the identification of tendencies among different approaches and selection of the most adequate ones (Biolchini et al., 2005). Some research topics of secondary studies in the Software Engineering area are service-oriented architecture (Oliveira and Nakagawa, 2011), software evolvability (Breivold et al., 2012), and software architecture optimization methods (Aleti et al., 2013).

SoS is a very comprehensive term and has several related studies from different areas. The use of a systematic technique for reviewing the literature for investigation into SoS software architectures seems to be a good strategy. In order to conduct our SM, we followed the process proposed by Kitchenham (Kitchenham, 2004) and illustrated in Figure A.1. This process is composed of three main phases: planning, conduction, and reporting. The next section describes how each phase was executed in our SM.

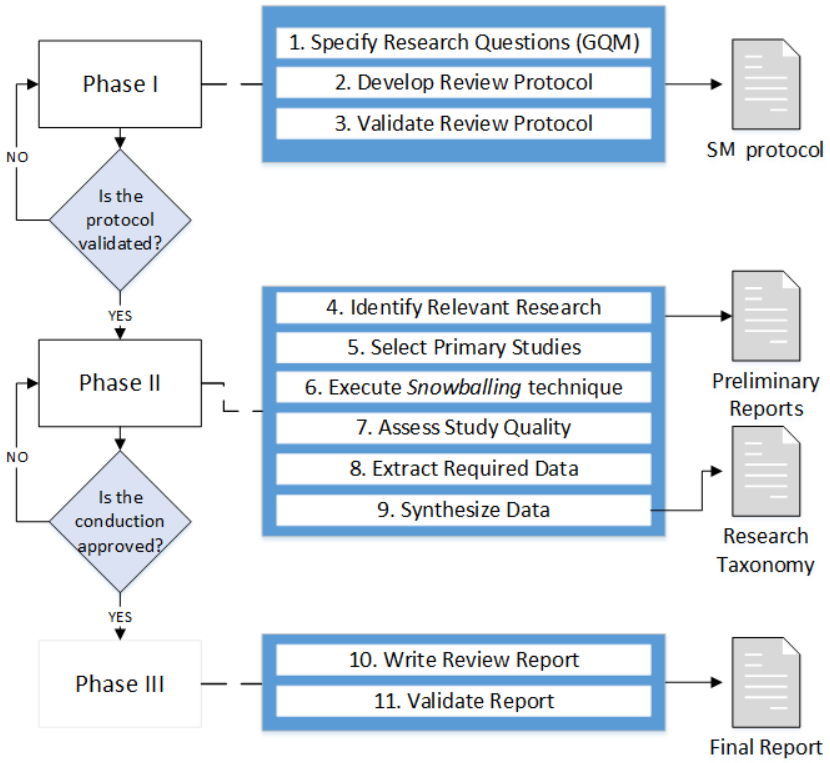


Figure A.1: SM process

A.1 Phase 1: Planning

In this phase, the SM protocol is established, including the research questions, search strategy, selection criteria, and data extraction and synthesis methods. Our SM was conducted from January/2014 to July/2014 and involved six researchers (the co-authors of this work). First, the specific goal, research questions, and associated metrics were organized and established using the Goal/Question/Metric (GQM) approach (Basili et al., 1999). Figure A.2 shows the result of the GQM application. A hierarchical structure supported the establishment of the research questions and required metrics for answering the questions.

A.1.1 Research Questions

As mentioned by Kitchenham and Charters (Kitchenham and Charters, 2007), SMs generally have broader research questions driving them and often ask multiple research questions. Therefore, aiming at identifying as many evidences as possible of the research involving SoS software architectures, the following Research Questions (RQs) were established:

RQ 1: What are the main architecturally significant characteristics of SoS? Although there is a known set of characteristics, characteristics that are architecturally relevant must be investigated. In other words, this RQ will identify characteristics of SoS that are relevant to their software architecture.

RQ 2: What are the main quality attributes of SoS software architectures? Since quality attributes must be incorporated in the software architectures, this RQ aims at identifying quality attributes that are commonly incorporated in SoS software architectures.

RQ 3: How have SoS software architectures been represented? This RQ aims at identifying the most common approaches (e.g., methods, techniques, and models) proposed and/or used to represent SoS software architectures.

RQ 4: How have SoS software architectures been evaluated? The aim of this RQ is to investigate the architectural evaluation methods proposed and/or applied to SoS software architectures.

RQ 5: How have SoS software architectures been constructed? This RQ is concerned with ways of designing SoS software architectures, including architectural styles, architectural frameworks, and architectural patterns.

RQ 6: How have SoS software architectures been evolved? Considering that SoS continuously evolve, this RQ addresses approaches proposed to support evolution of SoS software architectures.

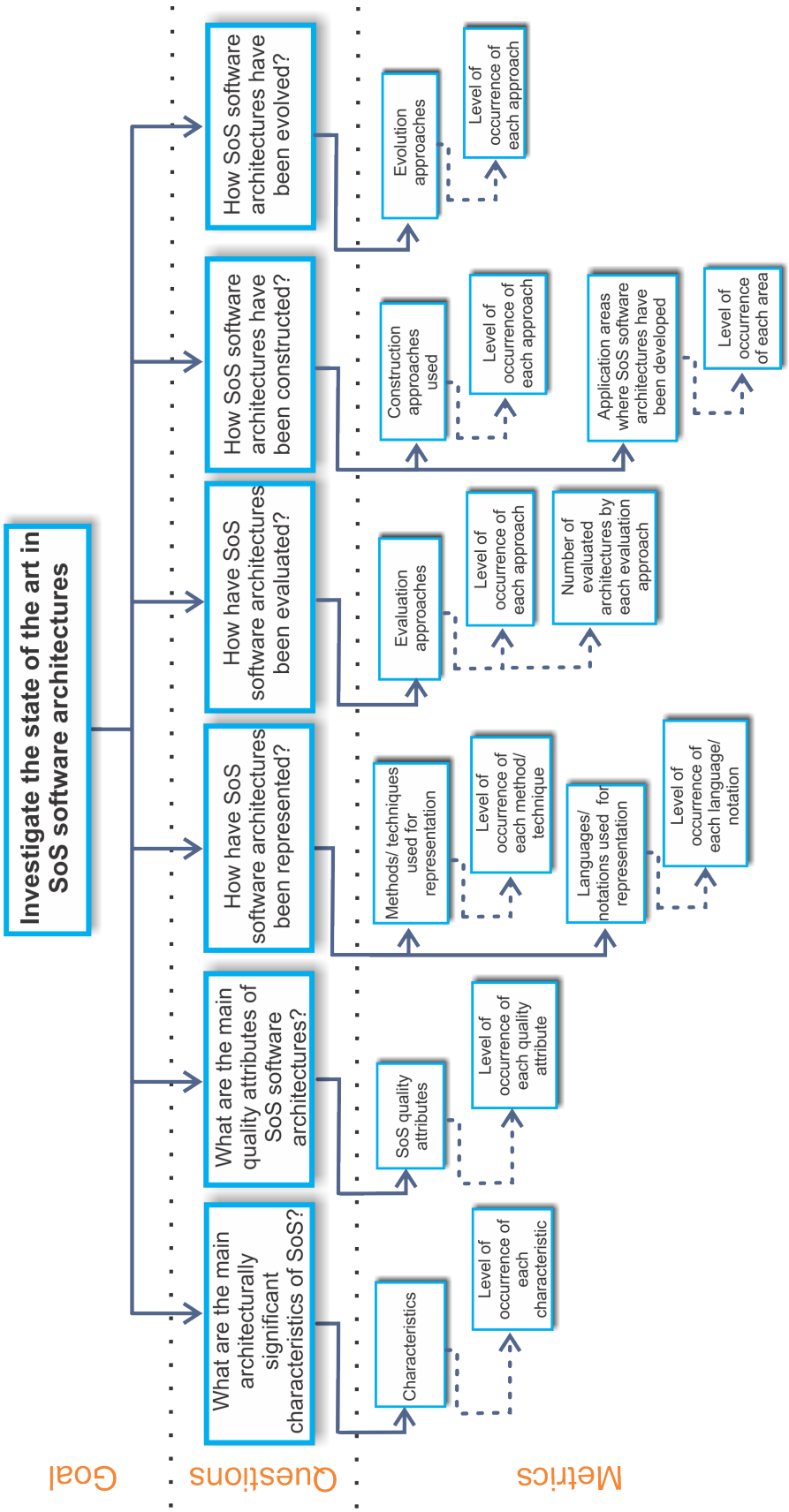


Figure A.2: Defining Research Questions and Metrics using GQM

A.1.2 Search Strategy

The search terms for SMs are **less highly focused** than for SLRs (Kitchenham and Charters, 2007), therefore, in order to establish the search strategy based on the RQs, we initially identified three general keywords: "Software", "Architecture", and "System of Systems". Terms related to "System of Systems" were then identified: "System-of-Systems", "Systems of Systems", and "Systems-of-Systems". The final search string was obtained by the combination of the keywords and their synonyms. Keywords were connected by the logical operator AND, and the synonyms of these keywords were connected by the logical operator OR. The final search string was:

software AND architecture AND ("system of systems" OR "system-of-systems" OR "systems of systems" OR "systems-of-systems").

The accuracy of this search string was verified by a control list, i.e., a list of the primary studies that should be found by this search string. It was established based on the opinion of specialists on SoS, software architecture and the relevant studies identified in Nakagawa et al.(2013). This string was also iteratively applied to the Scopus[1] database and reviewed/improved, in terms of related terms and logical concatenation, until all studies of the control list had been in the results. We believe this final string is the most adequate to searches into primary studies in the area of SoS software architectures.

Regarding the search sources (i.e., publication databases), we chose the most effective ones for secondary studies in Software Engineering area, namely ACM Digital Library[2], IEEEXplore[3], ISI Web of Science[4], ScienceDirect[5], and Scopus (that indexes Springer[6]). Additionally, proceedings of specific SoS events were also included for investigation, namely International Workshop on Software Engineering for System-of-Systems (SESoS), the Workshop on Distributed Software Development, Software Ecosystems, and Systems-of-Systems (WDES), and the International System of Systems Engineering Conference (SoSE). Particularly, the International Council on Systems Engineering (INCOSE) is a relevant source of studies, however, its publications present restrictions of access. Thus, we included for analysis only the INCOSE studies available in the aforementioned basis. The criteria considered for their selection were availability of primary studies,

[1] `http://www.scopus.com/`
[2] `http://dl.acm.org/`
[3] `http://ieeexplore.ieee.org/`
[4] `http://apps.isiknowledge.com/`
[5] `http://www.sciencedirect.com/`
[6] `http://www.springerlink.com/`

scientific publications and conferences coverage, and versatility of results exportation. Whenever possible, we also limited the search space to title, abstract, and keywords. Moreover, our SM considered studies published only in English, as this is the language widely adopted in conferences and journals related to the Software Engineering area and also in the search sources.

To avoid missing important primary study, we applied the snowballing technique, which is a non-probabilistic sampling technique that checks the reference lists of the included studies to retrieve other studies (Wohlin, 2014).

A.1.3 Inclusion and Exclusion Criteria

The inclusion and exclusion criteria are used to evaluate each primary study obtained and select studies that provide evidence about the RQs. The Inclusion Criteria (IC) used in our SM were:

IC 1: The primary study exhibits an architecturally relevant characteristic of SoS;

IC 2: The primary study shows a quality attribute for SoS software architectures;

IC 3: The primary study addresses an approach to represent SoS software architectures;

IC 4: The primary study addresses an approach to evaluate SoS software architectures;

IC 5: The primary study addresses an approach to design SoS software architectures; and

IC 6: The primary study addresses an approach to support the evolution of SoS software architectures.

The Exclusion Criteria (EC) are used to exclude studies that do not contribute to answering the RQs. The exclusion criteria used in our SM were:

EC 1: The primary study is not related to SoS;

EC 2: The primary study is not related to software architecture;

EC 3: The primary study does not have an abstract or its full text is not available;

EC 4: The primary study is written in a language other than English;

EC 5: The primary study is directly related to another primary study of the same author. In this case, only the most recent primary study is considered; and

EC 6: The primary study is a compilation of works or a tutorial in a conference or workshop.

A.1.4 Quality Assessment

In order to analyze the quality of the included primary studies, we developed a checklist containing six Quality Questions (QQ) (see Table A.1) adapted from the generic quality checklist proposed by Kitchenham (Kitchenham, 2004). QQ was established for the assessment of studies of each RQ, as the RQs encompass different focuses. From QQ3 to QQ6, we consider that a study shows an approach evaluated if a survey, case study, experiment, and/or industrial use is contained in this study. The following scale-points to each QQ were adopted: (i) Yes - 1 point; (ii) No - 0 point; and (iii) Partially - 0.5 point.

Table A.1: SM - Checklist for the assessment of the quality of primary studies

Research Question (RQ)	**Id**	**Quality Question (QQ)**
RQ1	QQ1	Is the way the characteristic(s) of SoS influence their software architecture clear?
RQ2	QQ2	Is the way the quality attribute(s) of SoS influence their software architecture clear?
RQ3	QQ3	Has the approach to represent SoS software architectures been evaluated?
RQ4	QQ4	Has the approach to evaluate SoS software architectures been evaluated?
RQ5	QQ5	Has the approach to build SoS software architectures been evaluated?
RQ6	QQ6	Has the approach to evolve SoS software architectures been evaluated?

A.1.5 Selection of Primary Studies

After the primary studies have been searched in the databases using the search string, the studies interesting to our SM are selected in 3 **three** main steps: (i) an initial selection of the studies is conducted based on the reading of title and abstract of each primary study and the application of selection criteria; (ii) the full text of each previously selected primary study is read and the selection criteria are applied again; and (iii) the quality of the final included studies is assessed. Moreover, each study is evaluated by at least two researchers. In case of disagreement, consensus meetings are scheduled to solve it.

A.1.6 Data Extraction and Synthesis Method

Forms based on the metrics showed in Figure A.2 are used to support and organize the data extraction process. Data are extracted independently by reviewers.

It is worth highlighting that we present in details herein the protocol of our SM and information of its conduction. Hence, it will be possible to audit our SM, clarifying all executed steps, and also making it possible to re-conduct it or other SM in related topics.

A.1.7 Threats to Validity

The threats to the validity of our SM are:

- Missing of important primary studies: The publication databases used in this SM are considered the most relevant available ones (Dybået al., 2007; Kitchenham and Charters, 2007). Although no limit was placed on the publication date of the primary studies, some studies may have been missed;
- Specialist's suggestions: Besides using publication databases to retrieve primary studies, we considered studies suggested by specialists in Software Architecture and SoS. We believe that these suggestions have not introduced any bias, since none of the studies was written by these specialists and they have never worked with authors of the studies suggested; and
- Data extraction: Since not all information was clearly available in the primary studies, these information had to be interpreted. In order to ensure the validity of our SM, discussions among reviewers were conducted whenever a doubt occurred.

A.2 List of Primary Studies

Table A.2: SM: list of selected studies

Title	**Citation**
System-of-systems security engineering	(Bodeau, 1994)
Two-level process model for integrated system development	(Rossak et al., 1994)
Architecting principles for systems-of-systems	(Maier, 1998)
Inferred Designs	(Perrochon and Mann, 1999)

Kinesthetics eXtreme: An external infrastructure for monitoring distributed legacy systems	(Kaiser et al., 2003)
Quantifiable Architecting of Dependable Systems of Embedded Systems	(Liang and Luqi, 2003)
Automated generation of integrated architectures and end-to-end network models	(Bonilla et al., 2005)
Architectural framework for a system-of-systems	(Caffall and Michael, 2005)
A unifying framework for systems modeling, control systems design,and system operation	(Dvorak et al., 2005)
System software framework for system of systems avionics	(Ferguson et al., 2005)
Systems of systems and coordinated atomic actions	(Schaefer, 2005)
An adaptable outdoor robotic platform: architecture, communications, and control	(Briggs et al., 2006)
Architecture-Based Interoperability Evaluation in Evolutions of Networked Enterprises	(Chen, 2006)
Rapid systems of systems integration - Combining an architecture-centric approach with enterprise service bus infrastructure	(Kruger et al., 2006)
A distributed simulation approach for modeling and analyzing systems of systems	(Sharawi et al., 2006)
Quality assurance of the timing properties of real-time, reactive system-of-systems	(Shing et al., 2006)
Safety, software architecture and MIL-STD-1760	(Squair, 2006)
DDSOS: A dynamic distributed service-oriented simulation framework	(Tsai et al., 2006)
Specification, Validation and Run-time Monitoring of SOA Based System-of-Systems Temporal Behaviors	(Cook et al., 2007)
Developing Performance-based Requirements for Open Architecture Design	(Naegle, 2007)
Extensible Virtual Environment Systems Using System of Systems Engineering Approach	(Oliveira and Pereira, 2007)
A Discrete Event XML based Simulation Framework for System of Systems Architectures	(Sahin et al., 2007)

Conceptual SOS Model and Simulation Systems for A Next Generation National Healthcare Information Network (NHIN-2): Creating A Net-Centric, Extensible, Context Aware, Dynamic Discovery Framework for Robust, Secure, Flexible, Safe, and Reliable Healthcare	(Sloane et al., 2007)
Data fusion enabled networks	(Vila, 2007)
Architecting communication network of networks for Space System of Systems	(Bhasin and Hayden, 2008b)
A System-of-Systems Engineering GEOSS: Architectural Approach	(Butterfield et al., 2008)
A Method for Collaborative Development of Systems of Systems in the Office Domain	(Carbon et al., 2008)
Architecture Principles and the GEOSS Clearinghouse	(Christian, 2008)
Applying Object Oriented Systems Engineering to Complex Systems	(Cloutier and Griego, 2008)
A Rich Services Approach to CoCoME	(Demchak et al., 2008)
Precision guidance of agricultural tractors for autonomous farming	(Eaton et al., 2008)
Aspect-oriented modeling approach to define routing in enterprise service bus architectures	(Ermagan et al., 2008)
Reasoning about Hybrid System of Systems Designs	(Gamble and Gamble, 2008)
Bridging requirements and architecture for systems of systems	(Haley and Nuseibeh, 2008)
Using Sequence Diagrams to Detect Communication Problems between Systems	(Lindvall et al., 2008)
Toward an Evolutionary System of Systems Architecture	(Schroh et al., 2009)
Toward an Evolutionary System of Systems Architecture	(Selberg and Austin, 2008)
AGSOA - Agile Governance for Service Oriented Architecture (SOA) Systems: A Methodology to Deliver 21st Century Military Net-Centric Systems of Systems	(Sloane et al., 2008)
Fundamentals and architectures of Complex Distributed Systems	(Tianfield, 2008)
Scaling up software architecture evaluation processes	(Zhu et al., 2008)

Towards Behavioral Reflexion Models	(Ackermann et al., 2009)
Resource-Definition Policies for Autonomic Computing	(Calinescu, 2009)
A uniform approach for system of systems architecture evaluation	(Gagliardi et al., 2009)
A systems engineering process for development of federated simulations	(Kewley Jr. and Andreas, 2009)
The verification and validation of software architecture for systems of systems	(Michael et al., 2009)
GI-Cat: A Mediation Solution for Building a Clearinghouse Catalog Service	(Nativi et al., 2009)
Architectural Patterns and Auto-Fusion Process for Automated Multisensor Fusion in SOA System-of-Systems	(Rothenhaus et al., 2009)
Developing an approach for analyzing and verifying system communication	(Stratton et al., 2009)
From Multi-Agent to Multi-Organization Systems: Utilizing Middleware Approaches	(Wester-Ebbinghaus et al., 2009)
Methodology for object-oriented system architecture development	(Acheson, 2010)
A net-centric XML based system of systems architecture for human tracking	(Bowen and Sahin, 2010)
A quality of service framework for adaptive and dependable large scale system-of-systems	(Bull et al., 2010)
Software Engineering Techniques for the Development of Systems of Systems	(Calinescu and Kwiatkowska, 2010)
Facilitating system-of-systems evolution with architecture support	(Chen and Han, 2001)
Using relational model transformations to reduce complexity in SoS requirements traceability: Preliminary investigation	(Dickerson and Valerdi, 2010)
Addressing the Integration Challenge for Avionics and Automotive Systems: From Components to Rich Services	(Farcas et al., 2010)
Eliciting software safety requirements in complex systems	(Menon and Kelly, 2010)
Systems of systems applications for telemedicine	(Petcu and Petrescu, 2010)

Engineering Lessons for Systems of Systems Learned from Service-Oriented Systems	(Simanta et al., 2010)
Assessing system software performance in complex system of systems environments	(Wessel and Meyer, 2010)
Measuring the architectural complexity of military Systems-of-Systems	(Domerçant and Mavris, 2011)
Agile development for system of systems: Cyber security integration into information repositories architecture	(Farroha and Farroha, 2011)
Singleton to sandwich chunking into buslets for better system development	(Hodges et al., 2011)
Service Orientation and Systems of Systems	(Lewis et al., 2011)
Software engineering of component-based systems-of-systems: A reference framework	(Loiret et al., 2011)
Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model-driven systems development process	(Wang and Dagli, 2011)
The process of architecting for software/system engineering	(Chigani and Balci, 2012)
COAST: An Architectural Style for Decentralized On-Demand Tailored Services	(Gorlick et al., 2012)
System of Systems to provide Quality of Service monitoring, management and response in Cloud Computing environments	(Hershey et al., 2012)
Supporting Awareness during Collaborative and Distributed Configuration of Multi Product Lines	(Holl et al., 2012)
Architecting the Human Space Flight Program with Systems Modeling Language (SysML)	(Jackson et al., 2012)
Scaling Up Software Architecture Analysis	(Kazman et al., 2012)
Towards integrated rule-driven software development for IT ecosystems	(Mensing et al., 2012)
Multi-agent architecture with space-time components for the simulation of urban transportation systems	(Nguyen et al., 2012)
Extending VDM-RT to enable the formal modelling of System of Systems	(Nielsen and Larsen, 2012)
Interface specification for system-of-systems architectures	(Payne et al., 2012)
Reengineering legacy systems towards system of systems development	(Ramos et al., 2012)

Model-based development of fault tolerant systems of systems	(Andrews et al., 2013)
Challenges for SoS architecture description	(Batista, 2013)
Net-centric System of Systems framework for human detection	(Bowen and Sahin, 2013)
Semi-formal and formal interface specification for system of systems architecture	(Bryans et al., 2013)
A reliability-based measurement of interoperability for systems of systems	(Jones-Wyatt et al., 2013)
Supporting architectural decision making for systems-of-systems design under uncertainty	(Lytra and Zdun, 2013)
Model-driven systems engineering for netcentric system of systems with DEVS unified process	(Mittal and Risco Martin, 2013)
Perspectives and challenges of reference architectures in multi software product line	(Nakagawa and Oquendo, 2013)
Issues and challenges in ecosystems for federated embedded systems	(Papatheocharous et al., 2013)
Large-scale Smart Grids As System of Systems	(Pérez et al., 2013)
Platform based approach for automation of workflows in a system of systems	(Ploom et al., 2013)
A Model-Driven Approach for Evaluating System of Systems	(Xia et al., 2013)
Perspectives on System of Systems for pilgrimage ritual guidance and management	(Alwakeel et al., 2014)

Appendix

B

Systematic Literature Review on SoS Architecting Processes: Study Protocol and List of Included Primary Studies

This appendix lists, in Table B.4, the primary studies included in the systematic review discussed in Section 2.2.3.

This appendix presents the research protocol and the list of included studies of the SM whose results are summarized and discussed in Section 2.2.

B.1 Research Methodology

An SLR is a well-defined, systematic procedure recently advocated as a useful mean of identifying, evaluating, and interpreting existing work from the literature on a given research topic (Kitchenham, 2004). This type of evidence-based secondary study follows a rigorous methodology that seeks to minimize bias while enabling other researchers to reproduce the same process when exploring the same research topic. Such a methodology can be viewed as the main point that differentiates an SLR from traditional literature reviews as it is able to provide scientific value for the obtained findings.

Despite the greater conduction effort when compared to traditional literature reviews, SLRs have become increasingly popular in the last years mainly due to the multiple

benefits that they offer. First, SLRs are able to provide a comprehensive overview of the state of the art (or state of the practice) on the investigated research topic as well as identify research challenges and opportunities in this context. Second, they provide a credible, unbiased evaluation of the analyzed primary studies. Third, given a software engineering problem, an SLR can be a suitable strategy to identify similarities and/or differences among distinct approaches and reveal the most adequate ones (Brereton et al., 2007).

As outlined by Kitchenham and Charters (2007), an SLR comprises three basic steps (see Figure B.1). The *Planning* step yields a protocol defining the research questions to be answered, the search strategy to be adopted, the criteria to be used to select primary studies, and the methods for extracting and synthesizing data. In the *Conduction* (or *Execution*) step, relevant primary studies on the investigated topic are identified, selected, and evaluated according to the established protocol. Finally, the *Reporting* (or *Analysis*) step aims to aggregate information extracted from relevant primary studies considering the research questions and outlines conclusions from them.

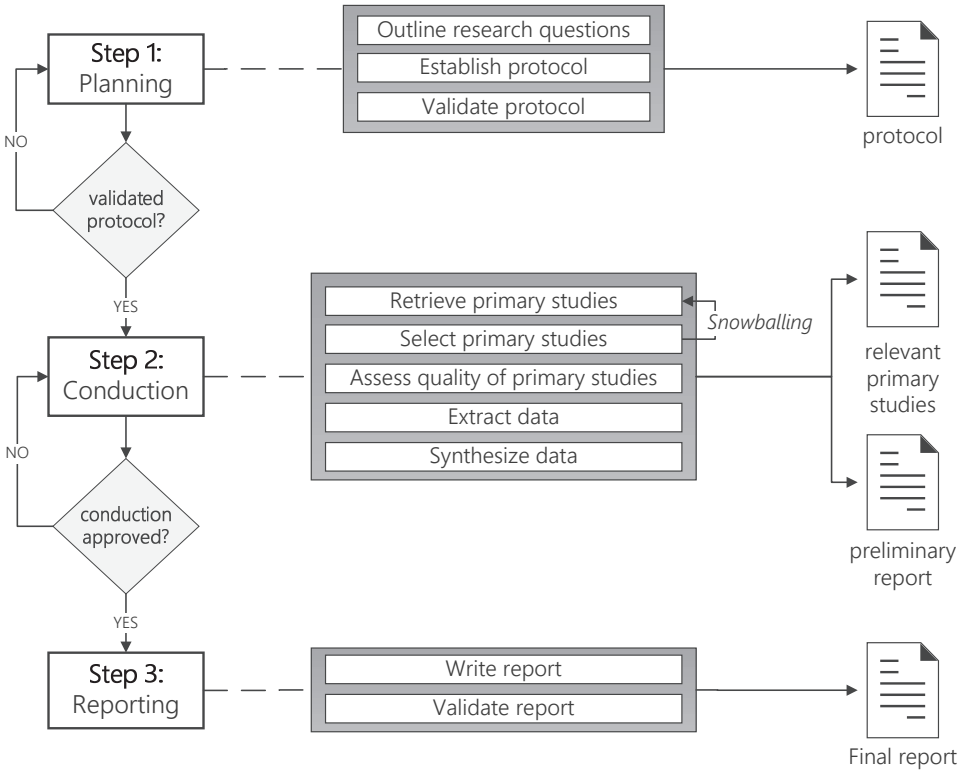


Figure B.1: Process for reviewing literature in SLRs (Kitchenham and Charters, 2007).

B.1.1 Research Questions

Aiming at finding primary studies to understand how SoS software architectures have been constructed and what are the requirements and challenges in this context, we proposed the research questions (RQs) outlined in Table B.1.

Table B.1: Research questions and respective goals

Research question	Goal
RQ1: What are the important steps and artifacts to be considered in the construction of SoS software architectures?	Identify the general steps and artifacts on the architectural design of SoS.
RQ2: What are the important steps and artifacts to be considered in the construction of SoS software architectures?	Identify the general steps and artifacts on the architectural design of SoS.

B.1.2 Search Strategy

In order to retrieve primary studies, we used an automated search procedure performed over five electronic databases (see Table B.2), which are among the most popular ones in Software Engineering and have a high coverage of potentially relevant studies (Chen et al., 2010; Dybået al., 2007). Furthermore, we considered other important criteria, such as: (i) coverage of the electronic database; (ii) content update, i.e., if the publications are regularly updated; (iii) availability of the full text of the primary study; (iv) easiness of building the search through fields and commands available at the electronic database; (v) quality of the results, which is related to the accuracy of the results obtained by the automated search procedure; and (vi) versatility to export results (Dieste et al., 2009).

Based on the defined RQs, four main keywords were initially identified, namely *system of systems*, *software*, *architecture*, and *construction*. In addition, possible variations such as synonyms and singular/plural forms were considered, thereby resulting in the following search string:

```
("system of systems" OR "system-of-systems" OR
"systems of systems" OR "systems-of-systems") AND
(construction OR architecting OR method OR process OR design OR build OR
development) AND software AND architecture
```

in which the main keywords were connected by using the `AND` logical operator. In turn, the possible variations and synonyms were connected by using the `OR` logical operator.

Table B.2: Electronic databases used in the automated search procedure

Database	URL
IEEEXplore	`http://ieeexplore.ieee.org`
ACM Digital Library	`http://dl.acm.org`
ScienceDirect.com	`http://www.sciencedirect.com`
Scopus	`http://www.scopus.com`
Web of Science	`http://www.webofknowledge.com`

Due to technical limitations of the electronic publication databases or low precision of the used search string, an automated search procedure may miss relevant studies. Aiming at overcoming such a limitation and increasing the comprehensiveness of this study, we conducted a *snowballing* procedure (Wohlin, 2014). This non-probabilistic technique consists of checking the reference lists of the studies in order to find additional, potentially relevant studies not retrieved by the automated search procedure.

B.1.3 Selecion Criteria

Selection criteria were used to evaluate each retrieved primary study according to the defined RQs. The main goal was to include studies that are potentially relevant to answer the RQs and to exclude the ones that do not contribute to answer them.

We considered the following two inclusion criteria:

1. **IC1:** The study presents a process or method to construct SoS software architectures.
2. **IC2:** The study establishes a process requirement on constructing SoS software architectures.

We also established the following five exclusion criteria:

1. The study does not address the construction of SoS software architectures.
2. The study is a previous version of a more complete study about the same research.
3. The study does not have an abstract or the full text is not available.
4. The study is a table of contents, foreword, tutorial, editorial, keynote talk, or summary of conference/workshop.

5. The study is not written in English, the most common language in scientific papers.

In this SLR, a given primary study is considered as relevant if it does not meet any of the aforementioned exclusion criteria and it meets at least one inclusion criterion.

B.1.4 Data Extraction

In order to extract data from each selected primary study aiming at synthesizing results and supporting conclusions, we built a data extraction form with 12 items, as outlined in Table B.3. These data items were defined based on each RQ and other relevant information. During the data extraction process, data of each primary study were independently extracted by the researchers when considering each research question and recorded on a spreadsheet. Conflicts found during the process were solved by discussions between the researchers.

Table B.3: Data items extracted from selected primary studies

Number	Item
1	Title
2	Author(s) and respective affiliation(s)
3	Publication year
4	Venue
5	Goal of the study
6	Application domain (or generic, if not specified)
7	Discussed challenges and/or process requirements
8	Proposed solution approach
9	Architectural analysis activities/artifacts
10	Architectural synthesis activities/artifacts
11	Architectural evaluation activities/artifacts
12	Additional comments

B.1.5 Quality Assessment

A reliable mechanism to increase the level of confidence in the findings of secondary studies is establishing criteria to assess the quality of the selected primary studies as means of ensuring that the collected evidences are relevant and present scientific value (Kitchenham et al., 2009; Zhou et al., 2015). We performed a simplified procedure with

a checklist aiming to assess: (i) the rigor of the research methods employed to establish the validity of the study; (ii) the credibility of the study for ensuring that its findings are meaningful; and (iii) the relevance of the study (Zhang et al., 2011; Zhou et al., 2015). We have used the following two quality assessment questions:

1. **Q1:** Is the proposal well defined in terms of activities, artifacts, and roles?

2. **Q2:** Does the study evaluate the proposed solution?

We adopted three possible answers for each quality assessment question, namely *yes*, *partially yes*, and *no*. Each of these answers was quantified by assigning a numerical score to it, that is, *yes* = 1.0, *partially yes* = 0.5, and *no* = 0.0. In the end, the quality assessment score of a given primary study is determined by summing all scores assigned to it with respect to the quality assessment questions. Therefore, a given primary study may receive scores ranging from 0.0 (minimum) to 2.0 (maximum) since there are only two quality assessment questions.

B.2 Threats to Validity

The conducted SLR and its results may have been affected by some threats to validity. In the following, we discuss some of these limitations.

Incompleteness of the search procedure. The completeness of this SLR may have been affected by missing relevant studies. In order to reduce this threat, we have used electronic databases (see Table B.2) that are among the most relevant available sources in Software Engineering (Chen et al., 2010; Dybået al., 2007). In addition, the snowballing procedure (Wohlin, 2014) was performed aiming at finding additional studies not retrieved by the automated search procedure. Nonetheless, there are still limitations. First, some studies may have been missed due to technical limitations of the automated search engines, an issue that is out of the control of the researchers. Second, the selected electronic databases do not represent an exhaustive list of publication sources, so that other databases might also be included. Therefore, other possibly relevant studies could have been identified and considered in this SLR.

Bias on study selection. In order to make the results of this SLR reproducible, the protocol presented in Section B.1 clearly established the search terms used in the automated search procedure, search sources, and criteria for selecting the primary studies. However, different researchers tend to have different understandings on these criteria, so that the results of the study selection performed by different researchers are likely to be varied. Even though the drawn conclusions may have been influenced by the researchers'

opinions, we have striven to mitigate the effect of any personal bias or misinterpretation by adopting a multiple-revision strategy.

Inaccuracy of data extraction. Bias on data extraction may result in inaccuracy of the extracted data items, thus affecting the analysis of the selected studies. We have striven to reduce this bias by clearly defining the data items outlined in the data extraction spreadsheets. In addition, the data items to be extracted in this SLR were discussed among the researchers and agreed upon their meaning.

Bias on data synthesis. Not all studies sufficiently and clearly describe the details of information to be extracted as data items aiming at supporting the answers to the defined RQs. Therefore, we have had to infer certain pieces of information regarding data items during data synthesis. In order to minimize the inaccuracy of such inferences, we have conducted discussions aiming at solving any disagreement and clarifying potential ambiguities.

B.3 SLR: List of Selected Studies

Table B.4: Systematic literature review: list of selected studies

Title	**Citation**
Two-level process model for integrated system development	(Rossak et al., 1994)
Quantifiable software architecture for dependable systems of systems	(Liang and Luqi, 2003)
A System-of-Systems Engineering GEOSS: Architectural approach	(Butterfield et al., 2008)
A process for systems-of-systems architecting	(Griendling and Mavris, 2010)
A design methodology for real-time distributed software architecture based on the behavioral properties and its application to advanced automotive software	(Aoyama and Tanabe, 2011)
A method for the generation and evaluation of architecture alternatives on the cloud	(Iacobucci and Mavris, 2011)
Architecting ultra-large-scale green information systems	(Chen and Kazman, 2012)
The process of architecting for software/system engineering	(Chigani and Balci, 2012)
Scaling up software architecture analysis	(Kazman et al., 2012)
Towards integrated rule-driven software development for IT ecosystems	(Mensing et al., 2012)
A data-centric capability-focused approach for system-of-systems architecture modeling and analysis	(Ge et al., 2013)
Supporting architectural decision making for systems-of-systems design under uncertainty	(Lytra and Zdun, 2013)
Goncalves2015 A meta-process to construct software architectures for system of systems	(Gonçalves et al., 2015)

Appendix

C

The OMG's Essence Standard

C.1 Essence Language

Essence Standard is an effort of the SEMAT[1] (Software Engineering Method and Theory) community, published as an OMG's standard in 2014[2], to support practitioners, project managers, and process engineers in authoring, application, and management of development processes[3]. Despite it be a recent standard, successful applications were already reported from industry (Jacobson, 2015; McDonough, 2014). Among the abundance of unique processes that are hard to compare, the difficulty of experimental evaluation and validation, and the gap between academic research and its practical application, Essence Standard proposes key features and goals to encompass these lacks that have influenced our choice to represent SOAR using it (Jacobson et al., 2013; Object Management Group (OMG), 2014):

- Possibility of using different levels of abstraction when describing processes, supporting the establishment of common bases for similar processes sharing vocabulary and knowledge. In SoS, different views of processes can be provided for different stakeholders. For example, process engineers can be more interested in specific de-

[1] `http://www.semat.org`

[2] The most recent version of Essence Standard is the 1.1 published in 2014.

[3] The Essence specification originally adopts the term "method" to represents a composition of development practices. In this thesis, we will use the convention that "process" is the term that refers to this composition.

tails of activities while managers can be more interest in general aspects and results of ongoing work. It is also possible to structure different levels of abstraction in the same process, allowing project managers to choose how complete is the process that they need. For instance, inexperienced teams can be interested in more lower-level views of a process with more specific guidelines;

- Focus on practical use, supporting project teams on identifying opportunities of process evolution and monitoring process health, thus evolving their way of working to be as adequate as possible for different development stages of each system. Therefore, as systems evolve in their life cycles, their processes also change maintaining the suitability at each development stage; and

- Essence Standard provides a foundation for easy process authoring, which allows teams to share their development solutions in a modularized way independently from single and complex processes.

Essence Standard comprises a language and a kernel for creation, use, and improvement of software engineering processes. Essence Language is a language for process authoring proposed to make processes visible and useful to developers. Despite it pays extreme attention to syntax, emphasizes intuitive and concrete graphical syntax over formal semantics. The focus is on providing a description in a language that can be easily understood by the wide developers community whose interests are to quickly understand and use this language (Object Management Group (OMG), 2014). Represented in this language, Essence Kernel was conceived to be a very comprehensive common basis for grounding any software development process. Therefore, it captures essential elements of software engineering common to all software engineering methods, structured in a reusable way on any development process (Object Management Group (OMG), 2014). Additionally, Essence Standard has a development environment for Essence Language, the *EssWork Practice Workbench*[4] to creation of processes with Essence. This tool provides an easy, intuitive way to support project teams and process engineers on developing and deploying customized processes on Essence Language.

C.1.1 Fundamentals

Figure C.1 presents the layered architecture of using Essence Language for process authoring. The first main structure that can be described with this language is the kernel, which provides a common ground for development endeavors. Given a scope, e.g. Software Engineering, a kernel provides common basis that determines "what must be done" for

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

all processes related to it. Therefore, it provides basic elements (i.e., common concepts, goals, and competencies) that must be considered when authoring processes to a such scope. At a lower abstraction level, practices are the second main structure in Essence Language, which describe "how to do" determinations described on kernels. Therefore, practices provide a more concrete guidance to do something with a specific objective. They must describe how to handle a specific aspect of a development endeavor, including all relevant elements, e.g., activities and work products, necessary to express that the purposes of the practice were achieved. Finally, processes are compositions of practices assembled to satisfy a specific project context. This structure of composition allows separation of "what" must be done that is included in the kernels, from "how" to perform it, which is included in practices and processes.

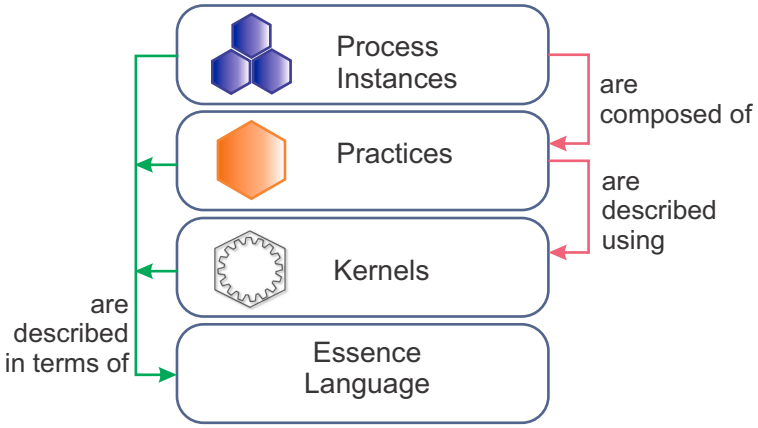


Figure C.1: Essence architecture (Adapted from (Object Management Group (OMG), 2014))

Furthermore, Essence Language offers both textual and graphical syntax, in which nobody is constrained to use a graphical notation in situations in which textual notation is easier to handle, and vice-versa. Furthermore, it has not only a static base for processes description, but also additional dynamic semantics to enable description of what is actually being done in a running project. Therefore, process representations can be more than static specifications of what to do, but active guides that can be consulted in running development endeavors returning directions on what to do next (Object Management Group (OMG), 2014).

C.1.2 Main Elements

Figure C.2 informally shows the main elements of Essence Language, their relationships, and graphical representations. Following, we describe these elements used to compose kernels, practices, and processes. In this description, we considered the level of details needed for understanding the proposal of this thesis.

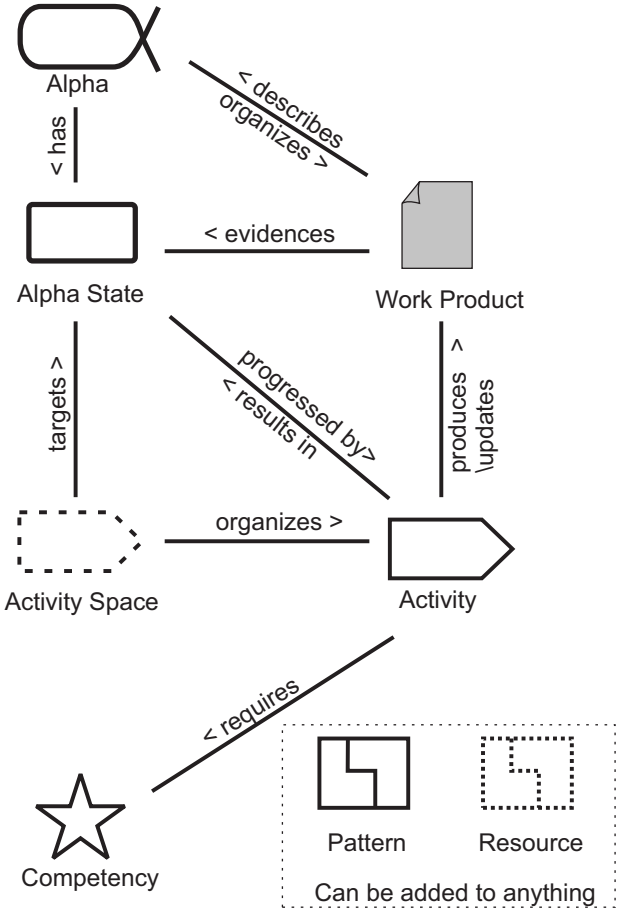


Figure C.2: Essence Language conceptual overview (Adapted from (Object Management Group (OMG), 2014))

Areas of Concern: are the central concerns that impact the process conduction and developers have to pay special attention to. The are graphically represented with different colors assigned to the graphical representation of the other elements of Essence Language. Therefore, each element fall into at most one of these colors expressing what is its main focus in the process context.

Alpha: is an acronym of "Abstract-Level Progress Health Attribute" that represents any issue whose process evolution can be understood, monitored, directed, and controlled. Changes in alphas expressing evolution towards achieving the objectives of the process and supporting project teams to understand their own way of work.

Alpha State: is a specification of a state situation that an alpha can assume. It represents important and remarkable stages in the life-cycle of an alpha. The alphas have well-defined states and each alpha state is determined by the fulfillment of checkpoint items in a specific checklist. A checkpoint is a condition, that can be tested as true or false, that contributes to the determination of whether a state has been attained or not. For each alpha, the states are organized in a linear order

denoting the usual way of progression. Despite the change of states can assume any sequence according to each particular life-cycle, when a particular state is reached, all the previous states in the linear order must to be reached as well. This approach enables accurately plan and control the alphas evolution through expected states.

Work Product: representation of concrete artifacts of value and relevance in a process. Work products provide a concrete representation to alphas describing them in different forms, such as models, documents, specifications, and software parts. Thus, they can be inputs or outputs of activities that are created, modified, used, or deleted during a process.

Activity Space: a high-level element that can be understood as a repository for a group of activities with a common goal. It represents "something to be done" and is described in terms of alphas and respective states. The definition of activity spaces is based on alphas and their states. For each activity space, a set of previously required alpha states is described its input and these states must change during its execution reaching the expected output, i.e., the set of alpha states to be reached revealing the well-execution of such activity space.

Activity: guidance of "how" to perform activity spaces to change alpha states. Activities are defined in practices (not in kernels) to accomplish activity spaces purposes, i.e., to reach the expected set of alpha states.

Competency: specification of abilities, capabilities, attainments, knowledge, and skills necessary to do a specific type of work in a process. Each activity can be associated with competencies that performer(s) must have to perform it.

Patterns and Resources: generic concepts that can be attached to any element of Essence Language. Resources can be templates for work products and tools for activities. Patterns are arrangements of Essence Language elements in meaningful structure, e.g., a pattern to organize a workflow for a set of activity spaces. When instantiating processes based on predefined practices, process authors can adapt these practices by adding or replacing specialized resources and patterns.

C.2 Essence Kernel

Essence Kernel is a body of knowledge about software development endeavor. It defines a common basis for software development practices in a scalable and flexible way. Therefore, this kernel provides a basis in which different software engineering practices can be composed to different needs. Moreover, this kernel can be also extended with convenient

kernel extensions to build more specific bodies of knowledge for any recurrent challenge on software engineering. The Essence Kernel is structured into tree discrete areas of concern, each one having its own representing color and focusing on a specific aspect of software engineering. Figure C.3 shows this structure in which "Customer" (green) area is related to actual use and exploitation of software system; "Solution" (yellow) area includes everything related to the specification and development of the software; and "Endeavor" (blue) area that encompasses everything about team and its way-of-working. Following, an overview of Essence Kernel is presented including its alphas and activity spaces.

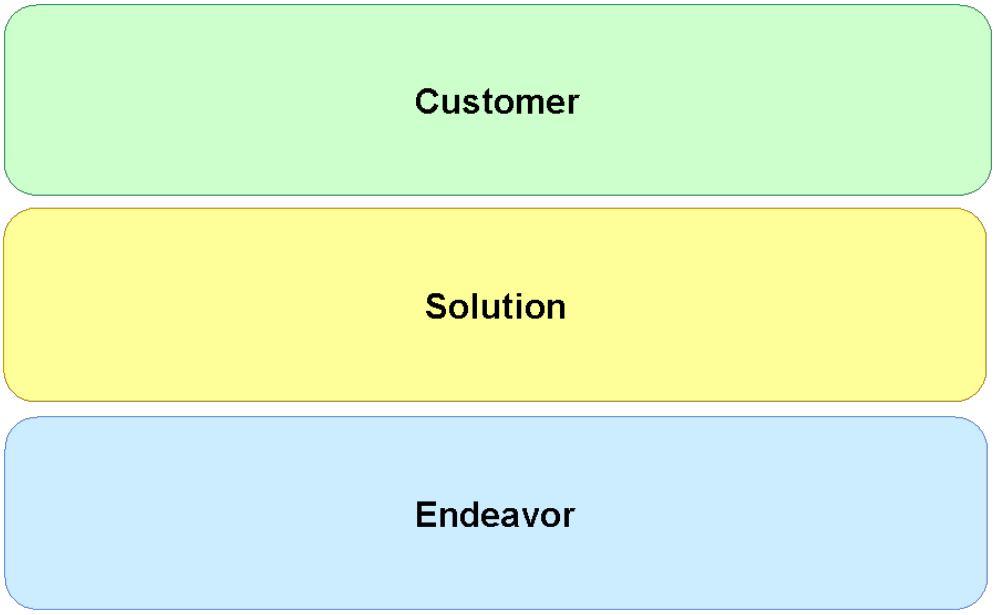


Figure C.3: Essence Kernel Areas (Object Management Group (OMG), 2014).

C.2.1 Essence Kernel Alphas

Alphas determine the "things to work with" in a kernel. Figure C.4 shows Essence Kernel alphas, their inter-relationships, and areas of concern. Following, each alpha is described.

In Customer area of concern, the main challenge is to understand opportunities that must be addressed.

- **Opportunity:** Circumstances that makes it appropriate to develop or change a software system. The set of opportunities justifies software engineering endeavors and represents the understanding of stakeholders' needs. It helps on shaping requirements for a new software system by providing justification in terms of needs.
- **Stakeholders:** Any subject (e.g., people, groups, organizations) who affects or is affected by the related software system. Stakeholders may have involvement

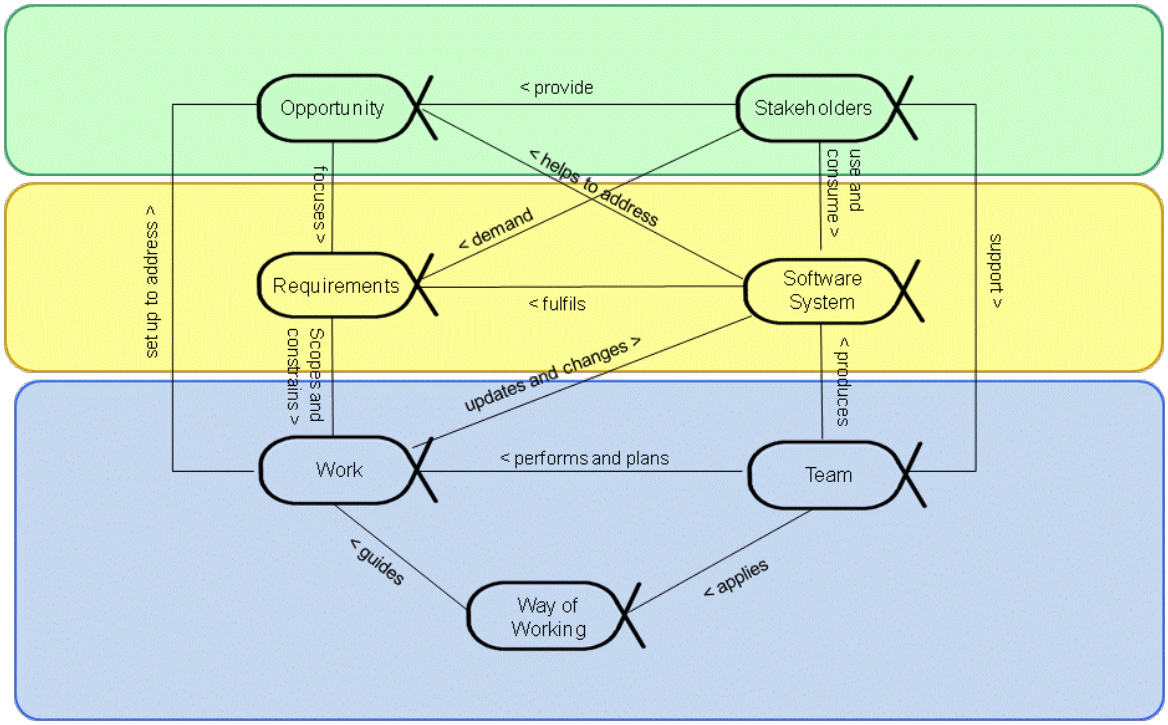


Figure C.4: The Alphas of Essence Kernel (Object Management Group (OMG), 2014).

with software system development in several different aspects; for example, being customer or even participant of development team.

In Solution area of concern, elements and strategies must be encompassed to adequately build a system.

- **Requirements:** What software system must accomplish in order to meet opportunity and satisfy stakeholders.
- **Software System:** A system based on software that is the primary product of software engineering endeavors in a project. A software system can be included in a larger software, hardware or business solution.

In Endeavor area of concern, the elements related to team's way-of-working have to be addressed.

- **Work:** Mental or physical effort done in order to achieve a given result. In Essence Kernel, work is everything that team does to meet the goals related to software development. Moreover, specific practices must be conceived in order to guide this work and also express the team's way-of-working.
- **Team:** A group of people with specific skills and competencies that are actively engaged in development, maintenance, delivery or support of a specific software system.

- **Way-of-Working:** A tailored set of common practices and tools used by a team to guide and support their work. The standardization and specification of way-of-working allows its monitoring and systematic evolution.

C.2.2 Essence Kernel Activity Spaces

Essence Kernel also provides a set of activity spaces that is shown in Figure C.5. This set complements alphas describing the "things to do" on software engineering endeavors. Following, each activity space is described grouped by areas of concern.

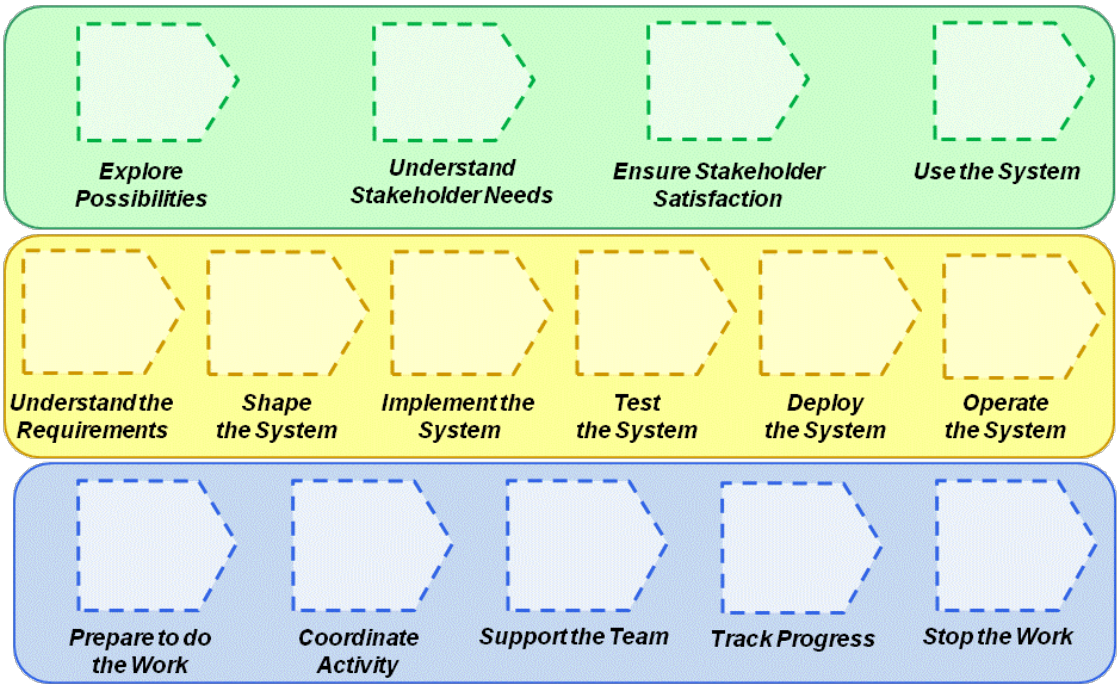


Figure C.5: Activity spaces of Essence Kernel(Object Management Group (OMG), 2014).

In Customer area of concern, team has to understand the system context and explore opportunities by adequately involving stakeholders.

- **Explore Possibilities:** Explore possibilities analyzing each opportunity to be addressed by system.
- **Understand Stakeholder Needs:** Interact with stakeholders to understand their needs and ensure that right results will be achieved.
- **Ensure Stakeholder Satisfaction:** To verify if identified opportunities have been successfully addressed and gain the stakeholders' acceptance of the system produced.
- **Use the System:** Perform verification and validation of system by observing the use of system in a live environment.

In Solution area of concern, team has to develop adequate solutions to meet identified opportunities, thus satisfying stakeholders.

- **Understand Requirements:** Identify the requirements by establishing a shared understanding of what the system to be produced must do.
- **Shape the system:** Design the system by optimizing development, change, and maintenance efforts. This activity space can include the overall design and architecting of system.
- **Implement the System:** Execute the system, or a part of it.
- **Test the System:** Verify whether the produced system meets established requirements.
- **Deploy the System:** Make the validated system available for use outside of development sphere.
- **Operate the System:** Support system use in its real operation environment.

In Endeavor area of concern, team has to guarantee its own operation in an expected way-of-working.

- **Prepare to do the Work:** Set up the needed elements of work environment.
- **Coordinate Activity:** Manage the team's work. This includes all on-going planning, replanning, and control of the team on performing its activities.
- **Support the Team:** Help the team members on supporting themselves, collaborate and improve their way-of-working.
- **Track Progress:** Measure and assess the relevant progress indicators of team.
- **Stop the Work:** Stop software engineering endeavor and verify team's responsibilities.

C.2.3 Essence Kernel Competencies

Essence Kernel also provides a set of competencies that describe the "needed competencies" when performing a software engineering endeavor. Figure C.6 presents these competencies that are described following. Following, competencies are described grouped by areas of concern.

Figure C.6: Competencies of Essence Kernel (Object Management Group (OMG), 2014).

In Customer area of concern, team has to be able to explore business and technical aspects of system domain and to have the ability to accurately communicate views of their stakeholders.

- **Stakeholder Representation:** The ability to gather, communicate, and balance the needs of other stakeholders, and to accurately represent their views.

In Solution area of concern, team has to have skills to build and operate a software system that fulfill established requirements.

- **Analysis:** Ability to understand opportunities and their related stakeholder needs, and to transform them into an agreed and consistent set of requirements.
- **Development:** Competency of design and program effective software systems following standards, norms, and technologies agreed by team.
- **Testing:** Competency to test system using agreed verification strategies.

In Endeavor area of concern, team has to be able to be auto manageable.

- **Leadership:** Individual capability of inspire and motivate a group of people to perform a successful work.
- **Management:** Ability to plan, coordinate, and evaluate the work done by a team.

Appendix

D

Using EssWork Practice Workbench to Build a SOAR-based Process Instance

EssWork Practice Workbench [1] is a tool that offers support to automate the generation of process instances in Essence Language. It is an environment based on Eclipse [2] 3.7 Indigo, and is therefore a Java [3] application. The main purpose of the EssWork Practice Workbench is to provide a comprehensive development environment for processes, methods, and practices authoring. This tool supports practice authors and process engineers to focus on essential business values of practice descriptions, and it makes easy and intuitive to develop and deploy processes instead of be concerned with details of Essence Language. In this context, an interactive version of Essence Kernel is available to be used in EssWork Practice Workbench allowing creation of kernel extensions and practices as well as their composition into processes instances.

Regarding the use of EssWork Practice Workbench, this appendix presents an illustrative example of a process instance generated in this tool. We considered the generation of a process instance based on SOAR-A to perform architectural analysis in a flood monitoring SoS project. Section D.1 introduces the flood monitoring application domain and discusses how SoS can be a suitable system class of this domain. Section D.2 describes a

[1] Avaiable for download at `http://www.ivarjacobson.com/Practice_Workbench_Download/`

[2] `http://www.eclipse.org/projects/project-plan.php?planurl=/eclipse/development/plans/eclipse_project_plan_3_7.xml#target_environments`

[3] `http://www.java.com`

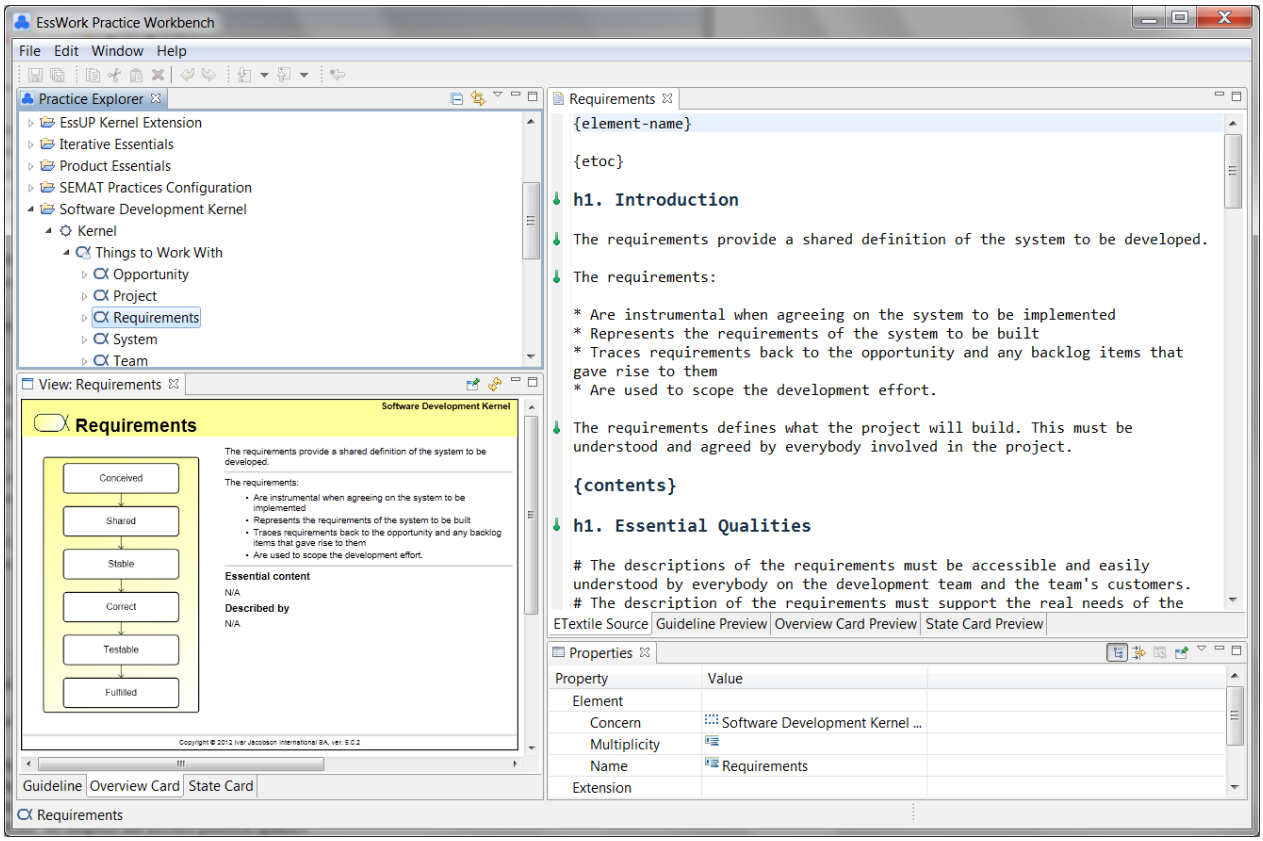


Figure D.1: Competencies of EssWork Practice Workbench.

process instance and illustrate how the SOAR representation in EssWork Practice Workbench [4] can provide support to process instantiation for SoS software architectures.

D.1 The Flood Monitoring Application Domain

The occurrence of floods in urban areas traversed by rivers represents a critical problem that can cause human and material losses (Degrossi et al., 2013). There are several occurrences across the globe affected by this scenario, such as Queensland between 2010-11, Thailand in 2011, China in 2012, and Germany and Hungary in 2013 (Horita et al., 2014). It has been noticed that rain forecast from meteorological systems and satellite data are not sufficient to support the management of flood emergencies. Since damages incurred during a flood scenario are directly correlated with the river level, up-to-date emergence actions also require monitoring of river water flow and level (Zhou and De Roure, 2007). In this context, there are several relevant initiatives to monitor urban rivers (Horita et al., 2014; Zhou and De Roure, 2007).

[4]The description of SOAR in EssWork Practice Workbench is available as HTML site at: `http://www.start.icmc.usp.br/html/SOAR/`. Furthermore, we provide a contact channel to request the source code of SOAR.

Flood monitoring systems can include several other constituent systems, such as: (i) embedded systems of water level sensors, which are spread in floodprone areas near the river and monitor the water level; and (ii) data management and analysis systems to collect and analyze information from sensors transmitting warnings when necessary. An example of approach to use these constituents is building Wireless Sensor Networks (WSNs) composed of motes, which are tiny hardware/software platforms equipped with an embedded CPU, low power wireless networking capabilities, and simple level sensors (Ueyama et al., 2010). These sensors can be based on pressure and/or ultrasound to respectively gauge depth and average speed of water. WSNs have been increasing employed in several real-world applications, also in flood monitoring in urban areas (Hughes et al., 2011). Systems that follow this approach are mainly composed of sensors spread in the proximity of the river to monitor physical and/or environmental conditions related to flood detection, e.g., water level and flow rate of the river. Data gathered by these sensors are forwarded through wireless network connections to a base station gateway that analyzes such data and then triggers alerts if a flood condition is detected. A possible strategy to architect these networks is to transmit sensed data in a multihop communication, in which data sensed by a mote in its respective site is successively sent to neighbors until reaching the gateway station that will process them. This communication strategy can take place by using wireless network connections such as WiFi, ZigBee (IEEE 802.15.4), General Packet Radio Services, and Bluetooth (Ueyama et al., 2010).

There are challenges in the flood monitoring domain related to how a complex network of heterogeneous systems can interact and operate to deliver flood monitoring information as expected considering multiple stakeholders. Regarding these challenges, SoS is a suitable class of systems to that domain. Cooperation of different systems can be established as emergent behaviors to meet global SoS capabilities, e.g., level monitoring reports and flood warnings. Furthermore, systems in this domain also involve multiple structures of stakeholders with different needs, e.g., researchers, emergency organizations, and local population.

D.2 An Process Instance to Flood Monitoring SoS

In this section, we describe how a process instance can be created with SOAR for a flood monitoring SoS, named FMSoS. We built this example based on the literature of flood monitoring, and supposing information about project particularities, such as team structure and decisions made during process instantiation. This example illustrates how a small team can use SOAR-A in its project to produce a process instance. This project is simpler than it could be in a real SoS because our purpose is to illustrate SOAR use

instead of being a complete description of everything that should be done in this kind of project.

D.2.1 Characterizing a Flood Monitoring SoS

As a first step before process instantiation, an analysis is performed to verify if the system to be built is in the category covered by SOAR, i.e., an acknowledged SoS. In this perspective, Table D.1 presents an analysis and the classification of FMSoS based on our conceptual model (previously presented in Section 2.1.2). First and foremost, FMSoS is software-intensive, software platforms and embedded software of water level sensors that must collaborate with this SoS. Moreover, there is geographical distribution, in which constituent systems are connected and collaborate towards the accomplishment of global missions to detect risk of flood and to trigger warning messages. Functionalities provided by this SoS arise from the collaboration among constituent elements, which are not able to provide such global missions if they are separately considered, thus encompassing the so-called emergent behavior. Despite the existence of a central control for FMSoS, the functionalities and cooperation of constituents depends on the negotiation with their owners. With these characteristics and the it is ones presented in Table D.1, FMSoS can be classified as an acknowledged SiSoS and, hence, compatible with SOAR.

D.2.2 Building a process instance in EssWork Practice Workbench

A process instance is a composition of practices assembled and adapted to a specific project. When authoring processes using SOAR, it is possible to choose a more adequate level of support according to both level of expertise of developers and process maturity of related organizations. In our example, we considered a small team inexperienced with SoS projects, in which members share multiple roles and responsibilities. In this case, there is no separated members or teams acting to conceive and manage the process instances. Development team is responsible to conceive, perform, and evolve its own development processes, including the architecting processes instance described in this appendix. We considered that development team agreed on use the more complete level of support offered by SOAR, in which SOAR practices are used in process instances. With the utilization of EssWork Practice Workbench, it is possible to produce and document process instances, obtaining as final products not only process instances documented in the tool but also HTML sites autocatically generated for them.

Figure D.2 exemplifies the edition, in EssWork Practice Workbench, of *analysis plan* work product to FMSoS project. In this case, the team added a new point to check this work product that refers to the use of redmine. Furthermore, it is possible not only edit

Table D.1: Analysis of FMSoS

SiSoS Model concepts	**WSN-Based Flood Monitoring System**
Global mission	Monitoring a river to detect risk of flood and to trigger warning messages
Software dominance	Software systems of constituents (e.g., software of level sensors) are essential to FMSoS operation
Operational independence	Meteorological systems and surveillance systems can operate independently of the FMSoS
Managerial independence	Level sensors, meteorological systems, and surveillance systems are independently managed, developed, and maintained
Evolutionary development	FMSoS must afford evolutionary changes in the set of constituents
Emergent behavior	Alerts of imminent floods can only be produced with the cooperation of river monitoring systems operating together
Geographical distribution	Level sensors, gateway, and other constituents are geographically dispersed
Connectivity	Level sensors and gateway are connected through wireless network connections
SiSoS Category (Classification)	***Acknowledged SoS***, because FMSoS functionalities depend on the negotiation with constituent owners

SOAR-A elements including particularities of each project, but also add new elements as illustrated in Figure D.3, in which is possible to notice that, to support the creation of elements aligned with the Essence Standard, the tool automatically provides the set of textual guidelines about fields to be filled.

The aforementioned edition of elements mainly occur in practices and kernels. After the practices and kernels were changed according to the project needs, a process instance must be finalized by assembling them. Figure D.4 shows the starting of this assemblage. After the process instance is named, the practices and kernels are selected. After that, the process instance must be described by filling the fields shown in Figure D.5.

Figure D.6 shows an HTML version of FMSoS process instance generated in Essence Workbench tool. In the left board, it is possible to navigate through descriptions of all elements included in the process instance, e.g., alphas, activity spaces, activities, and work products. The right board presents these descriptions according to what was selected in the left board. In Essence Workbench tool, development team can edit and share this instance. HTML sites describing process instances can be also created with Essence

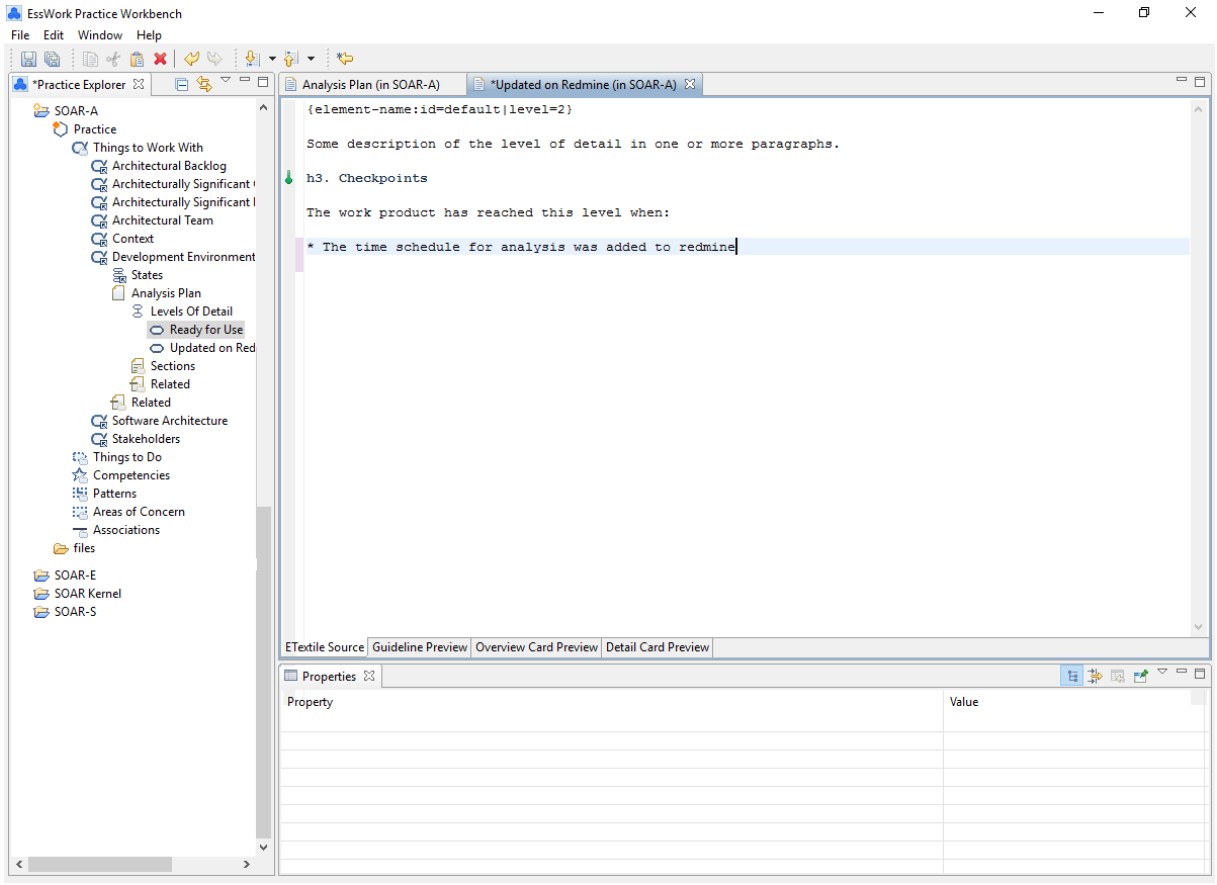


Figure D.2: Editing *analysis plan* work product

Workbench tool to facilitate the disclosure of their content to stakeholders. Essence Workbench tool can generate different views of process elements with different levels of detail. For example, Figure D.7 shows the summarized view, called “card view”, for the *planning analysis* activity, including required alphas and competencies, summary of goals, and alpha states to be reached (i.e., criterion name). Figure D.8 shows an extended view regarding the same activity, including a summary, the textual description of all related items, additional figures if included by team, etc.

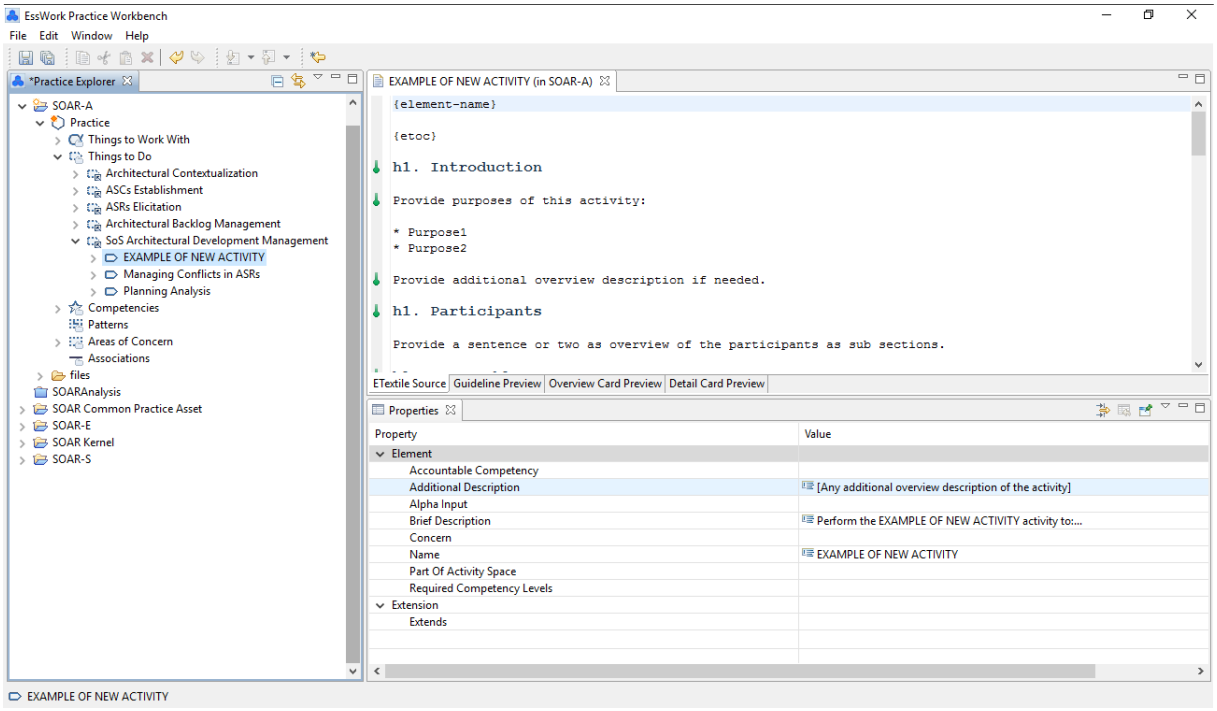


Figure D.3: Adding a new activity to SOAR-A

Considering the inherent evolutionary development of SoS, a process instance must evolve in accordance with SoS evolution. In this perspective, the strategy of the FMSoS project is to execute architectural analysis until a minimum set of ASRs is reached to initiate architectural synthesis and evaluation. Therefore, only SOAR-A is initially applied to support the establishment of activities, tasks, and work products for the first development iteration. Team members establish tasks to reach alpha states expected to express the well execution of architectural analysis.

Table D.2 presents activities, tasks, and work products established for the first iteration. First column presents activities from SOAR-A scheduled for the process instance. The first activity to be executed is *planning analysis*, in which team members will plan, organize, and convey to stakeholders the activities of first iteration. Tasks proposed for activities are described in the second column. The first task is about a meeting of development team to built an analysis plan based on its understanding about alpha states in the project. For this, team members play a technique proposed in Essence Standard called "poker game". This technique is based on the use of state cards that can be generated in Essence Workbench tool. For each alpha, team members use state cards to individually assess and choose which alpha states they believe the project curretly are and which states must be reached after the first iteration. Then they confront their choices and agree about what are the current alpha states and which states must be reached after the iteration. Figure D.9 presents an example of state cards for *architectural backlog* alpha. Each card presents an state and a respective checklist to be applied to verify if the state is reached.

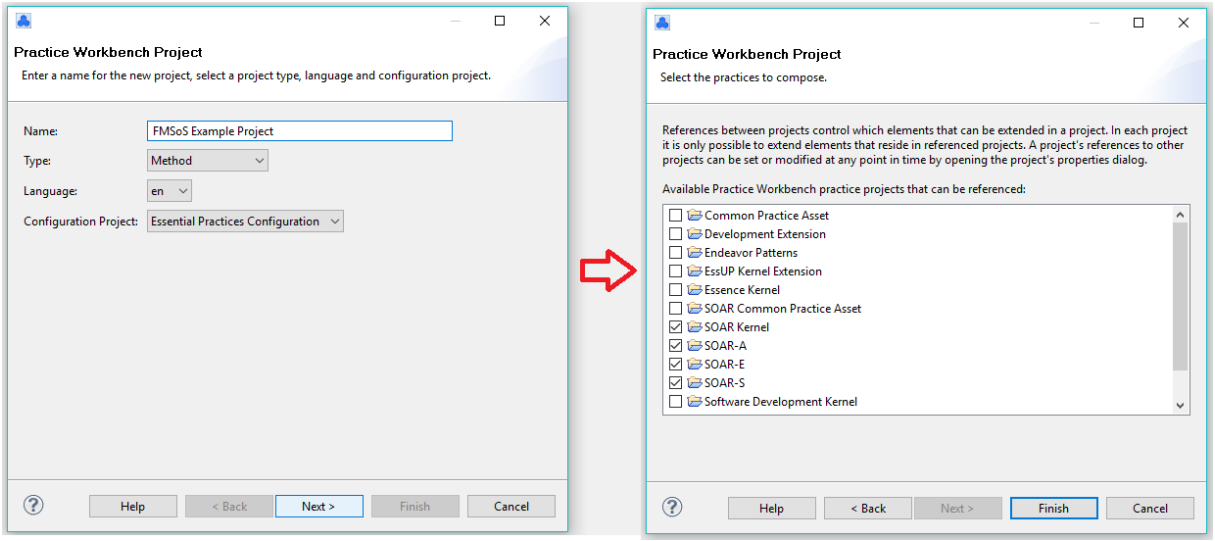


Figure D.4: Example of a process instance

Furthermore, the states are cumulative, e.g., when *updated* state is reached, *available* and *established* states have already been reached.

Work products to be manipulated in activities and tasks are presented in third column. These work products must express alphas and their states. SOAR practices suggest possibilities of work products and process authors must choose/adapt the more adequate ones for each development context. For example, *analysis plan* is implemented in FMSoS by using the Redmine Tool[5] to organize, update, and document this work product. During Task 1 of *planning analysis* activity, development team establishes a time schedule of two weeks to be followed in the first iteration.

Another work product from SOAR-A also considered is the *Context documentation*, which documents the SoS all information related to architectural context, e.g., organizational documents, governmental rules, etc. Also following SOAR-A recommendations, this documentation included a *mission model* to be produced in Task 2 of *identifying architectural context* activity, which must describe the global mission of SoS and individual missions of its constituent systems, such as: (i) monitoring river level for river monitoring systems; (ii) monitor city areas for surveillance systems; and (iii) monitor weather for meteorological systems. Fourth column presents the alpha states to be reached after each task is done. For example, the *stakeholders* alpha reaches *involved* state after Task 3 of *planning analysis* activity is executed, in which team members convey the analysis plan to stakeholders. Fifth column indicates status of tasks execution. According to this column, Tasks 1 and 2 of *planning analysis* activity was already done, Task 3 of the same activity is being performed and other tasks were not initiate yet.

Table D.2: Tasks of process instance

[5] `http://www.redmine.org/`

Activity	Task	Work Products	Alpha State to be Reached	Task Status
Planning Analysis	Task 1: make a development team meeting to play poker game with development team to understand the current alpha states and plan architectural analysis	Analysis plan Work scheme (Redmine)	Development Team: Formed	Done
	Task 2: contact the related stakeholders	Stakeholders map	Stakeholders: Involved	Done
	Task 3: make a meeting with stakeholders to show the iteration plan	Analysis plan (Redmine)	Development Environment: Working	Doing
Identifying Arch. Context	Task 1: identify what is relevant to be part of architectural context, e.g., organizational documents, and governmental rules	Context Documentation	Context: Arch. aspects identified	To do
	Task 2: meeting to establish and agree a missions model with the stakeholders			To do
Identifying ASCs	Task 1: conduct interviews to identify the stakeholders' concerns	List of ASCs	ASCs: established	To do
Managing Conflicts in ASRs	Task 1: analyze the general requirements documentation and the list of ASCs to identify possible conflicts that impact in the software architecture	ASRs documentation, List of ASCs	Stakeholders: in agreement	To do

Eliciting ASRs	Task 1: review requirements documentation already provided externally from architecting process activities to identify ASRs from general requirements	ASRs documentation	ASRs: elicited	To do
	Task 2: promote an discussion meeting to raise agreement regarding the ASRs			To do
	Task 3: produce and include in the ASRs documentation a quality model			To do
Identifying Self-ASRs	Task 1: analyze all generated ASRs documentation to identify possible new requirements from the software architecture	ASRs documentation	ASRs: self-checked, FMSoS: analyzed	To do
Updating Arch. Backlog on Analysis	Task 1: define a template for architectural backlog documentation, guidelines to,edit this documentation, and who is allow to perform this edition	Architectural Backlog Documentation (online repository)	Architectural Backlog: Established	To do
	Task 2: review and updated any document or information generated/changed during the iteration should be included in the backlog documentation	Architectural Backlog Documentation (online repository)	Architectural Backlog: Updated	To do

Based on graphic representation of Essence Language elements (see this representation in Section C.1.2), team members are free to manually built their own graphics and figures, and add them to process instance, according to target audiences and development contexts. Figure D.10 shows one of these possibilities: the representation of the evolution of alpha states in the first iteration and activities planned to reach these states. As FMSoS evolves during iterations, the architecting process must also evolves, with changes in practices, activities, tasks, and work products, to maintain its adequability to FMSoS project.

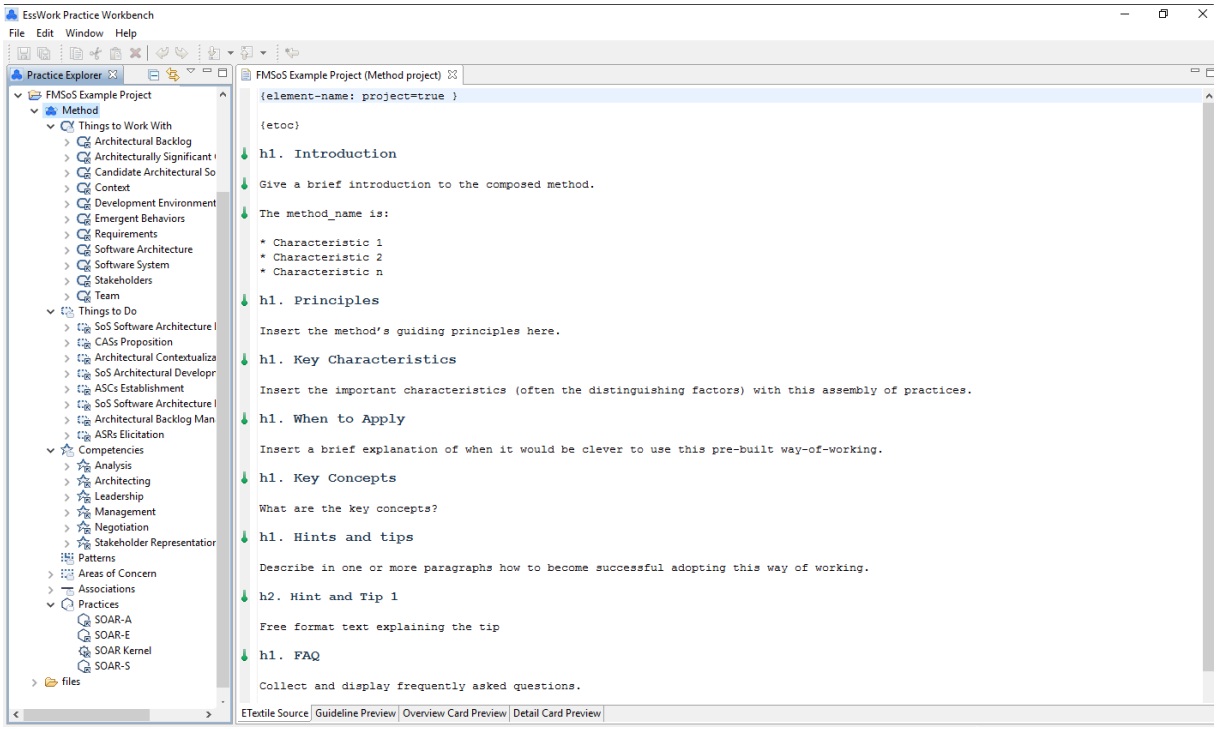


Figure D.5: Editing a final process instance

Several iterations are expected to establish a satisfactory set of ASRs and *ASRs Documentation* is the main work product to be incrementally produced during analysis. After first poker game, team members agreed on four initial iterations of 15 days will be focused on architectural analysis, and this is registered in *analysis plan*. During further poker games, alpha states must be analyzed to guide team to understand if this time schedule is adequate or if the plan must change. In FMSoS, there is a requirements document already leveraged including general requirements and the development team must analyse in Task 1 of *eliciting ASRs* activity which of these requirements are significant to architectural design, i.e., ASRs. When a minimum set of ASRs is achieved, *ASRs* alpha must reach the state *Self-checked*, in which the FMSoS ASRs were elicited including ASRs possibly generated by the software architecture itself. *ASRs Documentation* must contain established and agreed ASRs for the SoS software architecture.

An example of relevant ASR in FMSoS is the need to detect false positives, which is related to an emergent behavior resulting from the interactions among the constituent systems. Although both river monitoring and meteorological systems can independently emit alert messages indicating a critical condition for flooding, only the cooperation between them can avoid false positives. Also following SOAR-A directions, ASRs documentation includes a quality model, i.e., a set of quality characteristics and of relationships between them (ISO/IEC, 2011). In this perspective, performance, fault-tolerance, and availability are generic quality attributes that are relevant to the context of this flood monitoring. Nevertheless, there are other specific attributes that play an important role in the software

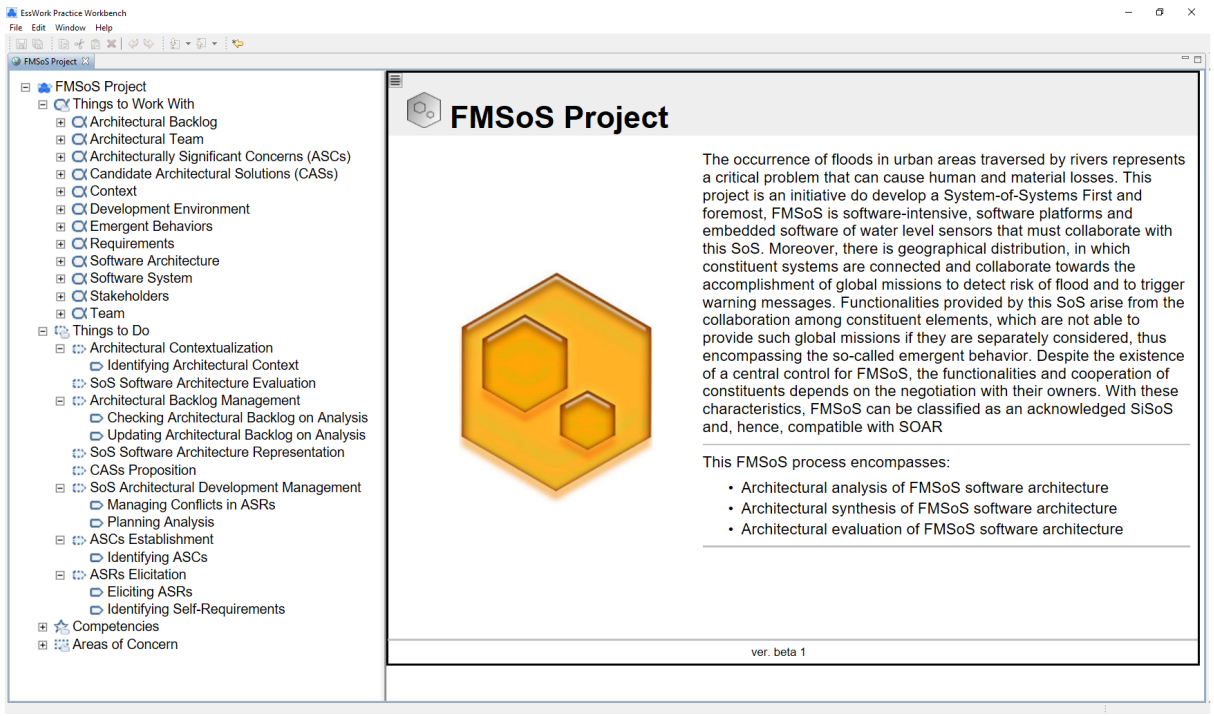


Figure D.6: HTML of process instance for FMSoS

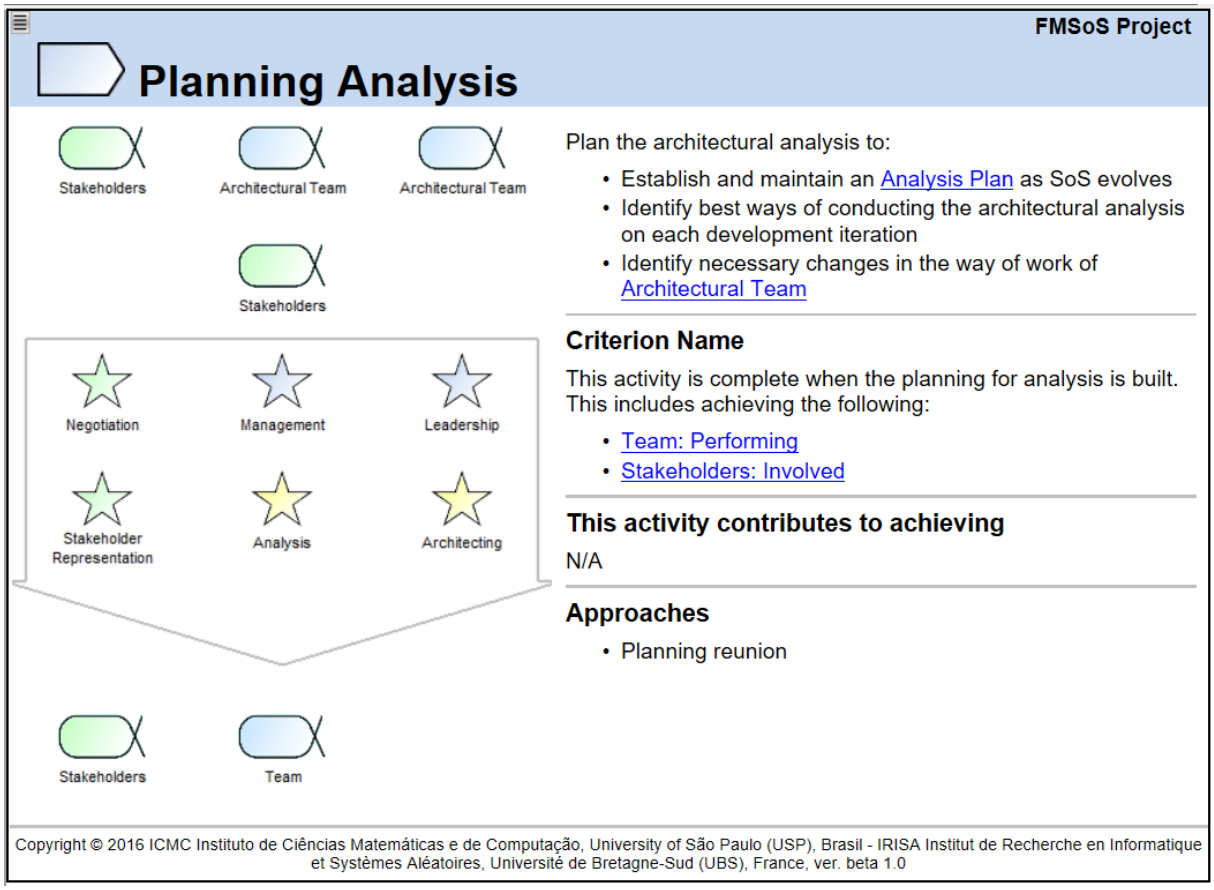
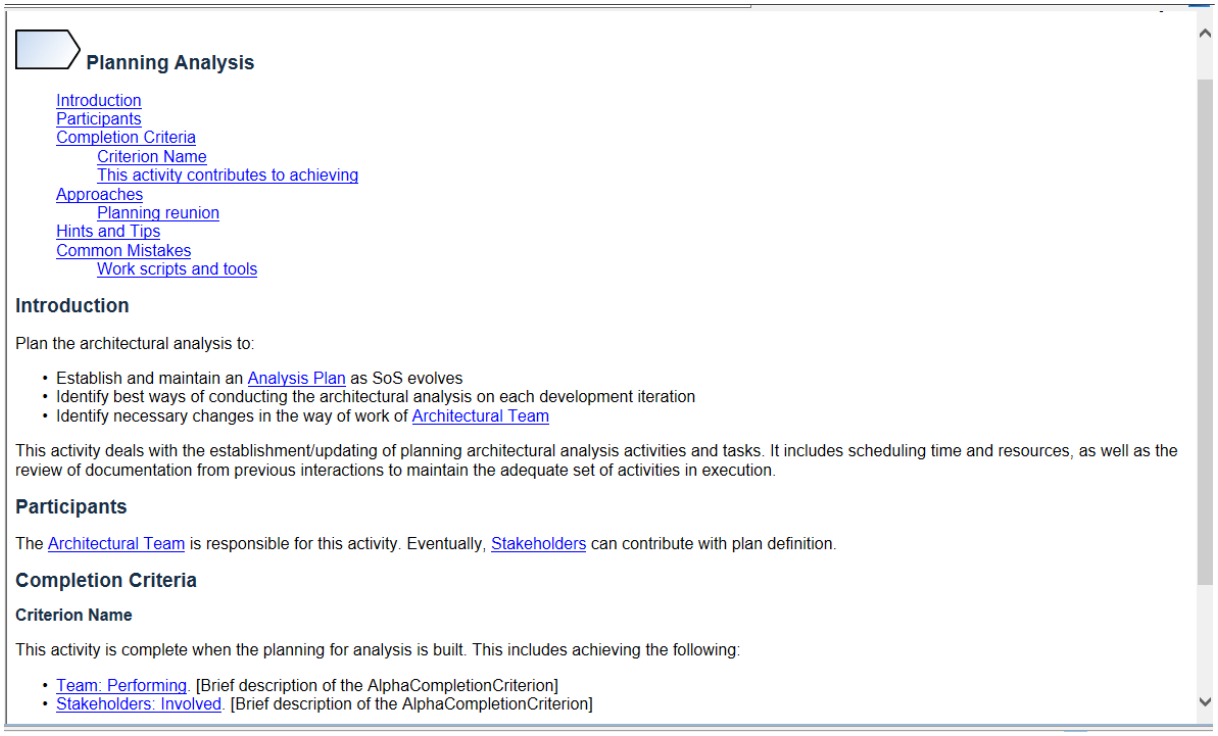


Figure D.7: Card view of acitivity

architecture of this SoS and hence must be handled as ASRs. For instance, accuracy is a quality attribute directly related the global mission of FMSoS of detecting floods with



Planning Analysis

Introduction
Participants
Completion Criteria
Criterion Name
This activity contributes to achieving
Approaches
Planning reunion
Hints and Tips
Common Mistakes
Work scripts and tools

Introduction

Plan the architectural analysis to:

- Establish and maintain an Analysis Plan as SoS evolves
- Identify best ways of conducting the architectural analysis on each development iteration
- Identify necessary changes in the way of work of Architectural Team

This activity deals with the establishment/updating of planning architectural analysis activities and tasks. It includes scheduling time and resources, as well as the review of documentation from previous interactions to maintain the adequate set of activities in execution.

Participants

The Architectural Team is responsible for this activity. Eventually, Stakeholders can contribute with plan definition.

Completion Criteria

Criterion Name

This activity is complete when the planning for analysis is built. This includes achieving the following:

- Team: Performing. [Brief description of the AlphaCompletionCriterion]
- Stakeholders: Involved. [Brief description of the AlphaCompletionCriterion]

Figure D.8: Card view of *planning analysis* acitivity

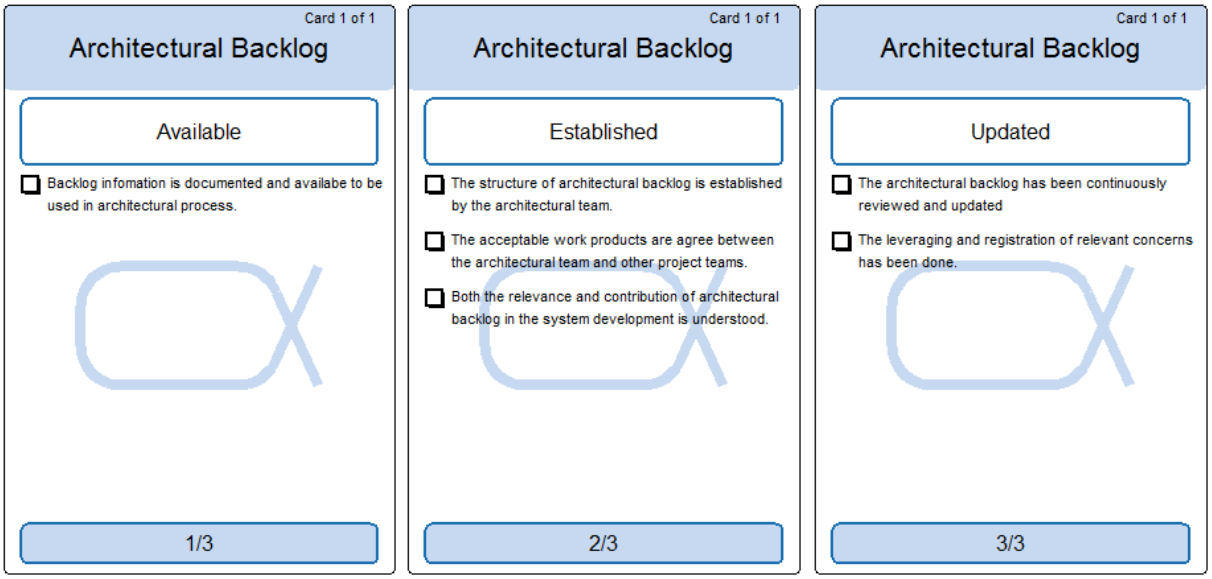


Figure D.9: State cards generated for *Architectural Backlog*

maximum confidence because inaccurate information can negatively impact disaster management strategies. Different constituent systems and their arrangement within FMSoS may have different influence with respect to accuracy of measures and predictions.

D.2.3 Following Iterations

Alpha states help to understand both progress and health of development processes. Therefore, the development team must review, at each development cycle or in pre-defined

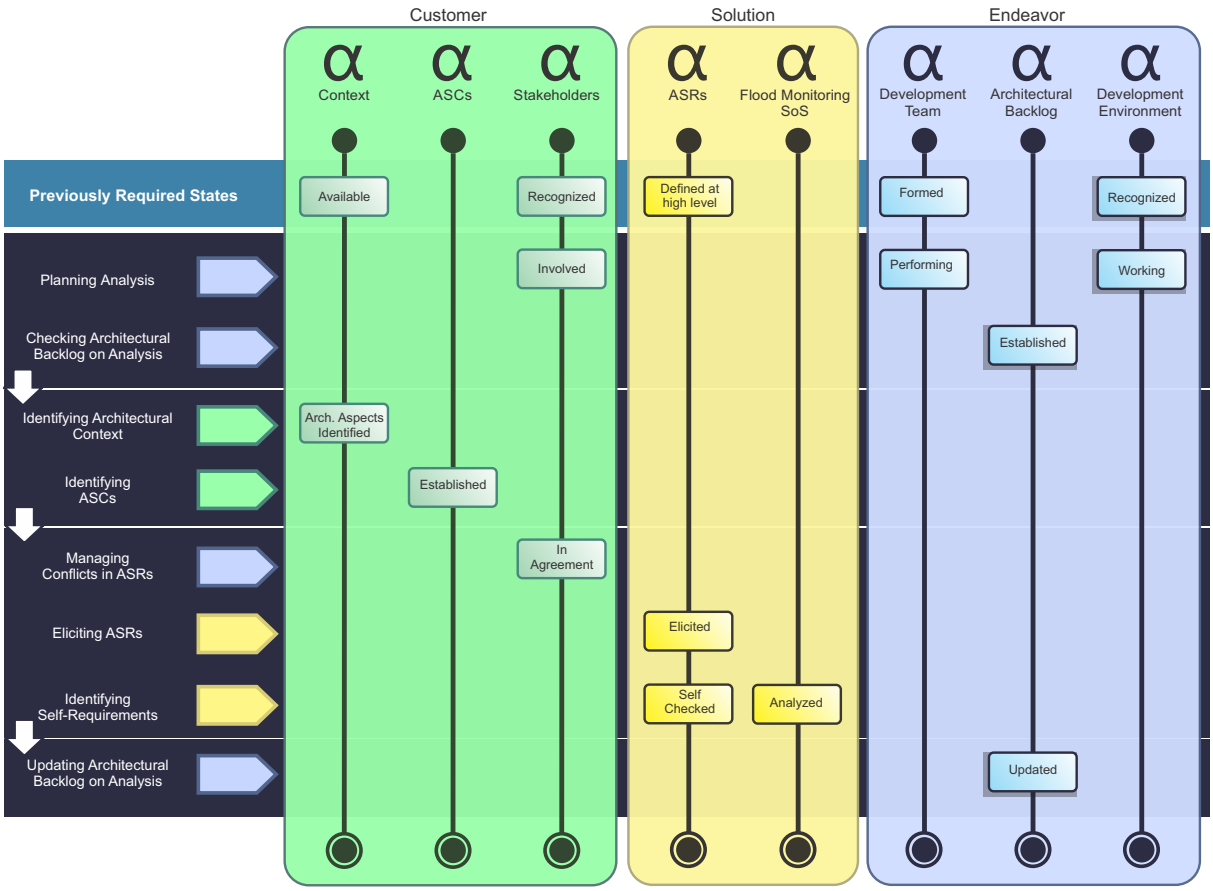


Figure D.10: Alpha states evolution chart (Adapted from (Jacobson et al., 2013))

verification points, if alpha states were reached as expected. In our example, this review is done during *planning analysis*, in which development team agree about alpha states, plan its evolution, and made adjustments in the process instance when necessary, e.g., the further introduction of SOAR-S and SOAR-E activities and work products. New alphas and alpha states can be also proposed by the development team when necessary to help on verify its project evolution. For example, the development team could propose a *project stage* alpha, with states to express when the architecting process admits the introduction of architectural synthesis and evaluation. In this illustrative appendix, we provide a simple and summarized example of how SOAR can support the instantiation an architecting processes for acknowledged SoS. This example shows SOAR-A flexibility to accommodate the particularities of each project, such as stage of development or development strategies used by team members.

Appendix

E

Survey Questionnaires

This appendix presents the online questionnaires used in the surveys conducted to evaluate some elements of SOAR approach, i.e., SOAR Kernel, SOAR-A practice, and SOAR-E practice. For sake of simplicity, only questions are presented and additional instructions of how to fill the online forms and how to access and read supplementary documentations are omitted. More details about these surveys are available in the Chapters 3, 4, and 6.

E.1 Questionnaire for personal profiles

Since the level of expertise of each subject was relevant on all surveys of this thesis, a common set of questions was included in all surveys in order to identify this information. These questions are presented as follows.

Telling us a little about yourself will help us in further analysis.

1. **How would you grade your knowledge/expertise about software architectures?** *
   Mark only one oval.

	1	2	3	4	
Beginner	◯	◯	◯	◯	Expert

2. **How would you grade your knowledge/expertise about SoS?** *
   Mark only one oval.

	1	2	3	4	
Beginner	◯	◯	◯	◯	Expert

3. **Which role describes you best?** *
   Mark only one oval.

   - ◯ Academic researcher
   - ◯ Practitioner of industry
   - ◯ Both of previous choices

E.2 Questionnaire of SOAR Kernel's survey

After read SOAR Kernel description in Essence Language, participants were asked to answer questions about it. Additionally, an overview of Essence Language and description of Essence Kernel were provides as reference documents. Specific questionnaire applied to evaluate the SOAR Kernel is presented as follows.

Evaluating completeness

1. **Do you think that SOAR Kernel encompasses all general aspects that are relevant when constructing acknowledged SoS software architectures?** *
   Mark only one oval.

   - ◯ Yes (SOAR Kernel satisfactorily encompasses this issue)
   - ◯ Partially yes (needs minimal additions)
   - ◯ Partially not (needs critical additions)
   - ◯ No (SOAR Kernel is not representative at all)

2. **If you do not answered "yes" in the last question, please point out the aspect(s) that might be missing.**

3. **The alphas must determine the "things to work with" in a kernel. In this context, how would you grade the alphas of SOAR Kernel?** *
   Mark only one oval.
   - Yes (the set of alphas is acceptable)
   - Partially Yes (less important alphas are missing)
   - Partially No (critical alphas are missing)
   - No (the set of alphas does not accomplish its purpose)

4. **Activity spaces must determine the "things to be done" in a kernel. In this context, how would you grade the activity spaces of SOAR Kernel?** *
   Mark only one oval.
   - Yes (the set of activity spaces is acceptable)
   - Partially Yes (less important activity spaces are missing)
   - Partially No (critical activity spaces are missing)
   - No (the set of activity spaces does not accomplish its purpose)

5. **Please point out other alphas or activity spaces that should be included.**

Evaluating correctness

6. **How would you grade SOAR Kernel in terms of correctness?** *
Mark only one oval per row.

	Correct (no wrong or misunderstood statements)	Partially correct (less important statements should be reviewed)	Partially incorrect (critical statements should be reviewed)	Totally incorrect (the amount of errors make SOAR useless)
Alphas				
Activity Spaces				

7. **Please, point out any statement/concept that you think incorrect/misunderstood, also indicating its respective alpha(s), activity space(s).**

Evaluating coherence

8. **How would you grade SOAR in terms of coherence?** *
Mark only one oval per row.

	Coherent (no relevant conflicts or wrong placed elements)	Partially coherent (less important elements are conflicting or wrong placed)	Partially coherent (critical elements are conflicting or wrong placed)	Totally incoherent (the amount of conflicts and/or disorganization makes SOAR useless)
Alphas				
Activity spaces				

9. **Please, point out any alpha or activity space that you think incorrectly placed or conflicting.**

Evaluating usability

10. **How would you grade the SOAR Kernel and its representation?** *
Mark only one oval per row.

	1 (low)	2	3	4 (high)
Clear				
Well organized				

11. **Specific difficulties.**
Regarding your reading, point out any difficulty that you have on understanding the SOAR Kernel and its elements.

Extra comments and suggestions

12. **Please, give your improvement sugestions to SOAR Kernel.**

E.3 Questionnaire of SOAR-A's survey

After read SOAR-A Practice description in Essence Language, participants were asked to answer questions about it. Furthermore, an overview of Essence Language and descriptions of SOAR Kernel and Essence Kernel were provides as reference documents. Specific questionnaire applied to evaluate the SOAR-A Practice is presented as follows.

Evaluating completeness

1. **Do you think that the work products of SOAR-A practice encompass all general aspects relevant to the architectural analysis in acknowledged SoS software architectures?** *
   Note: remember that SOAR-A is a comprehensive solution to be fulfilled with the particularities of each SoS project.
   Mark only one oval.
   - ◯ Yes (it does not miss any relevant aspect)
   - ◯ Partially yes (it needs minimal additions)
   - ◯ Partially not (it needs critical additions)
   - ◯ No (it is not representative at all)

2. **How would you grade the set of activities defined in the SOAR-A practice?** *
   Mark only one oval.
   - ◯ The set of activities is acceptable
   - ◯ Less important activities are missing
   - ◯ Critical elements are missing
   - ◯ The set of activities does not accomplish its purpose at all

3. **Please point out any other element (e.g., alpha, work product, activity space, activity) that should be included as well as any other issue that you consider as relevant.**

Evaluating correctness

4. **How would you grade the activities of the SOAR-A practice in terms of correctness?** *
   Mark only one oval.

   - ◯ Correct (there are no wrong or misunderstood statements)
   - ◯ Partially correct (less important statements should be reviewed)
   - ◯ Partially incorrect (critical statements should be reviewed)
   - ◯ Totally incorrect (the amount of errors makes SOAR-A practice useless)

5. **Please point out any statement/concept that you consider as incorrect/misunderstood.**

Evaluating coherence

6. **How would you grade the activities of the SOAR-A practice in terms of coherence?** *
   Mark only one oval.

   - ◯ Coherent (there are no relevant conflicts or incorrectly placed elements)
   - ◯ Partially coherent (less important elements are conflicting or incorrectly placed)
   - ◯ Partially coherent (critical elements are conflicting or incorrectly placed)
   - ◯ Totally incoherent (the amount of conflicts and/or disorganization make the SOAR-A practice useless)

7. **Please point out any element (e.g., alpha, work product, activity space, activity) that you think incorrectly placed or conflicting.**

Evaluating usability

8. **How would you grade the usability of the SOAR-A practice description?** *
   Mark only one oval per row.

	1 (low)	2	3	4 (high)
Clear	◯	◯	◯	◯
Well organized	◯	◯	◯	◯

9. **Specific difficulties**
   Regarding your reading, point out any difficulty that you had on understanding the SOAR-A practice and its elements.

Extra comments and suggestions

10. **Please give your improvement sugestions to the SOAR-A practice.**

E.4 Questionnaire of SOAR-E's survey

After read the SOAR-E Practice description in Essence Language, participants were asked to answer questions about it. Furthermore, an overview of Essence Language and descriptions of SOAR Kernel and Essence Kernel were provides as reference documents. Specific questionnaire applied to evaluate SOAR-E Practice is presented as follows.

Evaluating completeness

1. **Do you think that the work products of SOAR-E practice encompass all general aspects relevant to the architectural evaluation in SoS software architectures?** *
   Note: remember that SOAR-E is a comprehensive solution to be fulfilled with the particularities of each SoS project.
   Mark only one oval.

   - ◯ Yes (it does not miss any relevant aspect)
   - ◯ Partially yes (it needs minimal additions)
   - ◯ Partially not (it needs critical additions)
   - ◯ No (it is not representative at all)

2. **How would you grade the set of activities defined in the SOAR-E practice?** *
   Mark only one oval.

   - ◯ The set of activities is acceptable
   - ◯ Less important activities are missing
   - ◯ Critical elements are missing
   - ◯ The set of activities does not accomplish its purpose at all

3. **Please point out any other element (e.g., alpha, work product, activity space, activity) that should be included as well as any other issue that you consider as relevant.**

Evaluating correctness

4. **How would you grade the activities of the SOAR-E practice in terms of correctness?** *
   Mark only one oval.

   - ◯ Correct (there are no wrong or misunderstood statements)
   - ◯ Partially correct (less important statements should be reviewed)
   - ◯ Partially incorrect (critical statements should be reviewed)
   - ◯ Totally incorrect (the amount of errors makes SOAR-E practice useless)

5. **Please point out any statement/concept that you consider as incorrect/misunderstood.**

Evaluating coherence

6. **How would you grade the activities of the SOAR-E practice in terms of coherence?** *
   Mark only one oval.

   - ◯ Partially coherent (less important elements are conflicting or incorrectly placed)
   - ◯ Coherent (there are no relevant conflicts or incorrectly placed elements)
   - ◯ Partially coherent (critical elements are conflicting or incorrectly placed)
   - ◯ Totally incoherent (the amount of conflicts and/or disorganization make the SOAR-E practice useless)

7. **Please point out any element (e.g., alpha, work product, activity space, activity) that you think incorrectly placed or conflicting.**

Evaluating usability

8. **How would you grade the usability of the SOAR-E practice description?** *
   Mark only one oval per row.

	1 (low)	2	3	4 (high)
Clear				
Well organized				

9. **Specific difficulties**
   Regarding your reading, point out any difficulty that you had on understanding the SOAR-E practice and its elements.

Extra comments and suggestions

10. **Please give your improvement sugestions to the SOAR-E practice.**