Abstract—Nowadays, complex software-intensive systems have resulted from the integration of heterogeneous independent systems, thus leading to a new class of systems called Systems-of-Systems (SoS). As in any system, SoS architectures have been regarded as an important element for determining their success. However, the state of the art reveals shortcomings that contribute to compromise the quality of these systems, as their inherent characteristics (such as emergent behavior and evolutionary development) are often not properly addressed. In this context, this PhD research aims at investigating how SoS software architectures can be used to model and evolve these systems. As main contribution, an architecture-centric approach for developing software-intensive SoS with focus on the formal specification and dynamic reconfiguration of their architectures is proposed. Such an approach mainly intends to contribute to fill some of the relevant existing gaps regarding the development of software-intensive SoS driven by their software architectures.

Index Terms—Systems-of-systems, SoS, software architecture, architecture-driven engineering.

I. CONTEXT

Nowadays, it has been possible to notice that single and monolithic systems, which represent a significant amount of the current running systems, do not seem to be enough to provide the complex functionalities required by applications in domains such as environmental management, smart cities, energy, and the military domain. In addition, these systems must imperatively meet important requirements such as high criticality, resilience, availability, reliability, and performance. In order to overcome these limitations, there has been an increasing interest in the research and development of large and complex systems resulted from the integration of other independent heterogeneous systems. This new class of systems has been referred as systems-of-systems (SoS).

Among the different definitions available in the literature, an SoS can be understood as a distributed composition of heterogeneous independent systems that cooperate to form a larger and complex system in order to accomplish a given mission [12]. However, the result of such an integration is more than the sum of the constituents as it enables an SoS to offer new functionalities, which cannot be provided by any of these constituent systems if they work as individual entities.

SoS have been consensually distinguished from other complex and large-scale systems due to a set of characteristics inherent to this class of systems [6, 12]. These features are: (i) operational independence of constituent systems, which provide their own functionalities even when they do not cooperate with other systems within the SoS; (ii) managerial independence of constituent systems, which can be independently managed from the other systems and from the SoS; (iii) evolutionary development of the SoS, which can evolve as a response to changes related to the environment, to the constituent systems, or to its functions and purposes; (iv) emergent behavior, which enables the SoS to provide new functionalities resulted from the collaborative integration of constituent systems and that are not localized in a single system; and (v) geographical distribution of constituent systems, which are physically decoupled and exchange only information among them. Furthermore, SoS are inherently dynamic, i.e., they can be composed, operated, and reconfigured at runtime.

II. PROBLEM STATEMENT

In the SoS context, software architectures1 have been regarded as a significant element for determining the success of such systems and contributing to their quality. Similarly to their counterparts in traditional, monolithic systems, decisions taken at the architectural level can directly affect the realization of business goals and the achievement of quality requirements [4]. In this perspective, SoS software architectures would be able to: (i) encompass the organization of the constituent systems; (ii) address the inherent characteristics of SoS; (iii) deal with functional and non-functional requirements; and (iv) systematize the principles that drive their evolution. However, existing architectural approaches do not adequately support the inherent features of SoS. For instance, emergent behavior is an essential feature of SoS typically not addressed in literature. Moreover, there are important gaps with respect to the design, representation, implementation, and evolution of SoS software architectures. In the following, three of these gaps identified in the literature are highlighted to be addressed in this PhD research.

Lacks related to the representation of SoS software architectures. A proper representation of SoS software architectures is quite important to the success of such systems as it can provide a basis for architectural analysis and guide their evolution. Recent literature reviews [9, 16] have shown that most of the existing representation approaches are semi-formal

1A software architecture is the fundamental conception of a system in terms of its constituent elements and the relationships among them and with the environment, as well as the principles that drive its design and evolution [10].
and based on the Unified Modeling Language (UML) and its derivations. However, the complex, critical nature of SoS calls for formal architecture description languages (ADLs) [13] as means of supporting rigorous specification, automated verification, and simulation of SoS software architectures. In addition, the verification of such properties becomes more important mainly when evolving an SoS, in order to continuously ensure its correctness before, during, and after evolution. As existing ADLs lack important features of SoS (in particular, evolutionary development and emergent behavior), a new ADL must be proposed in order to adequately support the representation and evolution of SoS software architectures.

Lacks related to interoperability and heterogeneity of constituent systems within an SoS. Constituent systems of an SoS are intrinsically independent, developed with different technologies and for diverse platforms. Considering such a high heterogeneity, interoperability among constituent systems within an SoS has been one of the most important topics addressed in this context as it is determinant for promoting emergent behaviors, for example [16]. In fact, it is not possible to have an SoS without an approach that enables its constituent systems to cooperate towards the accomplishment of the goals of the SoS. Despite the existing initiatives, more focus has been given to the discussion of challenges related to the interoperability among constituent systems [7]. Furthermore, no systematic approach to identify, select, and compose constituent systems to form an SoS at runtime was proposed yet.

Lacks related to the evolution of SoS software architectures. Finally, current literature still presents only initial proposals regarding the evolution of SoS software architectures [5, 18], thus meaning that few attention has been given to this topic in spite of their relevance [16]. In this context, it is important to understand why, how, when, and where such an evolution occurs and which characteristics and decisions must be considered in order to avoid future architectural degradation. Furthermore, it is essential to have integrated environments and tools, supporting mechanisms, and technologies able to transparently deal with evolution of SoS and hence to manage their dynamic architectures.

III. RESEARCH GOALS

Considering some of the issues related to the architectural design and development of complex SoS presented in Section II, the main goal of this PhD research is to investigate how SoS software architectures can be used as the central element for constructing and evolving SoS. In order to accomplish this goal, a set of research questions and hypotheses were outlined, as presented in Table I. It is noteworthy to mention that this research is focused on software-intensive SoS, which are mainly composed of software-intensive constituents [8]. This distinction is quite important as SoS can encompass other types of constituent systems.

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Hypotheses</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: How to represent SoS software architectures while addressing features inherent to this class of systems and also supporting validation of such architectures?</td>
<td>H1: A formal description language can properly tackle the inherent features of SoS software architectures.</td>
</tr>
<tr>
<td>RQ2: How to avoid degradation of SoS software architectures upon dynamic changes?</td>
<td>H2: Representing an SoS software architecture at runtime can address its inherent dynamicity and preserve its integrity.</td>
</tr>
<tr>
<td>RQ3: How to address the inherent heterogeneity of constituent systems while enabling their integration and collaboration within an SoS?</td>
<td>H3: A middleware platform can facilitate the integration of constituent systems into an SoS and abstract away their specificities.</td>
</tr>
</tbody>
</table>

IV. PROPOSED APPROACH

Aiming at answering the research questions, the main contribution is the proposal of an architecture-centric approach for constructing SoS focusing on the formal specification and dynamic reconfiguration of their software architectures. With such an approach, the goal is to support the specification, execution, and reconfiguration of SoS software architectures. The proposed approach is structured upon the elements depicted in Fig. 1 and briefly described as follows.

Architecture descriptions. At the architectural level, SoS software architectures are specified by using a formal ADL, as a response to RQ1. As discussed in [16], the SoS architectural characteristics go beyond the elements provided by the existing ADLs, thus being necessary to propose a new ADL (totally new or as an extension of an existing language) to adequately representing these features. The main elements that shall be considered when describing SoS software architectures are [1]: (i) both structural and behavioral elements of the SoS and its constituent systems; (ii) dynamic interactions among constituent systems; (iii) dynamic reconfiguration actions, an essential requirement in this context mainly due to emergent behavior intrinsic to an SoS and to the dynamic scenarios where they are deployed; and (iv) formal, explicit description of properties and quality attributes.

A software-intensive system is any system in which software plays an essential role in its design, development, deployment, and evolution [10].
**Architectural models at runtime.** A second concern in the proposed approach is representing an SoS software architecture at runtime, thus taking advantage of the models@runtime approach [3]. Models at runtime can be defined as the abstract representation of a system (including its structure and behavior) that exists in tandem with such a system during its execution. Such an approach has been recently advocated as promising to support dynamic evolution of software systems mainly in cases of unanticipated changes [2], as it is the case of the SoS context. Therefore, the intention is to leverage models@runtime as means of having traceable, manageable models representing elements of SoS software architectures at both design time and runtime, as well as the dynamic reconfiguration of such architectures. In addition, the causal connection between the architectural and execution levels allows addressing how reconfiguration actions specified at the former can be reflected into the latter (and vice-versa) while maintaining consistency between them, as a response to RQ2.

**Middleware platform.** Finally, a middleware platform between the architectural and execution levels can enable the seamless integration of constituents of an SoS, as a response to RQ3. Middleware platforms have been widely used as a key element to provide interoperability among heterogeneous distributed components (as it is the case of constituent systems within an SoS) while abstracting away their specificities. Moreover, the middleware platform is also in charge of dynamically synthesizing and deploying mediators, communication entities that do not only transmit data (as connectors in traditional systems), but are also able to specify and coordinate the interactions among constituent systems [11]. Finally, due to the dynamicity of SoS software architectures, such a platform can provide services (i) to enable the dynamic discovery of constituent systems within the SoS, (ii) to monitor the operations of both SoS and constituent systems at runtime, and (iii) to manage dynamic reconfiguration actions.

### V. Research Activities

**A. Ongoing Activities: SoS-ADL**

The development of the approach started from the architectural level, where the specification of SoS software architectures is the main artifact. In this perspective, the ARCHWARE Research Group of IRISA has developed SoS-ADL, a novel formal ADL to comprehensively describe SoS architectures while enabling their automated, rigorous analysis. This PhD research focuses on the dynamic features of SoS-ADL.

The formal foundations of SoS-ADL are based on process algebras [14], a mathematical theory for modeling concurrent systems and describing their communications/interactions at a high abstraction level. As process algebras have played a role of formal underpinnings for describing software architectures, SoS-ADL takes advantage of $\pi$-calculus [15], a computationally complete process algebra that provides an universal model of computation, and in particular of its $\pi$-ADL [17] extension developed for coping with architectural needs.

---

SoS-ADL extends the notions of concurrent communicating process defined in $\pi$-calculus by viewing constituent systems of an SoS as concurrent constrained processes that mediate interact with each other. Due to the inherent characteristics of SoS software architectures, such formal basis augments $\pi$-calculus by enabling dynamic bindings between architectural elements and comprising constraints about the environment to guide the interactions among constituent systems.

In SoS-ADL, SoS software architectures are described in terms of systems, mediators, and their coalitions, as depicted in Fig. 2. Systems represent the operationally and managerially independent constituent systems that collaborate among each other in order to compose the SoS as a whole. In SoS-ADL, a system comprises an interface with the environment (port) and an internal behavior specifying the capabilities related to the functionalities provided by such a system for fulfilling its mission. As previously mentioned, mediators specify the interactions among constituent systems, so that two systems can only communicate via a mediator between them. Similarly to system abstractions, a mediator comprises an interface with its environment (role) and an internal behavior specifying the coordination between interacting systems. Despite these elements are intuitively similar to the notions of components, connectors, and configurations defined in most existing ADLs [10, 13], they take a new interpretation in the SoS context.

As in $\pi$-calculus, SoS-ADL provides connections, which represent communication channels between systems and mediators. By used typed connections, these architectural elements can send (output connections) and receive (input connections) information. In order to attach a port of a system with a role of a mediator, there must be at least one binding between a connection of the former and a connection of the latter.

Coalitions define the ways in which how constituent systems and mediators can be arranged to form the software architecture of the SoS. It is important to mention that systems as architectural elements are not under control of the SoS itself (as they are operationally and managerially independent of it), but mediators are under such a control. Therefore, an SoS can create or reconfigure mediators at runtime, i.e., it can rearrange how constituent systems can interact among each other.

As SoS software architectures are inherently dynamic, the concrete constituent systems of an SoS are partially known or even unknown at design time. To cope with this requirement, SoS-ADL enables describing such architectures in an intentional way. In this perspective, systems and mediators that may be part of the SoS are abstractly described and the corresponding concrete elements are identified, discovered at

---

[^irisa]: http://www-archware.irisa.fr/
runtime. Therefore, different constituent systems and mediators can meet these intentional descriptions and hence result in different, alternative software architectures for the SoS.

Finally, bindings between systems and mediators are also dynamic, thus enabling different assemblies of these elements to compose the architecture. This is an important requirement in the SoS context since it might be impractical or even impossible to predict at design time how constituents and mediators will be connected in the SoS. Furthermore, it is possible to notice the role played by the middleware platform in this context as means of enabling such dynamic discovery of systems and synthesis of mediators within the SoS.

B. Next Activities

The next activities towards the accomplishment of the goals of this research encompass:

1) to endow the SoS-ADL language with architectural constructs for dynamic reconfiguration actions;
2) to specify the middleware platform between the architectural and runtime levels;
3) to define mappings from architectural elements to their respective runtime representations;
4) to develop proofs of concept to validate the approach;
5) to quantitatively and qualitatively evaluate the main elements of the proposed approach.

C. Evaluation

As means of evaluation, proofs of concept can be developed in order to validate the proposed approach from the specification of SoS software architectures to their execution and dynamic reconfiguration. Each of the elements that compose the approach can be also individually evaluated. For instance, the ADL used to specify SoS software architectures can be evaluated in terms of expressiveness for representing SoS architectural concepts by performing a controlled experiment with software architects. In turn, as the middleware platform introduces an additional layer between the architectural and execution levels, it is also possible to assess its impact in terms of performance. Finally, as executing an SoS in a real-world environment is unfeasible mainly due to its critical nature, simulations can be conducted in order to address the execution and dynamic reconfiguration of SoS software architectures.

VI. Expected Contributions

As main contributions to be resulted from this PhD research, it is possible to highlight: (i) a formal ADL to describe SoS software architectures, which can be further verified/validated; (ii) supporting mechanisms to dynamically evolve SoS, driven by their software architectures; (iii) the use of runtime models aiming to ensure consistency between the architectural and execution levels upon dynamic changes; and (iv) an integration of the proposed ADL with a middleware platform to promote interoperability and integration of independent heterogeneous constituent systems. Altogether, these elements within the proposed architecture-centric approach (another contribution itself) can support the specification, execution, adaptation, and validation of SoS software architectures. Furthermore, the proposed approach intends to contribute to fill some relevant existing gaps in this context and hence to improve quality in the development of software-intensive SoS due to the important role played by their software architectures.

ACKNOWLEDGMENT

This work was partially supported by the Brazilian National Institute of Science and Technology for Software Engineering (INES)\(^4\), funded by CNPq grant 573964/2008-4.

REFERENCES


\(^4\)http://www.ines.org.br