Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems

Flavio Oquendo(✉)

IRISA – UMR CNRS/Univ. Bretagne Sud, Vannes, France
flavio.oquendo@irisa.fr

Abstract. Software-intensive systems are often independently developed, operated, managed, and evolved. Progressively, communication networks enabled these independent systems to interact, yielding a new kind of complex system, i.e. a system that is itself composed of systems, the so-called System-of-Systems (SoS). By its very nature, SoS is evolutionarily developed and exhibits emergent behavior.

Actually, software architecture research has mainly focused on single systems, mostly large or very large distributed systems whose software architecture is described as design-time configurations of components linked together through connectors. However, it is well known that the restricted characteristics of single (even very large distributed) systems lead to architectural solutions (in terms of theories, languages, tools, and methods) that do not scale up to the case of systems-of-systems.

Indeed, novel architectural solutions are needed to handle the complexity of software-intensive systems-of-systems in particular regarding the software architecture challenges implied by evolutionary development and emergent behavior.

This paper presents the challenges facing software architecture research to address software-intensive systems-of-systems. It analyzes the discriminating characteristics of system-of-systems when compared with single systems from the software architecture perspective and focuses on recent advances in software architecture research to formally describe the architecture of software-intensive systems-of-systems.

Keywords: Software architecture · Software-intensive system-of-systems · Software architecture challenges · Research on formal architecture description · Formal behavioral modeling · Emergent behavior

1 Introduction

The complexity of software and the complexity of systems reliant on software have grown at a staggering rate. In particular, software-intensive systems have been rapidly evolved from being stand-alone systems in the past, to be part of networked systems in the present, to increasingly become systems-of-systems in the coming future [18].

B. Tekinerdogan et al. (Eds.): ECSA 2016, LNCS 9839, pp. 3–21, 2016.
DOI: 10.1007/978-3-319-48992-6_1

De facto, the pervasiveness of the communication networks increasingly has made possible to interconnect software-intensive systems that were independently developed, operated, managed, and evolved, yielding a new kind of complex system, i.e. a system that is itself composed of systems, the so-called System-of-Systems (SoS) [23].

SoSs are evolutionary developed from independent systems to achieve missions not possible to be accomplished by a system alone. They are architected to exhibit emergent behavior [20], i.e. behaviors that stem from the interactions among independent constituent systems which cannot be deduced from the behaviors of the constituent systems themselves. It means that the behavior of the whole SoS cannot be predicted through analysis only of the behaviors of its constituent systems, or stated simply: "the behavior of the whole SoS is more than the sum of the behaviors of its constituent systems".

This is the case of SoSs found in different areas as diverse as aeronautics, automotive, energy, healthcare, manufacturing, and transportation [10, 22]; and application domains that address societal needs as e.g. environmental monitoring, emergency coordination, traffic control, smart grids, and smart cities [18]. Moreover, ubiquitous platforms such as the Internet of Things (generalizing wireless sensor/actuator networks in the Cloud) and nascent classes of SoSs such as Cyber-Physical ones are accelerating the deployment of software-intensive SoSs, i.e. SoSs where software contributes essential influences to their design, construction, deployment, and evolution [17], as depicted in Fig. 1.

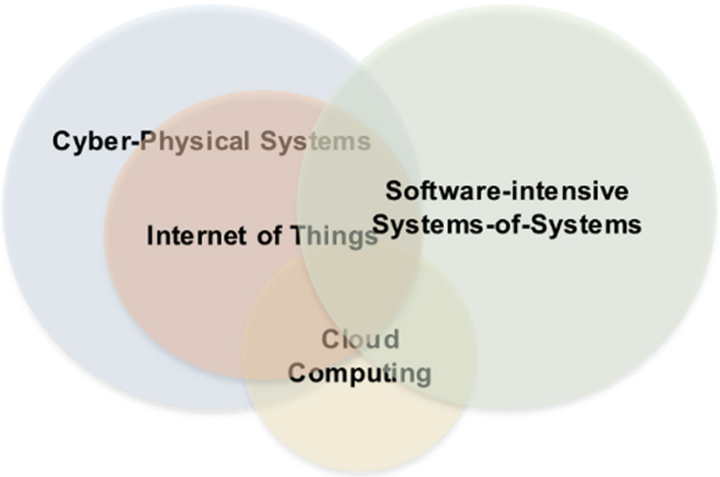

Fig. 1. SoSs and related enabling platforms

Additionally, besides SoSs that are developed in specific localities, e.g. a smart-city, some SoSs are being developed with a world-wide scope, e.g. the Global Earth Observation SoS (GEOSS) [13] that links Earth observation resources world-wide targeting missions for biodiversity and ecosystem sustainability.

It is worth highlighting that complexity is intrinsically associated to SoSs by its very nature that implies emergent behaviors. Note also that in SoSs, missions are achieved through emergent behaviors drawn from the local interactions among constituent systems.

Hence, complexity poses the need for separation of concerns between architecture and engineering [23]: (i) architecture focuses on designing and reasoning about interactions of parts and their emergent properties; (ii) engineering focuses on designing and constructing such parts and integrating them as architected.

Definitely, a key facet of the design of any software-intensive system is its architecture, i.e. the fundamental organization of a system embodied in its constituents, their relationships to each other, and to the environment, and the principles guiding its design and evolution, as defined in the ISO/IEC/IEEE Standard 42010 [17].

In particular, the ISO/IEC/IEEE Standard 42010 states the importance of having software architecture description as an essential first-class citizen artifact (similarly to the case of other architecture fields, e.g. civil architecture and naval architecture). Thereby, Architecture Description Languages (ADLs) are needed to express architecture descriptions. Note that we use the term ADL in the wider meaning defined by the ISO/IEC/IEEE Standard 42010: any form of expression enabling architecture descriptions.

Conceiving ADLs has been the subject of intensive research in the last 20 years resulting in the definition of several ADLs for modeling initially static architectures and then dynamic architectures of (often large or very large) single systems [24, 25, 35]. However, none of these ADLs have the expressive power to describe the architecture of a software-intensive SoS [14, 21].

It is worth to recall here that software intensive systems-of-systems are in general critical and very often safety-critical what is not the case of most of the software-only systems that were the subject of the research on software architecture description. It is also worth noting that among the ADLs proposed in the literature [24], the one that had a widely industrial adoption is AADL, the SAE Standard AS5506 [37], dedicated to safety-critical software-intensive systems in the avionics and automotive domains, where the architecture has a key role to satisfy safety-related requirements.

Therefore, to address the research challenges brought by SoSs, a novel ADL is needed for enabling the formal architecture description of software-intensive SoSs, in particular for the case of critical software-intensive SoSs [14]. This ADL must provide the expressive power to address the challenges raised by SoSs especially regarding correctness properties related to evolutionary development and emergent behavior. SoSs have indeed evolutionary architectures. Moreover, it must enable to prescribe SoS architectures abstractly at design-time without knowing which will be the actual concrete systems that will participate in the SoS at run-time.

The remainder of this paper is organized as follows. Section 2 discusses the notion of software-intensive SoS. Section 3 presents the main roadmaps for SoS research. Section 4 analyzes the distinctive characteristics of SoSs and their implications in terms of software architecture challenges. Section 5 discusses and introduces the essential SoS architectural concepts. Section 6 introduces emerging research on novel formal approaches for describing SoS architectures, focusing on SosADL, an emerging formal ADL for SoS. In Sect. 7, we present a case study, excerpt from a real SoS project,

summarizing lessons learnt from the application of SosADL in practice. In Sect. 8, we present related work on SoS architecture description. To conclude we summarize, in Sect. 9, the main contributions of this paper and outline future work.

2 The Notion of System-of-Systems

The notion of system and the related notion of software-intensive system are well known and defined in the ISO/IEC/IEEE Standard 42010. A system is a combination of components organized to accomplish a specific behavior for achieving a mission. Hence, a system exists to fulfill a mission in an environment. A software-intensive system is a system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole [17].

The notion of software-intensive system-of-systems is however relatively new, being the result of the ubiquity of computation and pervasiveness of communication networks.

A System-of-Systems (SoS, as stated) is a combination of constituents, which are themselves systems, that forms a more complex system to fulfill a mission, i.e. this composition forms a larger system that performs a mission not performable by one of the constituent systems alone [23], i.e. it creates emergent behavior.

For intuitively distinguishing an SoS from a single system, it is worth to recall that every constituent system of an SoS fulfills its own mission in its own right, and continues to operate to fulfill its mission during its participation in the SoS as well as when disassembled from the encompassing SoS.

For instance, an airport, e.g. Paris-Charles-de-Gaulle, is an SoS, but an airplane alone, e.g. an Airbus A380, is not. Indeed, if an airplane is disassembled in components, no component is a system in itself. In the case of an airport, the constituent systems are independent systems that will continue to operate, e.g. the air traffic control and the airlines, even if the airport is disassembled in its constituents.

Operationally, an airport is an SoS spanning multiple organizations, categorized into major facilities: (i) passenger, (ii) cargo, and (iii) aircraft departure, transfer and arrival. Each facility is shared and operated by different organizations, including air navigation services providers, ground handling, catering, airlines, various supporting units and the airport operator itself. The airport facilities are geographically distributed, managed by independent systems, and fall under multiple legal jurisdictions in regard to occupational health and safety, customs, quarantine, and security. For the airport to operate, these numerous constituent systems work together to create the emergent behavior that fulfill the airport mission.

As a software-intensive SoS, an airport is composed of independent systems that enable passengers, cargo, airplanes, information and services to be at the right place at the right time via the seamless collaboration of these constituent systems, from check-in, to security, to flight information displays, to baggage, to boarding, streamlining airport operations.

It is worth noting that the level of decentralization in the control of the constituent systems of an SoS varies, e.g. regarding airports, the level of subordination in a military airport and in a civil airport are very different. It is also worth noting that in some cases

the SoS has a central management, as it is the case of civil and military airports, and in others do not, as it is the case e.g. in a metroplex, i.e. the set of airports in close proximity sharing the airspace serving a city.

SoSs may be classified in four categories according to the levels of subordination and awareness of the constituent systems on the SoS [8, 23]:

- Directed SoS: an SoS that is centrally managed and which constituent systems have been especially developed or acquired to fit specific purposes in the SoS – the constituent systems maintain the ability to operate independently, but their actual operation is subordinated to the central SoS management (i.e. the management system of the coalition of constituent systems); for instance, a military airport.
- Acknowledged SoS: an SoS that is centrally managed and which constituent systems operate under loose subordination – the constituent systems retain their independent ownership; for instance, a civil airport.
- Collaborative SoS: an SoS in which there is no central management and constituent systems voluntarily agree to fulfill a central mission – the constituent systems operate under the policies set by the SoS; for instance, a metroplex.
- Virtual SoS: an SoS in which there is no central management or centrally agreed mission – the constituent systems operate under local, possibly shared, policies; for instance, the airports of a continent such as Europe.

These different categories of SoSs bring the need to architect SoSs where local interactions of constituent systems influence the global desired behavior of the SoS taking into account the levels of subordination and awareness of the constituent systems on the SoS.

3 Roadmaps for the Research on Systems-of-Systems

Currently, the research on software-intensive SoSs is still in its infancy [14, 21]. In addition, SoSs are developed mostly in a case-by-case basis, not addressing neither cross-cutting concerns nor common foundations across SoS application domains [7].

Actually, the relevance and timeliness of progressing the state of the research for developing critical software-intensive SoSs from now on are highlighted in several roadmaps targeting year 2020 and beyond [9, 11, 41]. The needs for research on software-intensive SoSs have been addressed in different studies carried out by the initiative of the European Commission in the H2020 Program, as part of the European Digital Agenda [7].

More precisely, in 2014, two roadmaps for SoSs were proposed (supported by the European Commission) issued by the CSAs ROAD2SoS (Development of strategic research and engineering roadmaps in Systems-of-Systems) [9] and T-Area-SoS (Transatlantic research and education agenda in Systems-of-Systems) [11]. In 2015, the CSA CPSoS [16] presented a research agenda for developing cyber-physical SoSs.

All these roadmaps show the importance of progressing from the current situation, where software-intensive SoSs are basically developed in ad-hoc ways in specific application sectors, to a scientific approach providing rigorous theories, languages, tools, and methods for mastering the complexity of SoSs in general (transversally to application domains).

These roadmaps highlight that now is the right time to initiate research efforts on SoS to pave the way for developing critical software-intensive SoSs in particular regarding architectural solutions for trustworthily harnessing emergent behaviors to master the complexity of SoSs.

Overall, the long-term grand challenge raised by critical software-intensive SoSs calls for a novel paradigm and novel scientific approaches for specifying, architecting, analyzing, constructing, and evolving SoSs deployed in unpredictable open environments while assuring their continuous correctness.

In Europe, this effort started more intensively in 2010 when the European Commission launched a first Call for Research Projects addressing SoS as the main objective of study; in 2013 another Call for Projects had again SoS as an objective and in 2016 the third was opened. The projects funded in the first European Call have now ended: COMPASS (Comprehensive modelling for advanced Systems-of-Systems, from Oct. 2010 to Sept. 2014) [3] and DANSE (Designing for adaptability and evolution in System-of-Systems engineering, from Nov. 2010 to Oct. 2014) [4]. The projects of the second Call started in 2014 [7]: AMADEOS (Architecture for multi-criticality agile dependable evolutionary open System-of-Systems), DYMASOS (Dynamic management of coupled Systems-of-Systems), and LOCAL4GLOBAL (System-of-Systems that act locally for optimizing globally).

Regarding other parts of the world, in the USA, different research programs specifically targets SoS, in particular in the Software Engineering Institute [42] and Sandia National Laboratories [41] among others. In these programs, it is interesting to pinpoint the different research actions that have evaluated current technologies developed for single systems in terms of suitability/limitation for architecting and engineering SoSs. In addition, prospective studies have highlighted the overwhelming complexity of ultra-large-scale SoSs [12].

Note also that different industrial studies and studies from the industrial viewpoint have highlighted the importance, relevance and timeliness of software-intensive SoSs [10, 22].

4 Software Architecture Challenges in Systems-of-Systems

Due to its inherent complex nature, architecting SoSs is a grand research challenge, in particular for the case of critical software-intensive SoSs.

Precisely, an SoS is defined as a system constituted of systems having the following five intrinsic characteristics [23]:

- Operational independence: every constituent system of an SoS operate independently from each other for fulfilling its own mission;
- Managerial independence: every constituent system of an SoS is managed independently, and may decide to evolve in ways that were not foreseen when they were originally combined;
- Geographical distribution: the constituent systems of an SoS are physically decoupled (in the sense that only information can be transmitted between constituent systems, nor mass neither energy);

- Evolutionary development: as a consequence of the independence of the constituent systems, an SoS as a whole may evolve over time to respond to changes in its constituent systems and operational environment; moreover, the constituent systems are only partially known at design-time;
- Emergent behaviors: in an SoS, new behaviors emerge from the local interaction of its constituent systems (i.e. an emergent behavior that cannot be performed by a constituent system alone); furthermore, these emergent behaviors may be ephemeral because the systems composing the SoS evolve independently, which may impact their availability.

The combination of these five defining characteristics turns the architecture of SoSs to be naturally highly evolvable, frequently changing at run-time in unpredictable ways. SoSs have thereby evolutionary architectures (with the meaning that they dynamically adapt or evolve at run-time subject to the evolutionary development of the SoS).

Much work has addressed the issue of describing software architecture (in the sense of architecture of software-only systems as well as software-intensive systems). Most of the work carried out addressed static architectures (architectures which do not change at run-time) and some tackled dynamic architectures (architectures which may change at run-time). Therefore, we must pose the question whether these ADLs provide enough expressive power for describing SoS evolutionary architectures.

To address this question we will first analyze how and why SoSs are different from single systems, then analyze what are the implications of these distinctive characteristics for SoS architecture.

A single system and an SoS are both systems and as such they are developed with the purpose of fulfilling a mission. In both cases, they are themselves constituted of parts that architected together will provide the capabilities of the system as a whole to achieve the specified missions. The distinctive nature of an SoS when compared to a single system derives from its five defining characteristics.

To well understand what in an SoS is different from a single system, it is worth recalling that we are addressing software-intensive systems (not software-only systems) in this comparison. It is also worth noting that, in Systems Engineering, it is well known that the formalisms and technologies for single systems are not suitable to SoSs from a long time [23], SoS having its first dedicated international conference organized 10 years ago: the IEEE International Conference on System-of-Systems Engineering (SoSE), being in its 11th edition in 2016. In particular, the limitations of employing theories, languages, tools, and methods conceived for the architecture of single systems to the architecture of SoSs are well recognized and triggered a new thread of research [18–20, 23].

Let us now enumerate in Table 1 the key differences between both kind of systems (i.e. single systems and SoSs) by analyzing, for each distinctive characteristic, the nature of the constituent parts and the nature of the relationship between the whole and its constituent parts (see [8] for a deeper survey on the distinctive characteristics that differentiate systems-of-systems from single systems and their different graduations).

Table 1. Differences between single system and system-of-systems

Characteristic	Single system	System-of-systems (SoS)
Nature of the constituent parts		
Operational independence of constituents	• *None*: constituent components have no operational independence, i.e. they operate as designed to provide its functionality	• *Partial or Total*: constituent systems have operational independence, i.e. they operate independently (at least partially) from the SoS to fulfill its own mission
Managerial independence of constituents	• *None*: constituent components have no managerial independence, i.e. all decisions are made at the system level	• *Partial or Total*: constituent systems have managerial independence (at least partially) from the SoS to fulfill its own mission, in particular they may decide to evolve in ways that were not foreseen when they were originally combined in the SoS
Geographical distribution of constituents	• *None or Partial*: constituent components may be physically coupled (in the sense that mass and energy can be transmitted between constituent components)	• *Total*: constituent systems of an SoS are physically decoupled, i.e. only information can be transmitted between constituent systems, never mass nor energy
Nature of the relationships between the whole and its constituent parts		
Initial development of system	• *Total*: a single system is architected to meet a mission using constituent components developed or acquired to fit the mission at design-time	• *None or Partial*: an SoS is architected to meet a mission, but very often not knowing the constituent systems that will support the mission at run-time
Evolutionary development of system	• *None or Partial*: after initial development, constituent components may evolve under the control of the system at run-time	• *Total*: an SoS has no (or at most partial) control on how the constituent systems may evolve, in particular as their missions may not be aligned with the SoS mission
Emergent behavior of system	• *None*: constituent components have predictable behaviors from the system perspective as well as system behaviors are predictable from the behavior of the constituent components	• *Partial or Total*: even if the behaviors of constituent systems are predictable, their independence turns the SoS behavior unpredictable from the SoS perspective producing also emergent behaviors from local interactions

Undoubtedly, the main difference between an SoS and a single system is the nature of their constituent systems, specifically their level of independence, and the exhibition of emergent behavior.

Complexity is thereby innate to SoSs as they inherently exhibit emergent behavior: in SoSs, missions are achieved through emergent behavior drawn from the local interaction among constituent systems. In fact, an SoS is conceived to create desired emergent behaviors for fulfilling specific missions and may, by side effect, create undesirable behaviors possibly violating safety, which needs to be avoided. A further complicating factor is that these behaviors may be ephemeral because the systems constituting the SoS evolve independently, which may impact their availability. Additionally, the environment in which an SoS operates is generally known only partially at design-time and almost always is too unpredictable to be summarized within a fixed set of specifications (thereby there will inevitably be novel situations, possibly violating safety, to deal with at run-time).

Overall, major research challenges raised by software-intensive SoSs are fundamentally architectural: they are about how to organize the local interactions among constituent systems to enable the emergence of SoS-wide behaviors and properties derived from local behaviors and properties (by acting only on their interactions, without being able to act in the constituent systems themselves) subject to evolutions that are not controlled by the SoS due to the independent nature of constituents.

Therefore, enabling to describe SoS architectures is a grand research challenge.

5 Enhancing Architectural Concepts for SoS

Remember that a software architecture is defined to be the fundamental organization of a system embodied in its constituents, their relationships to each other, and to the environment, and the principles guiding its design and evolution [17]. In the architecture description of single systems, the core architectural concepts are the one of "component" to represent the constituents, the one of "connector" to represent the interactions among constituents, and the one of "configuration" to represent their composition.

As the restricted meaning of these concepts do not cope with the nature of SoS architectures, it is important to define novel concepts for describing SoS architectures as well as to name the new terms aligned with the SoS terminology.

These SoS concepts are the ones of "constituent system" of an SoS, "mediator" among constituent systems of an SoS, and "coalition" of mediated constituent systems of an SoS.

In addition, SoS architectures must be described in abstract terms at design-time (recall that concrete systems that will become constituents of the SoS are generally not known at design-time). The defined abstract architecture will then be evolutionarily concretized at run-time, by identifying and incorporating concrete systems.

In Table 2 we summarize these concepts and indicate how they are different and extends the ones of single systems.

Table 2. SoS architectural concepts

SoS architectural concepts	
Constituent system	Systems are the constituents of an SoS: a system has its own mission, is operationally independent, managerially independent, and may independently evolve. The concept of constituent system focuses on the capabilities to deliver system functionalities • Note that the concept of constituent system subsumes the concept of component in single systems: a component can be perceived as a "constituent system that is totally subordinated", oppositely to constituent systems in general that are, by definition, generally independent • Constituent systems exist independently of the SoS
Mediator	Mediators mediate the interaction of constituent systems of an SoS: a mediator has the purpose to achieve a specific emergent behavior by mediating the interaction among different constituent systems • Note that the concept of mediator in SosADL subsumes the concept of connector in single systems: mediators have a coordination role, while connectors have basically communication roles • Note that mediators are both operationally and managerially dependent of the SoS, and evolves under the control of the SoS for achieving emergent behaviors (an SoS has total control on mediators: it can create, evolve and destroy mediators at run-time)
Coalition	A coalition constitutes a temporary alliance for coordinated action among constituent systems connected via mediators (it is dynamically formed to fulfill the SoS mission through emergent behaviors) • Coalitions can be recomposed in different ways or with different systems in order to fulfil a specified SoS mission • Coalitions are declared by expressing SoS policies to select and bind existing constituent systems using mediators created by the SoS itself • Note that the SoS totally controls its mediators, but not at all its constituent systems, which are independent from the SoS
Design-time use of the architectural concepts	
Abstract architecture defined by intention	By the nature of SoSs, concrete systems that will actually participate are generally not known at design-time, therefore the SoS architecture needs to be defined abstractly, i.e. specifying constraints to select possible constituent systems and contracts to be fulfilled by constituent systems in mediators • Note that this is the opposite of the case of single systems where almost always the architecture is complete decided at design-time including all its concrete components

(*continued*)

Table 2. (*continued*)

	• Note that the concept of abstract architecture is different and does not have the same purpose as the concept of architectural style or pattern. Both, style and pattern, are codifications of design decisions used as architectural knowledge for designing abstract or concrete architectures. An abstract architecture is the expression of all possible valid concrete architectures in declarative terms. A concrete architecture is the actual architecture that operates at run-time
Run-time use of the architectural concepts	
Concrete architecture defined by extension	Once an SoS is initiated, concrete systems coping with the specified system abstractions needs to be identified to create concrete coalitions at run-time with the assistance of mediators • Note that a concrete system may enter or leave the SoS at run-time by its own decision (the SoS has no control on concrete systems); mediators oppositely are dynamically created and evolve under the control of the SoS

Therefore, in an SoS architecture description:

- Constituent systems are SoS architectural elements defined by intention (declaratively in terms of abstract systems) and selected at run-time (concretized).
- Mediators are SoS architectural elements defined by intention (declaratively in terms of abstract mediators) and created at run-time (concretized by the SoS) to achieve a goal, part of an encompassing mission (note that its architectural role is to mediate the interaction of constituent systems for creating emergent behavior).
- Coalitions are SoS architectural compositions of mediated constituent systems, defined by intention (declaratively in terms of possible systems and mediators and policies for their on-the-fly compositions) and evolutionarily created at run-time (concretized) to achieve an SoS mission in an operational environment.

6 Emerging Research on SoS Architecture Description

To address the research challenge of formally describing SoS architectures, in particular regarding its evolutionary development and the modeling of SoS emergent behaviors, we have started in 2013 a research project in collaboration with industrial SoS architects.

From this research emerged a novel architectural solution in terms of formal languages and supporting tools, especially conceived for formally modeling and analyzing the architecture of software-intensive SoSs. This novel solution for SoS architecture brings the following contributions to the state-of-the-art:

- A novel formal foundation for modeling SoS architectures: we conceived a novel process calculus in the family of the π-Calculus [26], named π-Calculus for SoS (for details on the π-Calculus for SoS see [31] in the proceedings of the 2016 IEEE SoS

Engineering Conference (SoSE 2016) which presents its formal definition and operational semantics).

- A novel formal architectural language embodying the SoS architectural concepts of constituent system, mediator, and coalition: grounded on the π-Calculus for SoS, we conceived a novel ADL based on the separation of concerns between architectural abstractions at design-time and architectural concretions at run-time (for details see [30] in the proceedings of the 2016 IEEE SoS Engineering Conference (SoSE 2016) which presents the concepts and notation of this novel ADL, named SosADL).
- A novel temporal logic for expressing correctness properties of highly dynamic software architectures (including SoS architectures) and verifying these properties with statistical model checking: we conceived a novel temporal logic, named DynBLTL, for supporting analysis of SoS architectures (for details on this temporal logic see [36] in the proceedings of the 2016 International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2016)); in addition we developed a novel statistical model checking method for verifying properties expressed on DynBLTL on architecture descriptions based on the π-Calculus (for details see [2] in these proceedings of ECSA 2016).
- A novel formalization for checking the architectural feasibility of SoS abstract architecture descriptions and for creating concrete architectures from SoS abstract architectures: it supports automated creation of concrete architectures from an abstract architecture given selected concrete constituent systems as well as supports the evolution of concrete architectures by automated constraint solving mechanisms (for details see [15] in the proceedings of the 2016 IEEE SoS Engineering Conference (SoSE 2016) which presents this novel formal system mechanizing the solving of concurrent constraints of SosADL).
- A novel approach for modeling SoS missions in terms of goals relating them to mediators and required SoS emergent behaviors (for details see [38] in the proceedings of the 2015 IEEE SoS Engineering Conference (SoSE 2015) which presents the SoS mission description notation and the supporting tool).
- The field validation of SosADL and its underlying π-Calculus for SoS drew from a real pilot project and related case study of a Flood Monitoring and Emergency Response SoS, summarized in the next section (for details see [32] in the proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016)).

Additionally, we have developed an SoS Architecture Development Environment (SosADE) for supporting the architecture-centric formal evolutionary development of SoSs using SosADL and associated analysis languages and tools. This toolset provides a model-driven architecture development environment where the SosADL meta-model is transformed to different analysis meta-models and converted to input languages of analysis tools, e.g. Kodkod for concurrent constraint solving, UPPAAL for model checking, DEVS for simulation, and PLASMA for statistical model checking.

7 Lessons Learnt from Applying SosADL in a Case Study

Formally defined in terms of the π-Calculus SoS, SosADL provides architectural concepts and notation for describing SoS architectures. The notation of SosADL is in particular presented in [30] and formally defined in [31]. Hereafter we will focus on its expressiveness as an ADL for describing SoS architectures by focusing in an excerpt of a case study carried out for architecting an SoS for Flood Monitoring and Emergency Response [32].

Flood Monitoring and Emergency Response SoSs address the problem of flash floods, which raise critical harms in different countries over rainy seasons. This becomes particularly critical in cities that are crossed by rivers such as the city of Sao Carlos, SP, Brazil, crossed by the Monjolinho river as shown in Fig. 2.

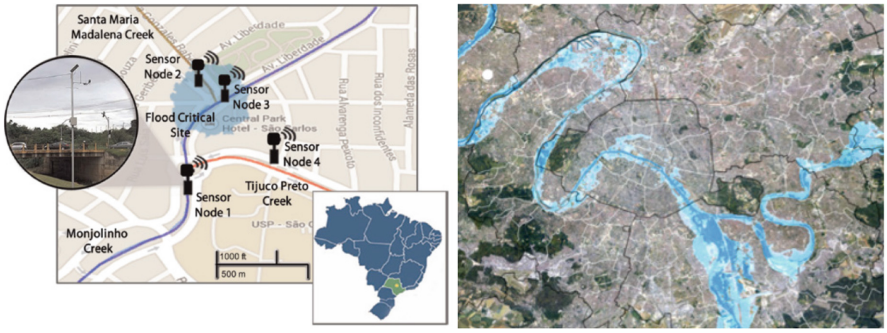

Fig. 2. Monjolinho river crossing the city of Sao Carlos with deployed wireless river sensors

This Flood Monitoring and Emergency Response SoSs has the five defining characteristics of an SoS. Let us now briefly present this in vivo field study in Table 3.

The aim of this field study was to assess the fitness for purpose and the usefulness of SosADL to support the architectural design of real-scale SoSs.

The result of the assessment based on this pilot project shown that the SosADL met the requirements for describing SoS architectures. As expected, using a formal ADL compels the SoS architects to study different architectural alternatives and take key architectural decisions based on SoS architecture analyses.

Learning SosADL in its basic form was quite straightforward; however, using the advanced features of the language needed interactions with the SosADL expert group. The SoS architecture editor and simulator were in practice the main tools to learn and use SosADL and the SoS architecture model finder and model checker were the key tools to show the added value of formally describing SoS architectures.

In fact, a key identified benefit of using SosADL was the ability, by its formal foundation, to validate and verify the studied SoS architectures very early in the application lifecycle with respect to the SoS correctness properties, in particular taking into account emergent behavior in a critical set as the one of flash flood.

Table 3. Field study of SosADL on a flood monitoring and emergency response SoS

Field study for architecting a WSN-based flood monitoring and emergency response SoS	
Purpose	The aim of this field study of a Flood Monitoring and Emergency Response SoS was to assess the fitness for purpose and the usefulness of SosADL and underlying formal foundation to support the architectural design of real-scale SoSs
Stakeholders	The SoS stakeholder is the DAEE (Sao Paulo's Water and Electricity Department), a government organization of the State of Sao Paulo, Brazil, responsible for managing water resources, including flood monitoring of urban rivers. Stakeholders of the constituent systems are the different city councils crossed by the Monjolinho river and the policy and fire departments of the city of Sao Carlos that own Unmanned Aerial Vehicles (UAVs) and have cars equipped with Vehicular Ad-hoc Networks (VANETs). The population, by downloading an App from the DAEE department, are involved as target of the alert actions. They may also register for getting alert messages by SMS
Mission	The mission of this SoS is to monitor potential flash floods and to handle related emergencies
Emergent behaviors	In order to fulfil its mission, this monitoring SoS needs to create and maintain an emergent behavior where sensor nodes (each including a sensor mote and an analog depth sensor) and UAVs (each including communication devices) will coordinate to enable an effective monitoring of the river and once a risk of flood is detected, to prepare the emergence response for warning vehicles with VANETs and drivers with smartphones approaching the flood area as well as inhabitants that live in potential flooding zones. Resilience of the SoS, even in case of failure of sensors and UAVs need to be managed as well as its operation in an energy-efficient way. The emergence response involves warning the policy and fire departments as well
SoS architecture	The architecture of this Flood Monitoring and Emergency Response SoS was described in SosADL as a Collaborative SoS having a self-organizing architecture based on mediators for connecting sensors and forming multihop ad-hoc networks for both flood monitoring and emergency response. The designed SoS architecture allows for continuous connections and reconfigurations around broken or blocked paths, supported by the SoS evolutionary architecture with possible participation of UAVs and VANETs (see [32] for details on the SoS architecture description)

The experimentation and the corresponding assessment have shown that SosADL and its toolset, SosADE, are de facto suitable for formally describing and analyzing real-scale SoS architectures.

8 Related Work on SoS Formal Architecture Description

Software-intensive SoS is a nascent domain. According to [14], ca. 75 % of the publications addressing software-intensive SoSs appeared in the last 5 years and ca. 90 % in the last 10 years.

We carried out a Systematic Literature Review (SLR)[1] to establish the state-of-the-art on architecture description of SoSs [14], which permitted to collect, evaluate, and summarize the research related to the following question: *Which modeling languages (including ADLs) have been used to describe SoS architectures?*

As a result of the SLR [14], the following modeling languages have been identified as the main ones used for SoS architecture description: UML (semi-formal) [40], SysML (semi-formal) [39], and CML [16] (formal). These findings are compatible with the findings of another SLR see [21] conducted independently.

More specifically, SysML was the baseline of two European FP7 projects (COMPASS [3] and DANSE [4]) for which they developed extensions for SoSs.

DANSE did not develop an ADL, but used SysML to semi-formally describe executable architectures that are then tested against interface contracts. The tests are applied to the traces obtained by executing architectures, against interface contracts expressed on GCSL (Goal Contract Specification Language) [4].

COMPASS developed a formal approach, in contrast to DANSE that extended a semi-formal one. In COMPASS, CML [28] was specifically designed for SoS modeling and analysis.

CML is not an ADL. It is a contract-based formal specification language to complement SysML: SysML is used to model the constituent systems and interfaces among them in an SoS and CML is used to enrich these specifications with interface contracts. A CML model is defined as a collection of process definitions (based on CSP/Circus [28]), which encapsulate state and operations written in VDM (Vienna Development Method) as well as interactions via synchronous communications.

CML is a low-level formal language, of which a key drawback (stated by their authors) is that SysML models when mapped to CML produce huge unintelligible descriptions (it was one of the lessons learned from COMPASS [28]).

In contrast to CML, SosADL enables the formal description of an SoS architecture as a whole being a full ADL according to the criteria of ISO/IEC/IEEE Standard 42010 [17], while CML is not, focusing only on contracts of interactions. Moreover, regarding SoS behavioral modeling, SosADL subsumes CML in terms of expressive power by its mathematical foundation based on the π-Calculus for SoS [31], subsuming CSP/Circus.

It is worth to recall here that SosADL as a formal architectural language follows previous work that focused on formalisms for describing software architectures which are dynamic [29] and self-evolving [27] in the scope of single systems. In particular, achievements of the ArchWare European Project [33] (FP5 ICT Program) were presented in a keynote [27] ten years ago in the 1st ECSA concentrating on an active architecture framework for supporting self-evolving software-intensive (single) systems architecturally described in π-ADL [29, 33] and currently supported by modern concurrent languages [1] in the Cloud.

Complementary to ADLs, software architecture models, patterns, and styles as well as software architecture-based frameworks have been studied for different kinds of

[1] We conducted automatic searches on the major publication databases related to the SoS domain (IEEE Xplore, ISI Web of Science, Science Direct, Scopus, SpringerLink, and ACM Digital Library), after having the defined the SLR protocol (see [14] for details on the SLR).

single systems exhibiting dynamic architectures, especially for autonomic systems, self-adaptive systems, self-organizing systems and more generally self-* systems. However, these different works have not targeted SoS and in particular have not at all addressed the key issue of emergent behavior as they have de facto limited their scope to single systems, e.g. [5].

Another thread of related work on SoSs is the one of implementation platforms. For the particular case of homogeneous constituent systems, a new generation of component frameworks and modeling languages have been designed to develop a specific class of SoSs, the so-called "ensembles" (an SoS that is only composed of homogeneous systems), e.g. DEECo (Dependable Ensembles of Emerging Components) [44] and SCEL (Service Component Ensemble Language) [44].

It is worth noting that "ensembles" denote a specific implementation style, which may be used to develop the implementation of SoS architectures designed with an SosADL. In this case, SoS homogeneous architectures described and analyzed with SosADL can be transformed to implementation models using SCEL or DEECo.

In summary, based on the study of the state-of-the-art carried out through the SLR, SosADL is positioned as a pioneering ADL having the expressive power to formally describe SoS architectures, no existing ADL being able to express these evolutionary architectures [14, 21]. Regarding detailed design and implementation in specific styles, it is complementary to technologies developed for e.g. "ensembles" as well as more generally to service-oriented architectural styles [43] applied to SoS implementation.

9 Conclusion and Future Work

This paper introduced the notion of software-intensive SoS, raised key software architecture challenges in particular related to SoS architecture description and briefly surveyed emerging research on ADLs for SoS addressing these challenges based on a paradigm shift from single systems to systems-of-systems.

Oppositely to single systems, SoSs exhibit emergent behavior. Hence, whether the behavior of a single system can be understood as the sum of the behaviors of its components, in SoSs, this reductionism fails: an SoS behaves in ways that cannot be predicted from analyzing exclusively its individual constituents. In addition, SoS is characterized by evolutionary development enabling to maintain emergent behavior for sustaining SoS missions.

Software-intensive SoS has become a hotspot in the last 5 years, from both the research and industry viewpoints. Indeed, various aspects of our lives and livelihoods have progressively become overly dependent on some sort of software-intensive SoS.

If SoS is a field well established in Systems Engineering and SoS architecture has been studied for two decades, it is yet in its infancy in Software Engineering and particularly in Software Architecture. Only 3 years ago, the first workshop on the architecture and engineering of software-intensive SoSs was launched: the first ACM Sigsoft/Sigplan International Workshop on Software Engineering for Systems-of-Systems was organized with ECSA 2013 [34] and since 2015 has been organized with ACM/IEEE ICSE, being in 2016 in its fourth edition. The first conference track dedicated to software-intensive SoS, SiSoS, will be organized only in 2017 at ACM SAC.

Beyond these initiatives of scientific forums, it is worth to highlight the increasing number of research initiatives targeting software-intensive SoSs such as the national research networks launched recently in France (CNRS GDR GPL/Research Network on Software-intensive Systems-of-Systems) and UK (VaVaS Research Network for the Verification and Validation of Autonomous SoSs), and the national programs been launched in different countries, e.g. Labex MS2T (Control of Technological Systems-of-Systems) and IRT SystemX (Engineering the Digital Systems of the Future) in France and the SoS Agenda initiative in Sweden.

These different initiatives are paving the way for the future software-intensive SoSs enabling to architect and engineer software-intensive SoSs in different application domains with guaranteed trustworthy properties [6], harnessing emergent behaviors for trustworthily achieving SoS missions in critical SoSs.

Regarding emergent research on SoS and more specifically on SosADL, future work is mainly related with the application of SosADL and its related languages and toolset in industrial-scale projects. They include joint work with DCNS for applying SosADL to architect naval SoSs, with IBM for applying SosADL to architect smart-farms in cooperative settings, and with SEGULA for applying SosADL to architect SoSs in the transport domain.

References

1. Cavalcante, E., Batista, T.V., Oquendo, F.: Supporting dynamic software architectures: from architectural description to implementation. In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), Montreal, Canada, pp. 31–40, May 2015
2. Cavalcante, E., Quilbeuf, J., Traonouez, L.M., Oquendo, F., Batista, T., Legay, A.: Statistical model checking of dynamic software architectures. In: Tekinerdogan, B., et al. (eds.) ECSA 2016. LNCS, vol. 9839, pp. 185–200. Springer, Heidelberg (2016)
3. COMPASS: Comprehensive Modelling for Advanced Systems of Systems. http://www.compass-research.eu
4. DANSE: Designing for Adaptability and Evolution in System-of-Systems Engineering. http://www.danse-ip.eu
5. Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: Lemos, R., Giese, H., Müller, Hausi, A., Shaw, M. (eds.). LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35813-5_1
6. ERCIM: Special Theme: Trustworthy Systems-of-Systems, ERCIM News, vol. 102, July 2015. http://ercim-news.ercim.eu/en102/
7. European Commission (EC) - Horizon 2020 Framework Program: H2020 Digital Agenda on Systems-of-Systems. https://ec.europa.eu/digital-agenda/en/system-systems
8. Firesmith, D.: Profiling systems using the defining characteristics of systems of systems (SoS), software engineering institute. SEI Technical report: CMU/SEI-2010-TN-001, 87 p., February 2010
9. FP7 CSA Road2SoS (Roadmaps to Systems-of-Systems Engineering) (2011–2013): Commonalities in SoS Applications Domains and Recommendations for Strategic Action. http://road2sos-project.eu/

10. FP7 CSA Road2SoS (Roadmaps to Systems-of-Systems Engineering): Survey on Industrial Needs and Benefits of SoS in Different SoS Domains: Multi-site Industrial Production Manufacturing, Multi-modal Traffic Control, Emergency and Crisis Management, Distributed Energy Generation and Smart Grids. http://road2sos-project.eu/
11. FP7 CSA T-AREA-SoS (Trans-Atlantic Research and Education Agenda on Systems-of-Systems) (2011–2013): Strategic Research Agenda on Systems-of-Systems Engineering. https://www.tareasos.eu/
12. Feiler, F., et al.: Ultra-Large-Scale Systems: The Software Challenge of the Future, Software Engineering Institute – SEI/CMU, 150 p., June 2006
13. GEO (Group on Earth Observations): Global Earth Observation System-of-Systems (GEOSS). http://www.earthobservations.org/geoss.php
14. Guessi, M., Nakagawa, E.Y., Oquendo, F.: A systematic literature review on the description of software architectures for systems-of-systems. In: Proceedings of the 30th ACM Symposium on Applied Computing (SAC), Salamanca, Spain, pp. 1–8, April 2015
15. Guessi, M., Oquendo, F., Nakagawa, E.Y.: Checking the architectural feasibility of systems-of-systems using formal descriptions. In: Proceedings of the 11th System-of-Systems Engineering Conference (SoSE), June 2016
16. H2020 CSA CPSoS (Roadmap for Cyber-Physical Systems-of-Systems) (2013–2016), Roadmap: Analysis of the State-of-the-Art and Future Challenges in Cyber-Physical Systems-of-Systems. http://www.cpsos.eu/
17. ISO/IEC/IEEE 42010:2011: Systems and Software Engineering – Architecture Description, 46 p., December 2011
18. Jamshidi, M.: System-of-Systems Engineering: Innovations for the 21st Century. Wiley, Hoboken (2009)
19. Jaradat, R.M., et al.: A histogram analysis for system-of-systems. Int. J. Syst.-Syst. Eng. **5** (3), 193–227 (2014)
20. Johnson, C.W.: Complexity in design and engineering. Reliab. Eng. Syst. Saf. **91**(12), 1475–1588 (2006)
21. Klein, J., van Vliet, H.: A systematic review of system-of-systems architecture research. In: Proceedings of the 9th International Conference on Quality of Software architectures (QoSA), Vancouver, Canada, pp. 13–22, June 2013
22. Korsten, P., Seider, C.: The World's 4 Trillion-Dollar Challenge: Using a System-of-Systems Approach to build a Smarter Planet, IBM, 20 p., January 2010. ibm.com/iibv
23. Maier, M.W.: Architecting principles for systems-of-systems. Syst. Eng. **1**(4), 267–284 (1998)
24. Malavolta, I., et al.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2013)
25. Medvidovic, N., Taylor, R.: A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. **26**(1), 70–93 (2000)
26. Milner, R.: Communicating and Mobile Systems: The π-Calculus, 174 p. Cambridge University Press, Cambridge (1999)
27. Morrison, R., Balasubramaniam, D., Oquendo, F., Warboys, B., Greenwood, R.M.: An active architecture approach to dynamic systems co-evolution. In: Oquendo, F. (ed.) ECSA 2007. LNCS, vol. 4758, pp. 2–10. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75132-8_2
28. Nielsen, C.B., et al.: Systems-of-systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput. Surv. **48**(2), 1–41 (2015)

29. Oquendo, F.: π-ADL: architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. ACM Sigsoft Softw. Eng. Not. **29**(3), 1–14 (2004)
30. Oquendo, F.: Formally describing the software architecture of systems-of-systems with SosADL. In: Proceedings of the 11th IEEE System-of-Systems Engineering Conference (SoSE), June 2016
31. Oquendo, F.: π-calculus for SoS: a foundation for formally describing software-intensive systems-of-systems. In: Proceedings of the 11th IEEE System-of-Systems Engineering Conference (SoSE), June 2016
32. Oquendo, F.: Case study on formally describing the architecture of a software-intensive system-of-systems with SosADL. In: Proceedings of 15th IEEE International Conference on Systems, Man, and Cybernetics (SMC), October 2016
33. Oquendo, F., Warboys, B., Morrison, R., Dindeleux, R., Gallo, F., Garavel, H., Occhipinti, C.: ArchWare: architecting evolvable software. In: Oquendo, F., Warboys, Brian, C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 257–271. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24769-2_23
34. Oquendo, F., et al.: Proceedings of the 1st ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS), Montpellier, France, July 2013
35. Ozkaya, M., Kloukinas, C.: "Are we there yet? Analyzing architecture description languages for formal analysis, usability, and realizability. In: Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Santander, Spain, pp. 177–184, September 2013
36. Quilbeuf, J., Cavalcante, E., Traonouez, L.-M., Oquendo, F., Batista, T., Legay, A.: A logic for the statistical model checking of dynamic software architectures. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 806–820. Springer, Heidelberg (2016). doi:10.1007/978-3-319-47166-2_56
37. SAE Standard AS5506-2012: Architecture Analysis & Design Language (AADL), 398 p., September 2012
38. Silva, E., Batista, T., Oquendo, F.: A mission-oriented approach for designing system-of-systems. In: Proceedings of the 10th IEEE System-of-Systems Engineering Conference (SoSE), pp. 346–351, May 2015
39. SysML: Systems Modeling Language. http://www.omg.org/spec/SysML
40. UML: Unified Modeling Language. http://www.omg.org/spec/UML
41. US Sandia National Laboratories, Roadmap: Roadmap for the Complex Adaptive Systems-of-Systems (CASoS) Engineering Initiative. http://www.sandia.gov/
42. US Software Engineering Institute/Carnegie Mellon University: System-of-Systems Program. http://www.sei.cmu.edu/sos/
43. Wirsing, M., Hölzl, M.: Rigorous Software Engineering for Service-Oriented Systems, 748 p. Springer, Heidelberg (2015)
44. Wirsing, M., et al.: Software Engineering for Collective Autonomic Systems, 537 p. Springer, Heidelberg (2015)