Service Oriented Computing and Applications
https://doi.org/10.1007/s11761-018-0245-1

SPECIAL ISSUE PAPER

A review and future directions of SOA-based software architecture modeling approaches for System of Systems

Ahmad Mohsin[1] · **Naeem Khalid Janjua**[1]

Received: 10 May 2018 / Revised: 2 October 2018 / Accepted: 6 October 2018

Abstract
Software architecture is a software system's earliest set of design decisions that are critical for the quality of the system desired by the stakeholders. The architecture makes it easier to reason about and manage change during different phases of complex software life cycle. The modeling of software architecture for System of Systems (SoS) is a challenging task because of a system's complexity arising from an integration of heterogeneous, distributed, managerially and operationally independent systems collaborating to achieve global missions. SoS is essentially dynamic and evolutionary by design requiring suitable architectural patterns to deal with runtime volatility. Service-oriented architecture offers several architectural features to these complex systems; these include, interoperability, loose coupling, abstraction and the provision of dynamic services based on standard interfaces and protocols. There is some research work available that provides critical analysis of current software architecture modeling approaches for SoS. However, none of them outlines the important characteristics of SoS or provides detailed analysis of current service-oriented architecture modeling approaches to model those characteristics. This article addresses this research gap and provides a taxonomy of software architecture modeling approaches, comparing and contrasting them using criteria critical for realization of SoS. Additionally, research gaps are identified, and future directions are outlined for building software architecture for SoS to model and reason about architecture quality in a more efficient way in service-oriented paradigm.

Keywords SOA · System of Systems · Architecture description languages · Architecture frameworks · Model-Driven Engineering · Dynamic architecture · Services orchestration · QoS

1 Introduction

The last decade has witnessed a tremendous growth in software architecture (SA), a sub-discipline of software engineering. SA focuses on architecture requirements specification and modeling of a software system's main computational parts (the components), the ways through which they interact (the connector) and their configurations at a certain level of abstractions. These abstractions known as models [1] along with architectural style[1] allow system architects to reason about key system properties, such as performance, reliability and security required by the stakeholders [2]. Therefore, the decisions made during the architecture design phase have significant implications for the software's economic and quality goals, and SA models help architects to organize structural components, configurations and reconfigurations and to perform architectural evaluations effectively.

Today, the rapid developments in inter-connectivity, ubiquity and agile software have led to rise of a new paradigm in software systems know as System of Systems (SoS) [3].

✉ Naeem Khalid Janjua
n.janjua@ecu.edu.au

Ahmad Mohsin
a.mohsin@ecu.edu.au

[1] School of Science, Edith Cowan University, Joondalup, WA, Australia

[1] Software *architecture style* may be defined as a typical way to organize system components with a set of features suitable to formulate a particular type of system architecture providing guidelines for design. A software architecture style can also apply one or more *architectural patterns* to solve a system problem. It is worth noting that in academia and industry terms, software architecture styles and patterns are used interchangeably.

Published online: 24 October 2018

SoS is a complex[2] mission-oriented system resulting from an integration of various legacy and new systems from different domains, directing resources and capabilities with managerial and operational independence. These complex systems are becoming more prominent in many domains such as healthcare, transportation, manufacturing, robotics and industrial-scale implementations. The recent emergence of IoT System of Systems (IoT-SoS), cyber-physical System of Systems (CPSoS) and industry 4.0 systems have accelerated the design and development of such complex systems. At the same time, sociotechnical SoSs, i.e., smart cities smart grids and emergency response systems, are altogether bringing new trend of systems engineering. Given the importance of SoS, a number of research projects are being carried out today at commercial as well as at academic and government levels to improve architecture and design practices. In this regard, some of the more prominent SoS research projects are T-AREA-SoS,[3] ROAD2SoS,[4] COMPASS[5] IRISA's ArchWare[6] and DARPA (SoSITE).[7]

While the architectural complexity of non-monolithic software systems is not as challenging as traditional monolithic systems,[8] it requires a set of coordinated software engineering activities spanning the SoS life cycle for modeling and eventually decreasing inherit problem of emergent complexity [4,5]. This is because the SoS Constituent Systems (CSs) interact with each other to form new configurations / coalitions to achieve a specific task with the significant characteristics of autonomy, belonging, connectivity, diversity and emergence where emergence is unforeseen behavior of the system due to undergoing topologies and changing environment of the system [3,6–8]. The biggest challenge faced by the SoS community is to model a dynamic architecture with the ability to specify unforeseen behaviors at runtime.

Plethora of research exists on modeling architecture for SoS and can be broadly categorized as follows:

1. Non-Service-oriented architecture (SOA)-driven approaches:

Architecture modeling approaches focus on building models essentially with styles other than SOA style. This includes various efforts at different times from the system engineering community, researchers and industrial SoS producers to model SoS with the help of Architecture Description Languages (ADLs), i.e., SysML,[9] EAST-ADL,[10] SosADL [9], model-driven architecture (MDA)-based architectural frameworks similar to DOADF,[11] MODAF[12] and CML [10] from COMPASS.[13]

2. SOA-driven approaches:

Architecture modeling approaches focus on building models for SoS using the SOA style. This includes various research attempts and standards mainly SoaML[14] by OMG SysML[15]-based profiles and some prominent ADLs [11–15] along with some model-driven engineering (MDE)-based architectural frameworks.

Various research surveys and reviews have focused on non-SOA-driven approaches for modeling SoS architecture. However to date, no significant research work has critically evaluated existing SOA-driven architecture modeling approaches specifically for SoS. Various surveys have been conducted from the SoS engineering and architecture perspective [7,16,17] individually on SoS and SOA which tried to identify architecture landscape for these systems. However, these research works do not give a description of SOA-based SoS architecture techniques or capabilities to inform readers about the specific features that an SOA-driven approach should have for SoS.

Therefore, to address the above-mentioned gap, this article provides a holistic and critical analysis of existing SOA-driven architecture modeling approaches for SoS. This research work first identifies the architectural characteristics of SoS and then uses those characteristics to categorize the existing modeling approaches into various taxonomies. After critical analysis, the research gaps, research trends and future directions are outlined to address the research issues associated with the modeling of SoS.

The rest of the paper is organized as follows: Sect. 2 describes the SoS landscape, showing its significance and

[2] Complex *Systems* are systems which are essentially nonlinear, networked, emergent and self-adaptive. Compared to traditional monolithic systems which are essentially linear in nature, SoS is in contrast nonlinear systems with unstable changing states.

[3] https://www.tareasos.eu/.

[4] http://www.road2sos-project.eu.

[5] http://www.compass-research.e.u.

[6] https://www.irisa.fr/fr/equipes/archware.

[7] https://www.darpa.mil/program/system-of-systems-integration-technology-and-experimentation.

[8] In a *monolithic system.* multiple functionalities, i.e., (data input/output, processing and error handling), all are packaged together under a single module to meet the system needs. A *monolithic system* is usually complex and difficult to maintain as it is not flexible enough to cope with rapid changes. On the other hand, *non-monolithic system* is designed in such a way that it has smaller functionalities/services, independent of each other, using simple communication, making it easy to deploy and maintain large distributed systems.

[9] http://www.omgsysml.org/.

[10] http://www.east-adl.info/.

[11] http://dodcio.defense.gov/Library/DoD-Architecture-Framework/.

[12] https://www.gov.uk/guidance/mod-architecture-framework.

[13] http://www.compass-research.eu/approach.html.

[14] https://www.omg.org/spec/SoaML/1.0.1/.

[15] http://www.omgsysml.org/.

describing formal definitions and classifications of SoS. Section 3 provides an overview of SOA as a candidate for modeling an SoS. Section 4 presents current state of the art of service-oriented SoS architecture modeling techniques. Section 5 presents the critical analysis of findings and results, and Sect. 6 highlights the research issues associated with SoA-SoS modeling techniques. Lastly, Sect. 7 suggests future research directions and concludes the paper.

1.1 Motivation

SoS is complex, dynamic and evolutionary systems requiring prior planning in anticipation of runtime changes of architecture at design time to cope with any uncertainty arising from emergent behaviors. These systems require loose coupling, dynamic binding, interoperability and continuous services provision. In principle, SOA offers these features to such complex systems but there are certain limitations in existing SOA-based modeling techniques in terms of meeting SoS architectural challenges. The overall objective of this article is to identify those modeling techniques with a comprehensive review of their key features, to determine the extent to which they align with the architectural characteristics of SoS, and to identify the deficiencies that need to be addressed. To address these concerns, the following research questions have been devised:

RQ1: What is the state of the art of SOA for modeling heterogeneous distributed complex systems?
RQ2: What are the key architecture modeling approaches in terms of Architecture Description Languages (ADLs), model-driven engineering and architectural frameworks for SOA?
RQ2.1: Do these techniques adequately address the issues of dynamic services identification, their composition and provision at runtime?
RQ2.2: If so, then what is the extent to which these techniques have capability to define abstract architecture at design time and realize at runtime?
RQ3: Do the existing techniques have formal foundations to provide syntax and semantics to reason about dynamic services modeling and QoS attributes?

To answer these questions, a taxonomy of SOA modeling techniques has been developed that has been investigated in terms of the desired architectural description features for SoS.

2 SoS landscape

2.1 Definition and classifications

Various definitions of SoS have been offered by different researchers at different times, the most recent being that of J.Boardman et .al [6] in 2006. They defined SoS as a collection of many entities also known as constituent Systems (CSs) and their relationships, coordinating to achieve a specific task with meaningful characteristics of autonomy, belonging, connectivity, diversity and emergence where emergence is unforeseen behavior of the system due to undergoing topologies[16] or configurations and changing environment of the system.

According to ISO/IEC/IEE 15288 (ISO, 2015), " A System of Systems (SoS) brings together a set of systems for achieving a task that none of the systems can accomplish on its own, where each constituent system keeps its own management, goals, and resources while coordinating within the SoS and adapting to meet SoS goals ".

In his classic definition of SoS Maier [8] describes, SoS as a unique type of systems built from other stand alone systems, working as constituent systems (CSs) which are larger in size, more complex with components exhibiting managerial and operational independence, evolutionary in nature, geographically dispersed and exhibit emergent behavior. SoS is nonlinear systems that are required to model dynamic reconfigurations with evolutionary development and emergent behavior. Compared to linear monolithic systems where output is according to the given input and their behaviors can be predicted by describing their components, connectors and resulting configurations at design time, SoS can not be modeled completely with the same descriptions due to dynamism. SoS with the help of emerging behavior resulting from the interactions of independent CSs, complete larger tasks which are not possible to be completed by individual systems alone [6] but it comes with cost of complexity. This requires formal understanding of the complexity of SoS, in order to determine their structural composition from architectural perspective.

Formal Definition Formally, at abstract level, SoS is simply the composition of various independent CSs represented as follows:

$$\sum_{i=1}^{n} CS_i = CS_1 + CS_2, ..., CS_n \tag{1}$$

SoS Structural Complexity Systems architectural complexity can be categorized into [18] structural, dynamic and

[16] A software architecture *topology* is a typical arrangement of software components and connectors resulting from their interactions. It is also called *architecture configuration* and also referred to as system *coalition* resulting from interaction of CSs of an SoS.

organizational complexity. From the architectural perspective, SoS can be described in terms of components also known as constituent systems—CSs, connectors which are mediators and configurations are also called coalitions. For providing and accessing specified services from different Components, Interfaces come into place. SoSC (Sos Complexity) is a sum of individual CSs complexities and Product of complexities of interfaces and coalitions.

$$\sum_{i=1}^{n} ICOMPcs_i + \left[\sum_{i=1}^{n} INTERFACEcs * \sum_{i=1}^{n} CONFcs\right] \gamma E(A) \tag{2}$$

The equations above can formally be defined as

$$\sum_{i=1}^{n} \alpha i + \left[\sum_{i=1}^{n}\sum_{j=1}^{n} \beta_{ij} A_{ij}\right] \gamma E(A) \tag{3}$$

Where *ICOMPcs* represents individual CS complexities in SoS, I*NTERFACEcs* term represents complexities caused by interfaces and the last term *CONFcs* shows complexity due to the coalitions among CSs. In equation 3, αi represents independent CSs complexity, β_{ij} denotes interface complexity between *ith* and *jth* CSs, Aij represents corresponding elements in the adjacency matrix, and γ represents a scaling factor for the sum of participating values.

In the final equation α represents complexity of *ith* CS element, interface the complexity between *ith* and *jth* CS elements, A_{ij} denotes corresponding element in the adjacency matrix, and lastly γ reflects factor showing the sum of participating values.

With the passing of time the complexity of SoS increases as new CSs are introduced: Some are deleted and modified adding complexity to the system architecture for modeling. It is important to see how such complexity can be reduced by applying a specific architectural pattern to describe the SoS architectural model early at design stage.

In terms of management and operation, Maier further classifies SoS into three categories: directed, collaborative and virtual SoS. It is important to understand that the organization and governance of systems may vary as depending on the type of system. However, Ian Sommerville [19,20] gives a more concise classification of SoS based on governance. Governance is the way in which policies, management, operations and evolutions of the SoS are defined by systems designers and policy makers. Hence, his classification of SoS is based on governance where he classifies SoS as:

- Organizational SoS: An e-procurement system of a large company provides formal directions to participating CSs namely ordering CS, accounting CS and inventory CS with the single policy making body to govern. Services provision is centrally directed by the governing body itself.
- Federated SoS: A disaster management system which integrates various services from different CSs and may include fire, police, coast guard and rapid response systems. Here governance is achieved through various organization's CSs collaborating to achieve a mission.
- Coalition SoS: An integrated Online Educational System of a university, providing various independent educational systems like LMS, students and faculty mailing services, research management tools, laboratories equipment management software and other related services. Participating CSs come from various sub-organizations of the university and interact with each other to achieve specific missions with informal collaborations.

Governance and management are contradictory since governance is about the aims and objectives of SoS whereas management is used to realize those defined objectives. Figure 1 shows classification of SoS, namely as organizational, federated and coalition SoS, based on different governance principles. In an organizational SoS, there is one policy making body to govern SoS , though CSs are independent but there exists a central body to manage it, whereas in federated SoS there is somewhat voluntary collaborative governance to manage CSs on mutually agreed policies. Governance, managerial and technical complexities contribute to architectural complexity of each type of the system. One important point that architects needs to understand is that in all these SoS categories CSs are independently managed and operated which is distinct feature of any SoS.

2.2 SoS emergent properties: A case for SoS architecture modeling

Due to the interconnection of the SoS CSs new properties of SoS arise; these are known as emergent properties and have a significant impact on the system architecture. These emergent properties become active and evident only when CSs of SoS start communicating or they undergo certain transformations during operational phase. In fact, emergent behavior and the evolution of the overall system require these systems to be modeled from the runtime perspective at design time. These can be classified into:

- Functional emergent properties: These are activated when all CSs work together to achieve some task . For example, the case of disaster management SoS has the functional property to manage disaster situation with the

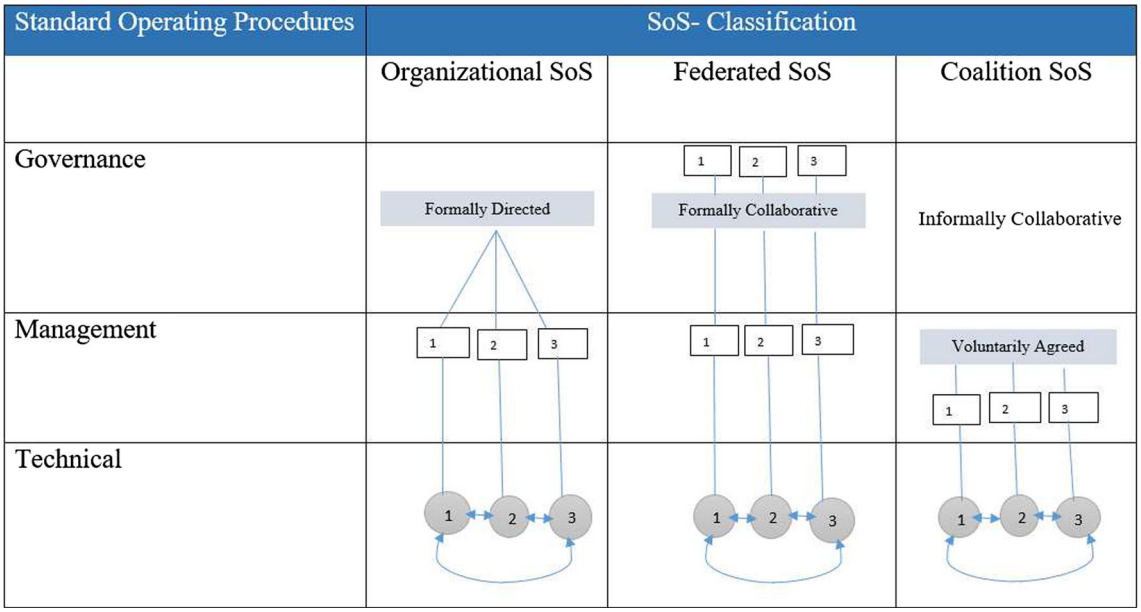

Fig. 1 SoS Classification based on Governance adopted and modified from [19]

help of police, firefighting and rescue services once it calls relevant CS's services in real time.

– Non-functional emergent properties: These are related to the behavior of the CSs in their SoS operational environment. These properties are family of system-ilities i.e. quality attributes which are critical to the SoS. Some examples include performance, security, reliability and availability. If these properties do not meet well-defined thresholds for SoS, then the system is useless or considered unsuccessful.

When visualizing a complex system architecture like that of SoS, it is vital to predict or model these emergent properties in advance during the design phase. Architectural decisions made in relation to these two types of emergence in SoS are significant tasks which require rigorous modeling and reasoning of the architecture description early in the system's life cycle.

The architectural patterns with certain tactics can be applied in a particular modeling notation to specify these emergent properties in relation to identified scenarios in order to model and simulate these behaviors for better analysis. Selection of an architectural style for SoS is a careful process that requires domain expertise by aligning architectural decisions with overall objectives of the system.

3 SOA-driven SoS architecture

SOA is a way of architecting and deploying systems where provision of services is the priority for performing system computational tasks. It is an approach for designing and developing complex systems as an architecture style. SOA incorporates architectural features to integrate legacy and emerging complex systems with well-defined interfaces, platform independence and interoperability. In a service-oriented system, a service represents a computational component which performs some specific tasks or functionality. SOA is one of the suitable architecture styles for modeling loosely coupled, dynamically composed and discoverable software components essentially as services [21]. It is being applied in a range of systems from enterprises to embedded systems to robotics, IoTs and cyber-physical systems [22–24], thus allowing the heterogeneous integration of many sub-systems to achieve specified global missions of a much larger system. In the literature [25] five different architectural styles have been identified for modeling SoS and among these SOA has been described as most suitable pattern for modeling the architecture of such systems. Every architectural style comes with its own pros and cons, and SoS needs architecture patterns which are capable of constructing systems in such a way that quality attributes are not compromised by evolution. In this regard, due to its unique features, SOA is one of the styles that can be applied to model SoS architecture to deal with its inherent complexities. SOA-based SoS allows CSs to provide their functionalities through public interfaces to interconnect with other constituents with well-defined protocols allowing new compositions without violating service-level agreements (SLAs) for better quality of service (QoS).

To further illustrate SOA-based SoS and complexity, lets illustrate the case of an autonomous vehicle or a self-driving car which is a typical form of emerging cyber-physical SoS.

Table 1 Service components architectural modeling

SOA modeling concepts	Description
Static services	A static service architecture model describes services at design time. It includes names, identifiers, provided and required services description
Dynamic services	Dynamic services model describes the evolution of services components due to varying requirements. Service components described when services are composed, discovered and updated at runtime
Services composition	Behavioral modeling of services in complex collaborations to form services composition and modeled as services orchestration and services choreography
Services reconfiguration:	A model's ability to describe services reconfiguration at runtime, and it includes how services architecture evolves at runtime under certain conditions

Autonomous vehicle drives and controls the car with the help of various CSs, participating as service components (SCs) to fulfill required functionalities, like city maps SC, emergency alerts SC, traffic congestions and weather forecast SCs, where every service component itself is an independent system. The structural complexity of such SOA-SoS is the sum of the individual SCs and product of integrated services. The Car's central controlling actuators and sensors receive and process information provided by these CSs as services at runtime. The overall behavior of the system depends upon the services provided by these components to the vehicle. SoS vehicle requests services from local mapping system which provides map's information according to required parameters for the navigation of the car. Vehicle decides better routes based on the information received from traffic management system that provides real-time traffic data in the area. Similarly it manages and adapts the car's internal environment factors such as temperature based on the real-time weather data provided as a services to the vehicle. These services are discovered, requested and composed at runtime, requiring such systems to be architecturally modeled statically and dynamically to verify and validate unknown behaviors. The dynamic discovery of various services and runtime addition or modification of services for this typical system adds structural complexities requiring to model it from design time and runtime perspectives with well-defined syntax and semantics.

On the one hand, SOA has dynamic and adaptive nature to accommodate dynamic runtime architectural changes. Ideally, SOA-based systems should exhibit dynamic services discovery, composition and choreography. However, SOA introduces architectural complexities to systems which need to be reduced by modeling and analyzing with proper validation and vitrification support. As in a work by [26], an SOA-based system must be modeled and described from different viewpoints of stakeholders. SOA has the ability to model the complex heterogeneous, distributed systems that run in different execution environments [8] ensuring maintainability, modifiability and interoperability. Apart from these quality attributes, there are several other important attributes which need to be described explicitly in models , i.e., performance, security, availability and reliability [27]. SOA-driven architecture has relatively higher support for these features as compared to non-SOA architectural modeling styles, essentially due to its underlying architectural characteristics, it offers to SoS [28–31]. Therefore, SoS architecture modeling can better be described to address the architectural challenges like emergence and evolution by employing SOA paradigm [25,32]. However, it is questionable whether existing SOA architecture modeling approaches can provide sufficient modeling notations for describing these quality attributes and further providing validation in order to realize abstract architecture at runtime.

Currently the key challenges [4] for SoS designers are regarded to architecture modeling at an abstract level. Service-oriented SoS can be described at various levels of abstraction; however, variability and dynamicity are causes of changes in QoS attributes and SLAs, and therefore, dynamic architecture modeling is vital to deal with complexity arising due to emergent behavior and evolution. From architectural perspective, SOA-SoS can have static and dynamic models with configurations defined as services composition. Table 1 describes services architectural modeling concepts along with their detailed descriptions.

4 State of the art

Recently there has been increased tendency to develop SoS with tremendous growth in sensors, actuators and embedded systems platforms giving rise to IoTs and CPS, Smart systems and Sociotechnical systems. These systems employ heterogeneous diverse architectural style of SOA in design and development [33–36]. SOA offers great features for SoS to design architecture at large scales as described in previous section. Industry and academia have taken many initiatives [4] toward the development of SoS, and it is worth noting that majority of these systems are using cloud infrastructure (public, private or hybrid) employing service-oriented computing styles (SOAP,RESTful and Microservices) to acquire

Table 2 Evaluation criteria for SOA-based architecture modeling approaches

Criterion	Description
Static architecture modeling	Static architecture modeling of SoS includes the defining of computational components as constituent systems—CSs—and; their relationships with the help of mediators and resulting collations at design time. For SOA-SoS, these CSs are essentially service components, their relationships are interactions and interfaces are provided and required interfaces. A static architecture description / model does not change during runtime
Dynamic architecture modeling	The ability of the architecture modeling approach to model architecture behavioral changes at runtime is the concern of this aspect. We search for descriptive and modeling approaches which support dynamic architectural modeling due to some changes in SoS as a result of addition, deletion or any other modification. From the SOA perspective it is associated with runtime services composition, services discovery, services orchestration and choreography among systems. A dynamic architecture description/model changes at runtime
Runtime reconfigurations	Runtime reconfigurations are the results of changes in the configuration which eventually leads to a Reconfiguration of the overall system layouts. When new services are introduced or existing services are removed or modified, SoS goes through a transformation. It is important to investigate the ability of existing AM techniques to assess this important variation and self-adaptation of SoS
Quality attributes representation	The ability of AM techniques to model quality attributes of the SoS at design time and runtime is considered a very important feature. How different modeling approaches support to describe quality attributes at both levels of architecture is the key question. We also how quality of service and SLAs may be impacted due to runtime changes in overall system architecture

the maximum benefits of the underlying architectural characteristics of SOA for SoS. Complex and large-scale distributed SOA-based systems can be modeled using different modeling techniques like Architecture Description Languages (ADLs), architectural frameworks and combined approaches. We investigate the capabilities of these architectural modeling approaches for designing service-oriented SoS. Key architectural issues faced by SOA-SoS include but are not limited to: (i) constituent systems–service components description, (ii)dynamic modeling of constituent service components to support emergent behavior evolution (iii)support for quality of service-related attributes and their integration with respect to evolution (iv) perform architectural analysis due to the continuous requirements changes.

In the preceding subsections, we sought these desired architectural modeling features rigorously in the existing literature. For a comprehensive investigation, there is a need to devise a set of criteria to evaluate existing literature of the SOA-SoS world. Consequently, we devised an evaluation criteria and moreover, developed a taxonomy related to the SOA-SoS architecture landscape. Table 2 outlines the criteria used to compare and classify the current state of the art into various categories of SOA-SoS architecture modeling techniques.

There are various methods for modeling software architecture of complex systems. Existing architecture modeling techniques such as ADLs, model-driven architecture (MDA)-based ADLs , UML profiles and MDE-based architectural frameworks have their specific notations and model kinds to provide architectural views. Each approach has its strengths and weaknesses in terms of describing SA as per ISO/IEC/IEEE standard 42010 [37] to address stakeholder concerns. Based on the qualitative analysis of the literature, we have compiled a taxonomy as shown in Fig. 2 for SOA-SoS architecture modeling techniques consisting of ADLs, MDE-based UML profiles and mixed approaches, where each category is further divided into sub-categories and individual techniques. The various categorizes of architectural approaches are investigated based on the architectural characteristics of SoS , ISO/IEC/IEEE 42010 standard [37] and 4+1 view architectural model [38,39]. The service-oriented paradigm as style for modeling an SoS that provides various associated benefits, described in detail in previous section, still faces challenges of services description at abstract level and their realization at concrete level in relation to SoS unique characteristics. Modeling an SOA-based SoS, specially when it comes to describing architecture dynamically due to emergent behavior and evolution, SOA for SoS can only be realized truly, if related architecture modeling approaches have enough capability to specify and reason about runtime changes [40]. Static or structural specification describes components in terms of service requester and service provider at design time with well-defined public interfaces for information exchange. During analysis of such models, the requester service component should be able to discover new services in order to address new requirements of the SoS. The outcome of such analysis will also help system architects to measure the adherence of SoS quality attributes with respect to SLAs. Each modeling category in taxonomy has been reviewed thoroughly to search these capabilities for modeling SOA-based SoS.

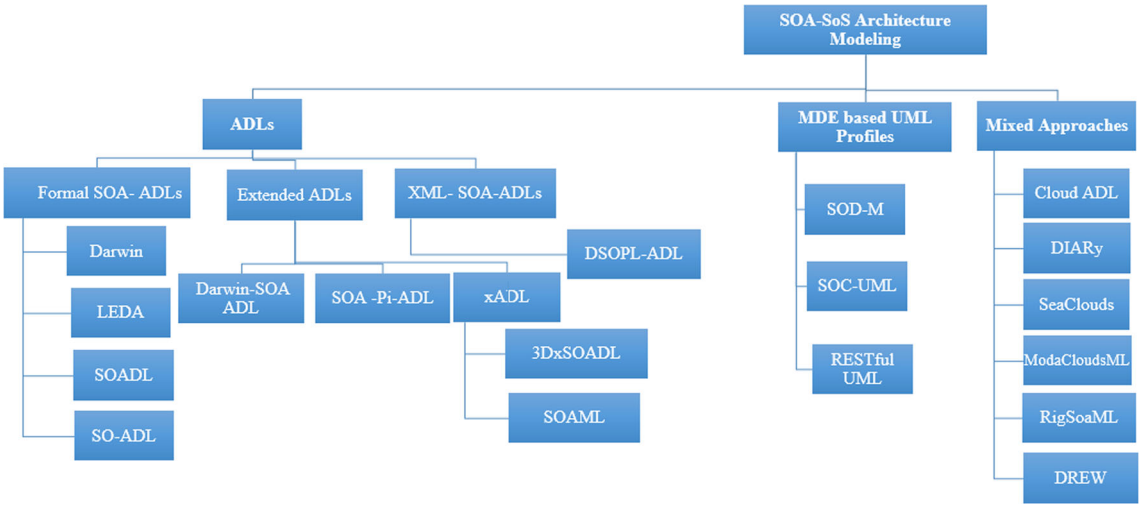

Fig. 2 SOA-SoS: Architecture Modeling Approaches Taxonomy

4.1 Architecture description languages to describe SOA

ADLs are formal notations for developing structural and behavioral models of large complex systems with underlying syntax and semantics. Our review identifies ADLs targeting SOA modeling from late 90s to the modern era of software architecture modeling and then consequently review existing literature to see if these ADLs provide features needed for SOA-SoS. Prominent notions for ADLs to describe software architecture are components, connectors and configurations. According to different researches [41–43], ADLs should have first-class connectors, formal semantics and well-defined interfaces to make to possible for performing systems architecture analysis early in the life cycle. On the one hand, SOA is an architectural paradigm which offers systems architects to design flexible, extensible and reusable services across heterogeneous environments enabling systems to dynamically discover services during execution time and tries to solve the problem of interoperability. But on the other hand SoS, architects need to specify these services, their compositions and orchestration statically and dynamically in keeping in line with functional and non-functional requirements evolution. Main problem faced by SoS type of systems is that they have their unique architectural characteristics which makes it difficult to predict unforeseen behaviors at runtime. Various ADLs have appeared from time to time to model services at the architecture level. Majority of these ADLs are formal based on $\pi-$ Calculus, FSP and CSP formalism, few are based on XML being less formal or semi-formal, while some have been extended from existing ones to provide abstract models. But existing formalism provided by these ADLs still may not be sufficient to reason about complex architecture of SoS. Hence, we conduct an in-depth investigation to evaluate the abilities of current ADLs to meet these needs.

SOA-ADLs can be categorized broadly into formal semantically rich ADLs with strong mathematical foundations, semiformal and informal procedural ADLs. There is a clear trend toward UML profiles-based development of ADLs with the incorporation of MDA approaches. We evaluate each language one by one critically using comparison factors of formalism support, structural representation, dynamic description, ability to realize runtime configurations and tool support.

4.1.1 Formal SOA-ADLs

In this part of literature review, we search for ADLs based on formalism to describe service-oriented architectures for large distributed systems. Large complex systems architectures evolve with the time, and hence, they are needed to be modeled from structural and behavioral viewpoints [44] in particular.

Formally well-founded ADLs play a vital role in describing, modeling and analyzing architectures dynamically. In this regard, the Darwin ADL was the first of its kind of languages to model services at structural and behavioral levels. Darwin [45] may be considered as pioneer ADL which tries to describe distributed systems architecture based on formalism. In this ADL components interaction is achieved by accessing required services through public interfaces. Each component is coupled with required service and provided service. To achieve formalism, Darwin was based on $\pi-$Calculus to enhance architecture behavioral specifications to model dynamic configurations. Later on to enhance its capability for behavioral specifications Finite State Process (FSP) [46] was incorporated in [47]. It introduced mainly

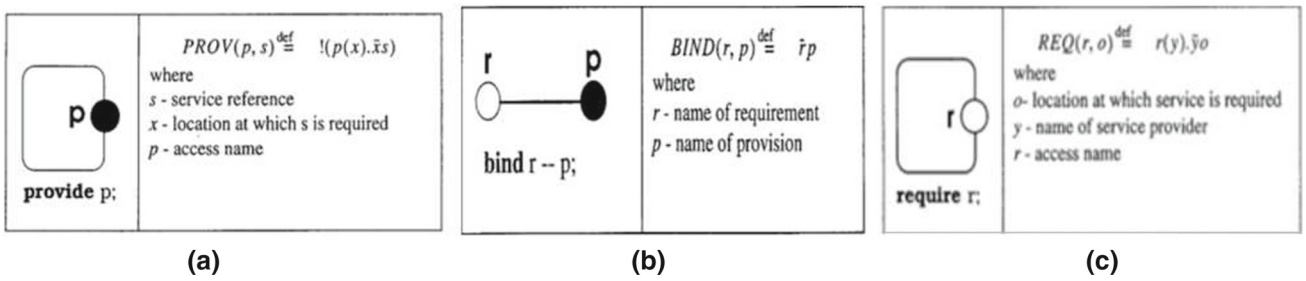

Fig. 3 Services orchestration in Darwin graphical representation: adopted from [45]

two types of instantiations namely lazy and dynamic instantiation for composite components services, where connectors among instantiated components are declared in bind statements. This ADL allows system architects create services components at runtime. A dynamic system architecture can be described using different agents modeled in π–Calculus namely: PROV as provided agent, REQ as required agent and BIND as binding agent illustrated in Fig. 3, where process p and r are service access names, that are exchanging data through binding mechanisms of communicating channels. As a result of composing these agents, services are managed as service components for implementing the respective behaviors.

Though Darwin tires to describe service components, their connections and configurations composing them dynamically, but it does not provide any features to analyze these models for further decision making in system development process as it lacks tool support.

LEDA was introduced in 1999 by Canal et al. [48] for modeling behavioral representation of architecture in distributed environments, and like Darwin it uses $\pi-$ Calculus for formal semantics. This ADL deals with architecture behaviors as internal behavior and external behavior. Role types are used for defining external behaviors to be instantiated in components, and similarly, internal behavior of components is defined using specs construct. Authors of this language further improved services composition, reuse and extension using refinement and validation to manage evolving complex systems architectures. Further they used object-oriented concepts like polymorphism for refining and extending dynamic services components. Using LEDA prototype architecture can be executed and analyzed because of formal semantics defined in $\pi-$ Calculus but still executable model at implementation level infancy required to model dynamic configurations . Xiangyang Jia et al. tried to develop a new ADL named SOADL [12] which is quite a unique effort that attempts to model SOA system capable of specifying interfaces, semantics, behaviors along with quality attributes of services described. It has the capability to describe SOA architecture for system statically as well as dynamically using XML as its metalanguage. Service components are described at abstract level implemented as architecture services, while connectors are described as binding between component service ports, making it simple for services composition. Reconfigurations are guaranteed by the SOADL by the triggering of events. One can say it is a much better language to support the dynamic behavior of SOA-based system architecture. Again like other ADLs claiming dynamic modeling, SOADL also used $\pi-$Calculus as a formal semantic for performing atomic and composite reconfigurations. It is capable to generate code from SOADL descriptions to BPEL notations and further allows architects to perform services composition validation to check unexpected behaviors. SO-ADL [49] provides constructs to model services (i.e. components, interfaces and service configurations) at design time, however it does not provide features to analyze constructed models to measure the quality attributes. The second important element of this language is connector which interconnects various services consisting of interfaces (inputs and outputs) using protocols. Configurations are provided as directed graphs of service components and connectors. This ADL does not define how services at runtime are composed, nor it provides notations about quality attributes specification at design time or runtime. In this work, the authors also proposed for a formal textual and visual description of architectural elements primarily based on this ADL.

4.1.2 Extended ADLs for SOA-based systems

Extended ADLs are mix of formal approaches and semiformal approaches primarily based on existing ADLs to enhance SOA modeling capabilities. Zhang Tao et al. proposed an extended SOA ADL [11] using Darwin. Since Darwin supports dynamic architecture description based on π- Calculus with formalism, it is flexible to be extended to various types of systems. They developed this ADL based on four key requirements of service-oriented systems. Authors define a service provider component type with output input interfaces and service specification. Connector type is a public publisher and requester interfaces coupled with service provider specifications. It has the capability to describe dynamic SOA Configurations at runtime. Service component serves as a key

component which holds all the information from the provider and fulfills requests allowing services consumers to replace services selection at runtime to improve QoS constraints.

π- ADL for SOA [50] is a similar effort like in [12] to model the dynamic behavior of SOA-based system architecture. At architecture level service components structure and composition is described as a part of component behavior. It is a formal ADL to describe the dynamic behavior of service compositions textually as well as visually with the ability to transform to SOA conforming structure. So in a sense, it tries to model SOA dynamically but it does not model services reconfiguration allowing the dynamic discovery and invocation of newly registered services. Miladi et al. [51] introduced *3DxSOADL* as extension of xADL [52] to describe distributed architectures based on SOA style. This ADL has the capability to deploy and manage system with repository container, services distribution , enabling dynamic evolution of architectures and its adaptation to meet the evolving requirements. It uses MDA technique as a refining process for deploying and managing service components reducing the gap between abstract modeling and implementation. P. Pannok et al. designed SOAML [13] in 2013 as another extension of xADL to model the SOA pattern of the software systems adopting this style. It encapsulates SOA style for describing provided and required services. Extensions for SOA with xADL are separately attached without changing the original ADL. In their research paper, [53] authors presented an Architecture Description Language (ADL), as an extension of the SOAML language, to specify integration of incremental architecture into existing cloud services. It allows architecture to automatically generate new services composition, deployment and reconfiguration scripts that change service invocations according to the specification. This is quite a unique work as this ADL tries to capture architectural characteristics of cloud-based SOA for such large complex systems and caters about quality attributes like elasticity and scalability. Key advantage of this ADL is that it comes with tool support to atomically manage new services composition and reconfigurations during Integration and deployment phases.

4.1.3 XML-based SOA-ADLs

In this section we evaluate SOA-ADLs based on XML, offering flexibility for multiple platforms with semi-formalism. These ADLs have less formal support but provide some promise to model dynamics. XML-based DSOPL-ADL [54] is modular ADL which allows architects to describe architecture variability and dynamic reconfigurations of services at runtime in service-oriented architecture-based product line systems. It may be considered a serious effort toward architecting SOA-based systems dynamically, and especially this ADL offers significant features for describing dynamic services reconfigurations as compared to previously discussed ADLs taking care of various phases of a system architecture life cycle at both design time as well as runtime. It has been divided into four key parts as:

- Structural Elements Description: Service name, service type and service description.
- Variability Description: Service binding with preconditions. It defines a mechanism for choosing an alternative service based on requirements variability.
- Context Description: Architecture behavior changes with changing contexts.
- Configuration Description: Services selection and composition at runtime by describing configurations rules at architectural level.

DSOPL-ADL extension [55] is an extended work of DSOPL-ADL authors tried to bring dynamic reconfigurations in services at architectural level by synchronizing architecture binding with contextual changes/variability. During system's execution, architecture can adapt its behavior to environmental changes that are specified as context elements and consequently generate a concrete architecture. Generated concrete architecture is transformable to another level of abstraction, such as to business process level.

AO-ADL was introduced by [56] with remarkable language capabilities and tool support to model services using architectural templates. AO-ADL is primarily an aspect-oriented ADL with range of modules to model systems and with a special emphasis on extended connectors to specify compositions. Based on architectural requirements its associated tool generates various services configurations depending on requirements described in the architectural template. But despite its ability to model variability, it does not tackle services composition neglecting to model how services composition is achieved either through choreography or orchestration.

Table 3 presents a critical analysis of ADLs support for formalism, ability to provide visualizations, tool support and capability to perform analysis for service-oriented systems architecture. It summarizes formal, extended and XML-based ADLs. After a detailed analysis of these ADLs it is interesting to note that majority of the ADLs designed earlier were formally based on strong mathematical principles like process algebra with different variants of $\pi-$ Calculus, as well as CSP [57] and FSP [46] notations, which is a strong evidence that earlier SOA-ADLs were more formal as compared to languages which appeared afterward. In terms of textual and graphical representation Pi-ADL for SOA can describe architecture in both ways. However, although it is based on Process Algebra, it does not provide facility for performing analysis on modeled architecture for further decision making. DSOPL-ADL is the least formal but it, in contrast,

Table 3 ADL's analysis for SOA-based Systems: +++ means fully supported; ++ means partially supported; – means not supported at all

SOA- ADLs	Formalism	Model visualization	Analysis capability	Tool support
DSOPL-ADL	–	++	–	+++
Extended Darwin for SOA	+++	+++	–	–
Π– ADL for SOA	+++	+++	–	–
3DxSOADL (Extended Xadl)	++	+++	–	–
SOAML	++	++	–	–
DARWIN	+++	++	++	–
LEDA	+++	++	–	–
AO-ADL	–	+++	–	–

provides the capability to describe architecture textually and comes with a tool support which converts ADL constructs to BPEL for services corresponding to required business goals.

4.2 Mixed approaches

This part of the literature review summarizes the efforts made toward modeling SOA-based systems aligned with emerging trends of IoTs and cloud computing giving a rise to SoS. These approaches are different from ADLs as they are designed to model specific domains of interests where services-oriented computing and cloud infrastructure play a vital role with less focus on formal notations but more emphasis is on providing multi-views for architecture models to stakeholders. These mixed approaches emphasize services orchestration, choreography and QoS for complex dynamic systems. One architectural framework [58] was proposed to model IoT systems based on the SOA style to manage an intelligent API layer as well as services publishing, subscription and QoS. The authors proposed the use of Microservices to fragment different IoT-SoS in order to cater system evolution and extensibility. Mircorservices are divided into functional services and non-functional services. This framework tries to specify key aspects of services architecture but still it lacks the capability to analyze impact of new configurations taking place as a result of changes. Perez and Rumpe [59] proposed CloudADL an extension of MontiArc [60] which was designed to model distributed CPS. CloudADL describes the high-level logical software architecture textually, but the technological requirements specific to the cloud environment must be specified by using other architecture description languages which remains a limitation in this approach. Dynamic reconfigurations of cloud application architectures were presented as DIARy [14], essentially a model-driven approach for services modeling. This approach allows architects to model dynamic architectures for services in cloud environments for dynamic composition and discovery. This technique presented good results for services orchestration in clouds, but still it needs empirical validation for multiple platforms.

In 2015, as a part of the European project SeaClouds,[17] Brogi et al. [15] developed a framework for dynamic cloud services selection to handle applications in dynamic environments using multiple cloud platforms as a service in heterogeneous clouds to deal with architectural issues like QoS and SLAs. The key challenge addressed by this work is service orchestration at runtime by conforming functional and non-functional attributes based on behavioral and contextual changes. It applies dynamic reconfiguration in the context of adding new services, replacing already in use or deleting existing services. This approach conforms with major SOA standards like OASIS[18] and TOSCA.[19] Seaclouds allows reconfiguration of architecture using chaining requirements provided as input with discover, planner and deplorer as main components. Although it promises to be a good approach for such complex systems for heterogeneous cloud services, authors fail to demonstrate any particular notation to Architect system at structural or behavioral level. Moreover, it does not employ any formal semantic approach to model the architectural changes at runtime. Under ModaClouds [61] project various methods and frameworks have been devised with tool support to model early design using meta-models capable of code generation for application deployments and to simulate runtime behavior of the systems in multi-cloud scenarios. The work done allows applications to choose suitable services of clouds from IaaS and PaaS perspectives using models at runtime [62] guaranteeing QoS despite ever-changing requirements.

In [63], the authors worked on modeling dynamic service-oriented SoS. Their contribution is toward describing the choreography of services, i.e., partial behavioral description

[17] http://www.seaclouds-project.eu/.

[18] https://www.oasis-open.org/standards.

[19] http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

at runtime by extending SoaML[20] as rigSoaML to provide formal semantics based on states and graphs transformation for services design. On the contrary, this technique lacks basic formalism grounds for describing dynamic services choreography as it uses UML as an underlying tool to describe services behaviors for services collaboration and roles. Moreover, it partially supports dynamic architecture specifications of SOA for SoS, as it does not provide complete descriptions when services undergo various behavioral changes. In its modeling approach, it does not focus on describing QoS-related attributes at either structural or behavioral level.

To model dynamic reconfiguration of web services at runtime, a middleware-based approach DREWS has been proposed recently by Rahamat et.al [64]. This model developed in JAVA tries to manage functional and QoS requirements at runtime for service-oriented systems. They also developed a tool to allow dynamic reconfiguration of self-adaptive systems for services binding. DREWS consists of three processes for managing, selecting and performing reconfigurations of services. This technique may be considered a step toward describing SOA self-adaptation at runtime, but the model itself lacks formal grounds for modeling dynamism. Earlier a similar effort for modeling dynamic reconfiguration of SOA systems was made by Jose Luiz Fiaderio et al. [65] in 2013 . Their approach was much better in terms of services binding and simulating this configuration in ADL style. This work is based on mathematical modeling as a formal approach for SOA-based description relying on graph-based method using service models as confirmation rules. Execution states are considered when modeling dynamic reconfiguration which is changing as business activities variate. For a particular goal, services orchestration is simulated with sub-configurations of participating components defined in activity modules. Their key contribution is specification of required services discovery and binding along with services matching process. This approach does lack any specific tool support to analyze the dynamic architecture impact on QoS and eventually on SLAs.

To model dynamic open system architectures, Jiang wang et al. [66] presented an Internetware-based technique for evolutionary self-adaptive service-oriented systems. Internetware actually promotes SoS paradigm where different CSs participate, connected through networks to achieve the global mission of the larger. They considered services composition, context awareness and the combination of structural and behavioral evolutions of the systems and pointed out impact on QoS allowing simulation of dynamic architecture with support of analysis. Its focus is more on components and connectors and does not talk about services binding and or even their resulting configurations at runtime. In efforts to promote SOA-based modeling for large complex systems, authors in[34] developed a technique to simulate SOA-based self-adaptive dynamic architecture for cyber-physical systems. They incorporated MAPE-k technique to simulate architectural changes in IoT devices and services of connected components in CPS. But this approach clearly lacks describing components systems, reasoning about binding at runtime and impact on quality attributes. It is more focused to architecting an application which may have dynamics as it undergoes the operation but does not talk about behavioral aspects of those architectural changes and configurations. Table 4 is a summary of various architecture frameworks or mixed approaches for large dynamic service-oriented systems, especially SoS. The main focus area of these efforts is on dynamic services orchestration and choreography as well as services collaboration at design time. The interesting points to note are that some of these techniques do consider cloud environments when trying to model SOA-driven systems such as, SOAML in [53], Cloud ADL in [59] and SeaClouds in [14,15] considered cloud architectural aspects while modeling architecture of service-oriented systems.

4.3 MDE-based UML profiles

UML has been there for a while providing various types of architecture modeling profiles for systems and has remained a highly useful language for describing medium to enterprise and complex architectures. Recently with the emergence of MDE suitability for architectures modeling UML with its variant of SysML has gained attention in systems engineering community. These are not rigorously formal approaches as compared to ADLs but provide different tools support to model transformations in different phases of the systems. There have been various efforts like [67] to model service-oriented patterns for different systems using UML as a modeling language. Researchers made efforts to model SOA using model-driven approach where UML profiles (SOD-M) have been described at architectural level. Marcos et al. [68] used UML profile to model architecture by incorporating MDA paradigm. They defined UML profiles at PIM level for SOA modeling with a focus on structure and behavior of the system with its evolution. Meta-models were defined in UML profiles for PIM-SOA, but there is no explicit representation of services components to capture runtime dynamics. Moreover, model transformations have not been clearly defined for SOA. In another work by Muhammad et al. [69] an SOC UML profile for distributed systems was proposed by defining stereotypes. Meta-models and associated constraints were defined at lower levels to make profiles semantically rich with a main focus of development of the complex system in SOC paradigm. This technique models services providers, interfaces and description at a structural level along associated non-functional requirements. This technique lacks in

[20] https://www.omg.org/spec/SoaML/.

Table 4 Mixed architectural approaches for SOA Systems: +++ means fully supported; ++ means partially supported; – means not supported at all

Name of the technique	Static/structural services architecture model	Services binding and composition description at design time	Dynamic reconfiguration ability	Tool support
SOAML extension for Clouds	+++	+++	–	–
Cloud ADL	+++	++	–	–
DIARy	+++	++	++	++
SeaClouds approach	+++	–	++	–
ModaCloudsML	++	–	++	++
RigSoaML	+++	++	–	–
DREWS	+++	++	++	++

modeling dynamic runtime behavior of services for embedded system; moreover, model transformations to the targeted platform are not represented. Keeping with latest developments in service-oriented computing (SOC) especially to model RESTful services various modeling attempts were made in past in [68,70]. In [70] authors have modeled the dynamic behavior of RESTful services by extending the UML for large-scale enterprise applications. But this approach has certain limitations to represent dynamic behavior at runtime of the services composition and discovery for large systems as the language used has certain semantic constraints to reason about dynamic architecture.

5 Discussion

SoS architectures are dynamic and evolutionary requiring to model unexpected behaviors that occur at runtime. Due to emergent behavior and evolution, new coalitions/configurations are formed which are not known at design time but ideally should be modeled with specific scenarios to analyze impact on architecture. SOA provides dynamic services selection and composition for heterogeneous distributed systems, but it requires rigorous modeling notations to describe and reason about dynamism and evolution of SoS. Key challenges faced by service-oriented systems are services description for proper identification, dynamic runtime composition and services provisioning without compromising quality attributes. Based on developed taxonomy of SOA-SoS architecture modeling approaches, formal ADLs (Pi-ADL, Darwin and LEDA) show some promise to model distributed large systems to describe service components, interfaces and compositions statically and partially dynamically. These ADLs are not fully dynamic ADLs from SoS perspective as underlying process algebra (FSP, CSP) for these languages essentially handles static processes on communication channels. Some of the investigated ADLs do provide textual as well as graphical descriptions of services components and connectors, but ability to capture quality attributes is very limited; moreover, these old ADLs do not have rich language vocabulary to describe QoS concerns, i.e., failing to specify runtime quality attributes such as performance, security and reliability. As far as tool support is concerned, only DSOPL-ADL and AO-ADL have associated tools to transform architecture models into further development phases but these two ADLs are semi-formal and do not have expressive power to realize runtime services composition and discovery. Table 5 summarizes existing ADL capabilities to model architecture at the structural and behavioral level and their ability to describe distributed systems architecture reconfiguration at runtime.

In comparison with abstraction and explicit architecture representation, both formal and semiformal ADLs tend to be more abstract and lack explicit reasoning for structural and behavioral changes leading to dynamism where service composition and discovery are challenges due to the emergence and evolution of SoS CSs interactions at runtime. On the whole ADLs investigated tend to describe large distributed systems but these systems solely are single systems, whereas SoS is composed of many independent systems where interactions require mediated central control to manage volatility and avoid unexpected results requiring future ADLs to be equipped with rich syntax and semantics to orchestrate services dynamically.

UML with MDE combination tries to describe services at abstract level, but dynamic description is far from the UML capabilities as it has very informal notations.

In mixed approaches, we found a diverse range of techniques using MDE capabilities to provide solutions for SOA where MDE automates the modeling process and streamlines models transformation. It is noticeable that these techniques do consider cloud-based service system so SLAs and QoS are among key architectural aspects covered in these techniques. This category of architectural modeling considers SoS classes of systems to model to some extent as these are considered for large-scale complex systems. Among prominent of these are ModacClouds and SeaClouds which exhibit the capability to mode SOA statically as well as dynamically

Table 5 Comparison of ADLs for desired features of SOA-based systems architecture

SOA-ADLs	Services structural/ static modeling	Services dynamic modeling at design time	Dynamic services realization at runtime	Quality attributes considered
DSOPL-DAL [55]	Yes	Yes	Yes partially	No
X-ADL SOA EXTENSION [13]	Yes	Yes	No	No
Extension of DSOPL-ADL [55]	Yes	Yes	Yes partially	No
SO-ADL [47]	Yes	No	No	No
Z. Tao, S. Mei-e, Y. Shi, Y. Peng [11]	Yes	Yes	No	No
SOAML [12]	Yes	Yes	No	No
Pi-ADL [50]	Yes	Yes		No
3DxSOADL - xADL Extension for SOA [51]	Yes	No	No	No
DARWIN [12,45]	Yes	Yes	No	No
LEDA [24]	Yes	Yes	No	No

where former also comes with tool support for architectural analysis. Most of these architectural modeling techniques focus on dynamic service orientation at runtime where services are key components for such systems. Among this category two techniques rigSoaML and DREWS are significant efforts toward defining architecture dynamically where ADL is not primary architectural modeling tool.

The graph in Fig. 4 shows the gradual shift of techniques being used for modeling SOA-based systems, from ADLs to mixed approaches. ADLs have been there to describe SOA from the late 90s to date, and there is a huge debate about ADL-based models capabilities to describe service-oriented architectures statically and dynamically. Darwin (1996), LEDA (1999) and SOADL (2007) all used π–Calculus as formal semantics for modeling distributed systems dynamically, whereas from 2010 onwards, XML-based ADLs such as xADL, DSOPL-ADL and SOAML also emerged with semiformal semantics to model SOA-based systems, providing more flexibility and extensibility. But compared to the use of formal ADLs, there is a visible trend to use UML profiles in combination with MDE to model SOA systems from 2012 to 2017 mainly (SOC-UML & RESTful UML profiles) but these profiles do not have sufficient capabilities to orchestrate services dynamicity . However, mixed approaches from 2009 to 2017 and onwards, especially Cloud ADL (2013), SeaClouds & ModaCloudsML(2015), are significant works that have the capability to architect cloud-based service-oriented systems and are relatively more suitable to describe modern complex systems using both formal and semiformal notations. Different techniques surveyed above have their own pros and cons for modeling distributed complex systems, but SoS architectural characteristics [4,16,71] demand state-of-the-art rigorous modeling methods to deal with the complex architectural needs of the future systems.

6 Research issues

System architecture is one of the important artifacts. The overall quality of SoS is largely dependent on right decisions made during the architecture and design phase. Architecting SoS is a big challenge as these systems are volatile in nature, i.e., change their behavior according to changes in the environment where unknown behaviors emerge and CSs get added, deleted and updated while system is in operation. This phenomena of volatility pose a challenge to model such architectures keeping in view their dynamic behavior [72]. The fact that majority of SoS developed are heterogeneous coming across from different domains, their design and maintenance is also a challenge to system engineering community [73]. Ideally, SOA-SoS architecture model should be able to deal with the inherent characteristics of SoS, specify functional and non-functional properties at structural and behavioral levels. Existing SOA architectural modeling techniques are deficient to exhibit such abilities for SoS architecture. After detailed critical analysis we have identified following research issues in existing works:

1. *Inadequate Syntax and Semantics for SOA-SoS Models.* Although there exist different modeling languages and MDE-based approaches evaluated in Tables 3, 4 and 5 to describe SOA architecture for large-scale distributed systems, based on the findings these techniques have certain limitations to model systems with unique SoS architectural characteristics to model dynamic services. To describe the architectural elements of SoS as service components together with, their interactions and collaborations, formal syntax and semantics are needed to be defined for precise specifications and reasoning capabilities allowing architects to describe abstract architecture of CSs to be defined as service components with proper interfaces to collaborate at design time that establishes a baseline for dynamic services description at runtime.

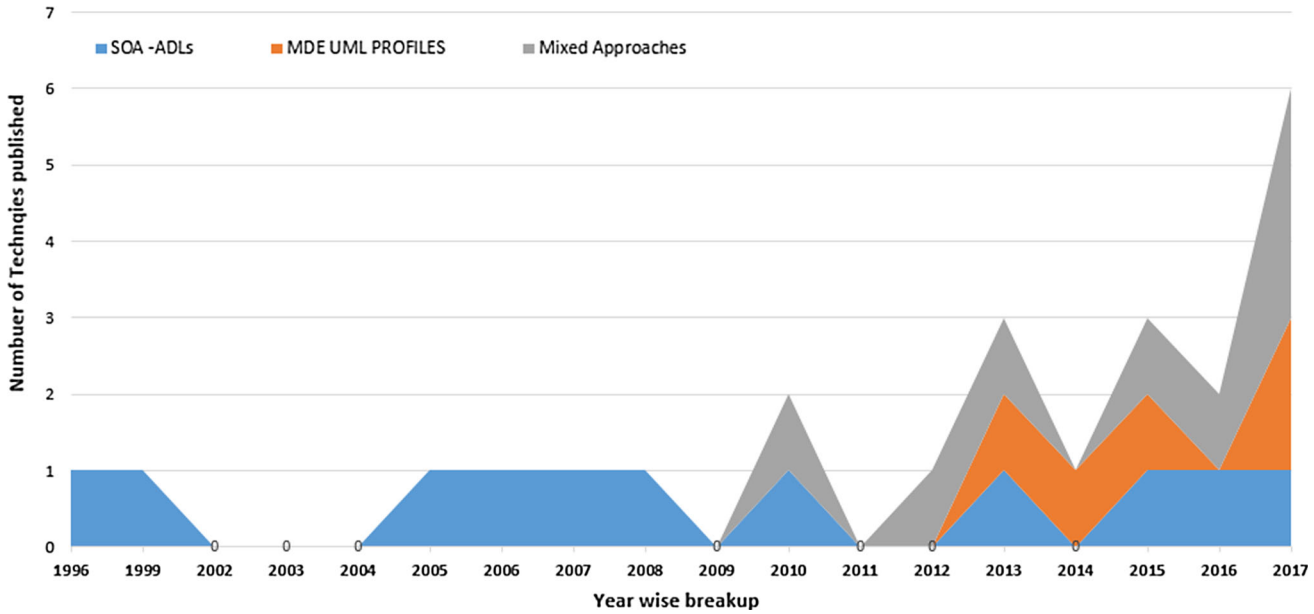

Fig. 4 Visualization of year-wise trend: SOA-SoS architecture modeling approaches

2. *Inability to specify and reason dynamic services at Design time.* Since SoS CSs are not completely known at design time and show emergent behavior leading to dynamism requiring architectural modeling methods to reason about dynamic architecture at design time, we found only a couple of formally well-defined ADLs like extended π–ADL in 2008, Darwin and LEDA which use process algebra of π-Calculus and try to address dynamic architecture at services level but dynamic services discovery and orchestration with respect to SoS is missing in these techniques. SysML profiles are only capable to describe the services description at design time but fail to realize the dynamic behavior at runtime. Services reconfiguration is vital when constituent service components are recomposed at runtime, and in this essence architecture model should be capable to perform services reconfiguration without effecting operations. Lastly these techniques do not have reasoning mechanism to model the dynamic services efficiently at design time and realize it at runtime for SoS.
3. *Inadequate tool Support for SOA-SoS Architecture Analysis.* SOA-SoS requires advance modeling and simulation tool set for performing better architectural analysis. In most of these architectural modeling techniques, the one most important shortcoming is that they lack sufficient tool support in order to analyze architecture from different perspectives. Although SOA-ADLs try to specify static and dynamic architectures, do not provide extensive tool support for architecture analysis. ModaCloudsML and SeaClouds in mixed approaches have some tool support, but this tool support is inadequate for analyzing the dynamics of SoS architecture.
4. *Inability to Specify QoS and SLAs for SOA-SOS.* The ability of constituent service components to provide and request services with interacting services components without violating QoS is crucial to achieving the global missions of SoS. Among investigated techniques, no technique try to give any modeling support either at a static or dynamic level to deal with changing QoS and SLAs. Further, these approaches fail to model architecture to align quality attributes with the emergent behavior of SoS resulting in system failure.

The majority of these architecture modeling techniques focus on single stand alone systems with design time configurations of components and connector but failing to specify and reason dynamic and evolutionary nature of SoS with respect to abstract services description that needs to be specified at design time and realized at runtime. SOA-based SoS architectures require to be designed and modeled from multi-viewpoints to fulfill various functional and non-functional requirements. SOA offers a great set of features to develop SoS systems, but it needs formal well-defined architecture modeling and simulation methods to deal with evolution and emergence that could enable designers to make better decisions and improve overall quality of the systems.

7 Conclusion and future research directions

The design of SoS brings additional challenges, the most crucial of which concern adaptability, integrability, interoperability, variability, dependability and security. The constituent systems should be available to share information

despite the heterogeneity of platforms. It is essential that common protocols and interfaces be conformed at the architectural level of SoS to ensure that CSs can interoperate cohesively at runtime and form configurations according to described requirements. For this, the SOA style(s) can provide the means for discovering, publishing and performing new capabilities at runtime while conforming to the laid requirements as it is more flexible and adaptable. Despite the emergence of very recent ADLs (SosADL, EAST-ADL, AADL and SysML) and model-driven frameworks (CML, DODAF,MODAF) for the SoS world, these techniques lack the expressive power to describe SoS with SOA perspective failing to cope with dynamicity and evolution at architectural level. Below are listed several key future research directions that could be pursued to overcome the existing limitations of architectural modeling techniques targeting SOA-SoS:

1. Need for abstract and concrete architectural descriptions for SOA-SoS based on formal syntax and semantics to describe architecture at design time.
2. Need for formal ADLs to describe SOA-SOS (description of service components of CSs and their external interfaces) with mediated control to form runtime services discovery and composition.
3. Based on identified gaps, ADLs should have the capability to model dynamic architecture from the behavioral viewpoint (service binding at runtime, dynamic services discovery, services orchestration and composition). For this SOA-SoS architecture must be modeled for expected changes related to emergent behavior and identify need for new services in the form of addition, deletion and replacement of the services.
4. ADLs must have the capability to describe quality attributes in terms of SLAs and QoS for SoS. ADLs must be able to ensure QoS for SoS due to dynamic reconfigurations in services composition for CSs and to ensure conformance to changing requirements.
5. Frameworks development to support formal ADLs for architecture reasoning about runtime changes and ensure QoS along with tool support to perform analysis to facilitate better architectural decision making.

To further strengthen SOA-based SoS adaptation, there is an urgent need to improve existing ADLs and architectural frameworks in line with SoS architectural characteristics. A combination of formal and informal approaches is required to specify SOA-SoS architecture statically as well as dynamically without compromising functional and non-functional requirements. These should model SOA-SoS architectures to provide multi-viewpoints for different stakeholders for decision making and ideally should support architectural descriptions at various stages of the SoS life cycle.

Of the various investigated approaches, ADLs show promise in regard to modeling SOA-based systems. However, to meet the requirements of SoS, these ADLs do not have capability to analyze and synthesize architecture models. Model-driven engineering (MDE)- and model- driven architecture (MDA)-based approaches to architect systems are gaining popularity because they provide automated analysis and rich tool support as compared to traditional architectural modeling approaches. SOA-ADLs can be strengthened by incorporating MDE approaches for model transformation, code generation and effective analysis. Along with this ADLs, meta-models can be provided strong semantics by incorporating ontological support. Therefore, a combination of ontologies and MDE can provide a way to build such ADLs which will allow system architects to model and simulate future SOA-SoS.

To conclude, this research survey investigated various architecture modeling techniques for emerging service-oriented SoS. To facilitate better understanding, this research article first described basic definitions, classifications and structural complexity of SoS. Further examples of SoS are provided with explanations, and importance of SOA-driven SoS has been elaborated. Different architectural modeling techniques were analyzed to determine their effectiveness in modeling SOA-based SoS. Our research concluded that there is a lack of effective alignment of SoS architectural characteristics toward services-oriented styles. It requires formal syntax and semantics, woven into frameworks with proven software tactics and industry practices to model dynamism and evolution in order to better address the SoS emergent behavior at architectural level.

References

1. Shaw M, DeLine R, Klein DV, Ross TL, Young DM, Zelesnik G (1995) Abstractions for software architecture and tools to support them. IEEE Trans Softw Eng
2. Shaw M, Garlan D (1996) Software architecture: perspectives on an emerging discipline, vol 1. Prentice Hall, Englewood Cliffs
3. Nielsen CB, Larsen PG, Fitzgerald J, Woodcock J, Peleska J (2015) Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput Surv (CSUR) 48(2):18
4. Oquendo F (2016) Software architecture challenges and emerging research in software-intensive systems-of-systems. In: Tekinerdogan B, Zdun U, Babar A (eds) Software architecture. Springer, Cham, pp 3–21
5. Jansen A, Bosch J (2005) Software architecture as a set of architectural design decisions. In: 5th working IEEE/IFIP conference on software architecture (WICSA'05), pp 109–120. https://doi.org/10.1109/WICSA.2005.61
6. Boardman J, Sauser B (2006) System of systems—the meaning of *of*. In: 2006 IEEE/SMC international conference on system of systems engineering, pp 6. https://doi.org/10.1109/SYSOSE.2006.1652284

7. Klein J, van Vliet H (2013) A systematic review of system-of-systems architecture research. In: Proceedings of the 9th international ACM Sigsoft conference on quality of software architectures, QoSA '13, pp 13–22. ACM, New York, NY, USA. https://doi.org/10.1145/2465478.2465490
8. Maier MW (1998) Architecting principles for systems-of-systems. Syst Eng J Int Counc Syst Eng 1(4):267–284
9. Oquendo F (2016) Formally describing the software architecture of systems-of-systems with sosadl. In: 2016 11th system of systems engineering conference (SoSE), pp 1–6
10. Woodcock J, Cavalcanti A, Fitzgerald J, Larsen P, Miyazawa A, Perry S (2012) Features of cml: a formal modelling language for systems of systems. In: 2012 7th International conference on system of systems engineering (SoSE), pp 1–6
11. Tao Z, Mei-e S, Shi Y, Peng Y, Zao-qing L (2005) Describing service-oriented architecture by extended darwin. Wuhan Univ J Nat Sci 10(6):971–976
12. Jia X, Ying S, Zhang T, Cao H, Xie D (2007) A new architecture description language for service-oriented architec. In: Sixth international conference on grid and cooperative computing (GCC 2007), pp 96–103. https://doi.org/10.1109/GCC.2007.18
13. Pannok P, Vatanawood W (2013) An xADL Extension for service oriented architecture design. In: 2013 International conference on information science and applications (ICISA), pp 1–3. Suwon
14. Zuniga-Prieto M, Gonzalez-Huerta J, Insfran E, Abrahao S (2018) Dynamic reconfiguration of cloud application architectures. Softw Pract Exp 48(2):327–344
15. Brogi A, Fazzolari M, Ibrahim A, Soldani J, Carrasco J, Cubo J, Durn F, Pimentel E, Nitto ED, Andria FD (2015) Adaptive management of applications across multiple clouds: The SeaClouds Approach. CLEI Electron J 18(1):2
16. Guessi M, Neto VVG, Bianchi T, Felizardo KR, Oquendo F, Nakagawa EY (2015) A systematic literature review on the description of software architectures for systems of systems. In: Proceedings of the 30th annual ACM symposium on applied computing, SAC '15, pp 1433–1440. ACM, New York, NY, USA. https://doi.org/10.1145/2695664.2695795
17. Mohammadi M, Mukhtar M (2013) A review of soa modeling approaches for enterprise information systems. Procedia Technology
18. Sinha K (2014) Structural Complexity and Its Implications for Design of Cyber Physical Systems. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA
19. Sammerville I (2016) Systems of systems in software engineering, 10th edn. Pearson Education limited, London
20. Sommerville I, Cliff D, Calinescu R, Keen J, Kelly T, Kwiatkowska M, Mcdermid J, Paige R (2012) Large-scale complex it systems. Commun ACM 55(7):71–77
21. Caffall D.S, Michael J.B (2005) Architectural framework for a system-of-systems. In: 2005 IEEE international conference on systems, man and cybernetics, vol 2, pp 1876–1881 Vol 2. https://doi.org/10.1109/ICSMC.2005.1571420
22. Thramboulidis K (2015) Service-oriented architecture in industrial automation systems—the case of IEC 61499: A Review. ArXiv e-prints
23. Vasilescu E, Mun SK (2006) Service oriented architecture (SOA) implications for large scale distributed health care enterprises. Distrib Diagn Home Healthc 2(1):91–94
24. Colombo AW, Karnouskos S, Bangemann T (2014) Towards the next generation of industrial cyber-physical systems. Springer, Cham, pp 1–22
25. Ingram C, Payne R, Perry S, Holt J, Hansen FO, Couto LD (2014) Modelling patterns for systems of systems architectures 2014: 146–153. https://doi.org/10.1109/SysCon.2014.6819249
26. Tsai Wt, Wei X, Paul R, Chung Jy, Huang Q, Chen Y (2007) Service-oriented system engineering (SOSE) and its applications to embedded system development. Serv Oriented Comput Appl 1(1):3–17
27. Bianco P, Kotermanski R, Merson P (2007) Evaluating a service-oriented architecture. Tech. Rep. CMU/SEI-2007-TR-015, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8443
28. Zhang Y, Liu X, Wang Z, Chen L (2012) A service-oriented method for system-of-systems requirements analysis and architecture design. ISSN 1796-217X **7**(2), 358
29. Kaur N, McLeod CS, Jain A, Harrisson R, Ahmad B, Colombo AW, Delsing J (2013) Design and simulation of a soa-based system of systems for automation in the residential sector. In: IEEE International conference on industrial technology: 25/02/2013-27/02/2013, pp 1976–1981. IEEE Communications Society
30. Stojanovic Z, Dahanayake A, Sol H (2004) Modeling and design of service-oriented architecture. In: 2004 IEEE International conference on systems, man and cybernetics. vol 5, pp 4147–4152. IEEE
31. Fiadeiro JL, Lopes A (2010) A model for dynamic reconfiguration in service-oriented architectures. In: European conference on software architecture, pp 70–85. Springer
32. Engelsberger M, Greiner T (2018) Dynamic reconfiguration of service-oriented resources in cyberphysical production systems by a process-independent approach with multiple criteria and multiple resource management operations. Future Gener Comput Syst 88:424–441
33. Srinivasulu P, Babu MS, Venkat R, Rajesh K (2017) Cloud service oriented architecture (CSoA) for agriculture through internet of things (IoT) and big data. In: Electrical, instrumentation and communication engineering (ICEICE) 2017, pp 1–6
34. Mohalik SK, Narendra NC, Badrinath R, Le DH (2017) Adaptive service-oriented architectures for cyber physical systems. In: 2017 IEEE symposium on service-oriented system engineering (SOSE, pp 57–62)
35. Clement SJ, McKee DW, Xu J (2017) Service-oriented reference architecture for smart cities. In: 2017 IEEE symposium on service-oriented system engineering (SOSE), pp 81–85. San Francisco, CA .41
36. Lewis G, Morris E, Simanta S, Smith D (2011) Service orientation and systems of systems. IEEE Softw 28(1):58–63
37. ISO/IEC/IEEE Systems and software engineering—architecture description. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) pp 1–46 (2011). https://doi.org/10.1109/IEEESTD.2011.6129467
38. Kruchten PB (1995) The 4+1 view model of architecture. IEEE Softw 12(6):42–50. https://doi.org/10.1109/52.469759
39. Bachmann F, Bass L, Clements P, Garlan D, Ivers J, Little M, Merson P, Nord R, Stafford J (2010) Documenting software architectures: views and beyond, 2nd edn. Addison-Wesley Professional, Boston
40. Papazoglou M (2008) Web services: Principles and technology. In: Pearson Education. Pearson pp 2, 14
41. Medvidovic N, Taylor RN (2000) A classification and comparison framework for software architecture description languages. IEEE Trans Softw Eng 26(1):70–93. https://doi.org/10.1109/32.825767
42. Ozkaya M, Kloukinas C (2013) Are we there yet? analyzing architecture description languages for formal analysis, usability, and realizability. In: 2013 39th EUROMICRO conference on software engineering and advanced applications (SEAA), pp 177–184. IEEE
43. Woods E, Hilliard R (2005) Architecture description languages in practice session report. In: null, pp 243–246. IEEE
44. Oquendo F (2008) Dynamic software architectures: formally modelling structure and behaviour with Pi-ADL. In: 2008 the third international conference on software engineering advances, pp 352–359. https://doi.org/10.1109/ICSEA.2008.47

45. Magee J, Kramer J (1996) Dynamic structure in software architectures. SIGSOFT Softw Eng Notes 21(6):3–14. https://doi.org/10.1145/250707.239104
46. Magee J, Kramer J (1999) State models and java programs. Wiley, Hoboken
47. Giannakopoulou D, Kramer J, Cheung SC (1999) Behaviour analysis of distributed systems using the tracta approach. Autom Softw Eng 6(1):7–35
48. Canal C, Pimentel E, Troya JM (1999) Specification and refinement of dynamic software architectures. Springer, Boston, pp 107–125. https://doi.org/10.1007/978-0-387-35563-4-7
49. Dan X, Shi Y, Tao Z, Xiang-Yang J, Zao-Qing L, Jun-Feng Y (2006) An Approach for Describing SOA. In: 2006 international conference on wireless communications, pp 1–4. Networking and Mobile Computing, Wuhan
50. Oquendo F (2008) Formal approach for the development of business processes in terms of service-oriented architectures using pi-adl. In: 2008 IEEE international symposium on service-oriented system engineering, pp 154–159. https://doi.org/10.1109/SOSE.2008.38
51. Miladi MN, Krichen I, Jmaiel M, Drira K (2010) An xADL extension for managing dynamic deployment in distributed service oriented architectures. In: Sirjani M, Arbab F (eds) Fundamentals of software engineering. FSEN 2009. Lecture notes in computer science, vol 5961. Springer, Berlin, Heidelberg
52. Dashofy EM, Hoek AVd, Taylor RN (2001) A highly-extensible, xml-based architecture description language. In: Proceedings of the working IEEE/IFIP conference on software architecture, WICSA '01, pp 103–. IEEE Computer Society
53. Zuiga-Prieto M, Insfran E, Abraho S (2016) Architecture description language for incremental integration of cloud services architectures. In: 2016 IEEE 10th international symposium on the maintenance and evolution of service-oriented and cloud-based environments (MESOCA), pp 16–23. https://doi.org/10.1109/MESOCA.2016.10
54. Adjoyan S, Seriai AD (2015) An architecture description language for dynamic service-oriented product lines. In: 27th Int. Conference on Software Engineering and Knowledge Engineering, Pittsburgh, United States, pp 1–6
55. Adjoyan S, Seriai AD (2017) Reconfigurable service-based architecture based on variability description. In: Proceedings of the ACM symposium on applied computing, vol Part F1280, pp 1154–1161. https://doi.org/10.1145/3019612.3019767
56. Pinto M, Fuentes L, Troya JM (2011) Specifying aspect-oriented architectures In ao-adl. Information and software technology
57. Hoare CAR (1978) Communicating sequential processes. Commun ACM 21(8):666–677
58. Uviase O, Kotonya G (2018) Iot architectural framework: connection and integration framework for iot systems. arXiv preprint arXiv:1803.04780
59. Perez AN, Rumpe B (2013) Modeling cloud architectures as interactive systems. In: 2nd International workshop on model-driven engineering for high performance and cloud computing, pp 15–24
60. Haber A, Ringert JO, Rumpe B (2014) Montiarc-architectural modeling of interactive distributed and cyber-physical systems. arXiv preprint arXiv:1409.6578
61. Lushpenko M, Ferry N, Song H, Chauvel F, Solberg A (2015) Using adaptation plans to control the behavior of models@ runtime. In MoDELS@ Run pp 11–20
62. Ferry N, Chauvel F, Song H, Rossini A, Lushpenko M, Solberg A (2018) CloudMF: model-driven management of multi-cloud applications. ACM Trans Internet Technol 18(2):16:1–16:24. https://doi.org/10.1145/3125621
63. Becker B (2014) Architectural modelling and verification of open service-oriented systems of systems. Doctoral thesis, Universität Potsdam
64. Ilahi R, Admodisastro N, Ali NM, Sultan ABM (2017) Dynamic reconfiguration of web service in service-oriented architecture. Adv Sci Lett 23(11):11553–11557
65. Fiadeiro JL, Lopes A (2013) A model for dynamic reconfiguration in service-oriented architectures. Softw Syst Model 12(2):349–367. https://doi.org/10.1007/s10270-012-0236-1
66. Wang J, Peng Q, Hu X (2014) A modeling: Internetware-based dynamic architecture evolution applying to soa. In: Proceedings of the 2014 IEEE 18th international conference on computer supported cooperative work in design (CSCWD), pp 100–105. https://doi.org/10.1109/CSCWD.2014.6846824
67. Amir R, Zeid A (2004) An UML profile for service oriented architectures. In: Companion to the 19th annual ACM SIGPLAN conference on object-oriented programming, pp 192–193. Systems, languages, and applications, OOPSLA 2004
68. Schreier S (2011) Modeling restful applications. In: Proceedings of the second international workshop on RESTful design, WS-REST '11, pp 15–21. ACM, New York, NY, USA. https://doi.org/10.1145/1967428.1967434
69. Muhammad WA, Radziah M, Dayang NAJ (2012) Soa4derts: a service-oriented uml profile for distributed embedded real-time systems. In: 2012 IEEE symposium on computers informatics (ISCI), pp 64–69. https://doi.org/10.1109/ISCI.2012.6222668
70. Rathod DM, Parikh SM, Buddhadev BV (2013) Structural and behavioral modeling of restful web service interface using uml. In: 2013 International conference on intelligent systems and signal processing (ISSP), pp 28–33. https://doi.org/10.1109/ISSP.2013.6526869
71. MacKenzie CM, Laskey K, McCabe F, Brown PF, Metz R, Hamilton BA (2006) Reference model for service oriented architecture 1.0. OASIS standard 12
72. Levine L, Morris EJ, Place PR, Plakosh D (2004) Sosi: system of systems interoperability. Tech. rep., Software Engineering Institute, Carnegie Mellon University. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20698
73. Sage AP, Cuppan CD (2001) On the systems engineering and management of systems of systems and federations of systems. Inf Knowl Syst Manag 2(4):325–345
74. Sommerville I, Cliff D, Calinescu R, Keen J, Kelly T, Kwiatkowska M, Mcdermid J, Paige R (2012) Large-scale complex it systems. Commun ACM 55(7):71–77. https://doi.org/10.1145/2209249.2209268 References [74, 75, 76] was provided in the reference list; however, this was not mentioned or cited in the manuscript. As a rule, if a citation is present in the text, then it should be present in the list. Please provide the location of where to insert the reference citation in the main body text. Kindly ensure that all references are cited in ascending numerical order
75. Uviase O (2016) IoT architectural framework: connection and integration framework for IoT systems In Proceedings ALP4IoT 2017M. In: Architecture description language for incremental integration of cloud services architectures, pp 16–23. Zuniga-Prieto, E Insfran and S Abrahao, 2016 IEEE 10th international symposium on the maintenance and evolution of service-oriented and cloud-based environments (MESOCA), Raleigh, NC
76. Zhang W, Zhang L (2015) Designing and modeling cyber physical systems by a service-based approach. In: 2015 6th IEEE international conference on software engineering and service science (ICSESS), pp 668–671