Garcés *et al. Journal of the Brazilian Computer Society* (2019) 25:8
https://doi.org/10.1186/s13173-019-0089-3

Journal of the Brazilian Computer Society

RESEARCH Open Access

Software mediators as first-class entities of systems-of-systems software architectures

Lina Garcés[1,2*], Flavio Oquendo[2] and Elisa Yumi Nakagawa[1]

Abstract

Context: In contrast to traditional software systems that are mostly created from scratch, current software systems are engineered as a junction of systems already in operation. Examples can be found in domains, such as smart cities, crisis and emergency, IoT, big data, industry 4.0, and connected health systems. Most of them can be considered systems-of-systems (SoS), since they refer to alliances of operational and managerial independent software-intensive systems, which are sometimes distributed over different environments. Therefore, SoS software architectures must be dynamic, evolve over time, and support the execution of emergent behaviors to accomplish SoS missions. They must be also designed to enable the connection of heterogeneous systems, making possible their interoperation, communication, coordination, cooperation, and collaboration, most of the times, in a seamless way. Similar challenges have been addressed by using software mediators as architectural entities. However, the application of mediators in SoS has not been properly explored.

Goal: This article introduces *MediArch*, a layered architecture that considers mediators as first-class software entities to be used in the construction of SoS architectures. Our objective is to present evidence about how *MediArch*'s can support the construction of SoS architectures.

Methods: The following four steps were conducted to define *MediArch*: (1) identification of mediation requirements to allow SoS properties; (2) establishment and categorization of twelve types of mediators, for enabling capabilities of communication and control of constituent systems interactions and conversion of heterogeneous messages exchanged through a mediation infrastructure; (3) specification of duties, behaviors, assumptions, and guarantees of mediators; and (4) organization of *MediArch* in three layers, namely, the constituents & consumer systems layer; the communication, conversion, & coordination layer; and the control layer. This architecture was used as the backbone for the software architectures of two SoS in different domains, namely, flood monitoring system-of-systems (FMSoS), and health-care supportive home system-of-systems (HSH-SoS), for providing evidence on how *MediArch* supports the architecting process of SoS.

Results: *MediArch* (1) supports the integration of independent constituent systems, (2) provides strategies to manage emergent behaviors, (3) defines different schemes of control authorities, (4) offers elements to support SoS evolution, and (5) promotes the resilience and adaptability of SoS architectures.

Conclusions: Although *MediArch* contributes to the establishment of SoS architectures, some challenges related to performance, resource consumption, security, safety, and non-disruptive reconfigurations must still be overcome.

Keywords: Software architecture, Systems-of-systems, Mediator, Connector, Crisis and emergency systems, Health-care supportive home systems

*Correspondence: linamgr@icmc.usp.br
[1]Department of Computer Systems, University of São Paulo, Av. Trabalhador São-carlense, 400 - Centro, 13566-590 São Carlos, Brazil
[2]IRISA - University of Southern Brittany, Rue Yves Mainguy, 56000 Vannes, France

Introduction

Software systems have grown in complexity and size and, therefore, require paradigms for their engineering, specifically for their architecting. Modern systems are conceived to execute composed functionalities that depend on interactions among heterogeneous and distributed software-intensive systems. The construction of these systems must contemplate a diversity of stakeholders and multidisciplinary practitioners and must consider the establishment of requirements and behaviors at early stages of their development, as well as during operation. In this context, a type of complex, software-intensive, and large systems has emerged and has been known as systems-of-systems (SoS).

SoS are comprised of other software systems (sometimes in operation) that are heterogeneous, distributed, and managerially and operationally independent of the SoS control. Behaviors of SoS emerge from the interaction among constituent systems; therefore, SoS missions cannot be achieved by any constituent alone, but through their cooperation [46, 52]. In short, missions are systems' activities to pursue stakeholders' goals [7]. In this perspective, software architectures of SoS are considered dynamic, since alliances of constituent systems are modified, sometimes seamlessly at runtime, depending on the SoS missions, constituent availability, and environmental settings [46, 52]. This scenario raises important challenges for the design of SoS architectures.

Software architects must deal with interoperability issues that arise during the dynamic exchange of data among constituent systems. This is mostly due to differences in protocols, data formats, workflows, and interfaces, which are usually neither well defined, nor documented. Interoperability problems have been presented in many types of software systems over the years and are still an open issue. For SoS, interoperability is a critical requirement, since it directly impacts on their capabilities of evolution, dynamicity, and mission achievement.

The software engineering community has conceived the mediator concept as a solution to deal with the integration of heterogeneous and distributed data sources [70]. In a general way, mediators have been used as software entities responsible for achieving interoperability among diverse and disperse software components through the translation of exchanged data and coordination of systems' interactions [72, 73]. In the SoS context, mediators have been proposed as architectural elements that connect constituent systems to enable their communication, coordination, cooperation, and collaboration [54].

Important advances in software connectors, mediators, emergent middleware, software platforms, and frameworks have been made to overcome interoperability problems in complex and large-scale systems [3, 8, 25, 36, 40, 42–44, 47, 62, 63]. However, existing approaches do not consider strategies to enable all interoperability levels, i.e., technical, semantic, syntactic, and organizational, neither other important issues of SoS architectures [32], such as dynamic integration of constituent systems, incorporation of new SoS missions or requirements, prediction of emergent behaviors, decision making, and resilience.

This article presents *MediArch*, a three-layered architecture that establishes mediators as first-class software entities to construct SoS. Mediators are located at the two higher layers and aim to offer capabilities of communication, conversion, coordination, and control. *MediArch* was used as a backbone for the design of two SoS, namely, FMSoS (Flood Monitoring SoS) and HSH-SoS (Healthcare Supportive Home SoS). Both SoS gave us evidence to conclude that *MediArch* advances the state-of-the-art on SoS architecting, since it (1) allows the integration of independent constituent systems, (2) provides strategies to control emergent behaviors, (3) defines strategies of control authorities, (4) gives structures for SoS evolution, and (5) promotes resilience and adaptability of SoS architectures.

The remainder of this article is structured as follows. The "Background" section presents the theoretical background used in this work. The "Materials and methods" section details the methods used to establish and assess *MediArch*. The "Software mediators for SoS" section describes *MediArch* and specifies all software mediators that compose this layered architecture. The "Evaluation of *MediArch*" section reports results obtained during *MediArch* assessment. The "Discussion" section discusses the results of constructing FMSoS and HSHSoS based on *MediArch*. In the same section, related works (the "Related work" section), contributions of *MediArch* (the "*MediArch* contributions" section), limitations of our work (the "Limitations and directions for future studies" section), and threats to validity (the "Threats to validity" section) are described. Finally, conclusions and future work are listed in the "Conclusions and future works" section.

Background

This section presents the theoretical foundation containing the main topics embraced in this work, namely, SoS and software mediators.

Systems-of-systems

SoS are systems whose constituent elements are systems themselves (e.g., software-intensive systems, information systems, embedded systems, ultra-large systems). Constituent systems collaborate among them to achieve high-level missions that cannot be addressed by any system independently. In short, missions are systems' activities to pursue stakeholders' goals [7]. For instance, a mission for

the Apollo 12 system is [7] "to perform inspection, survey, and sampling in lunar mare area."

A system can be considered an SoS if it presents the following characteristics [46, 52]:

- *Operational independence.* Constituent systems are independent and able to operate even when the SoS is disassembled; hence, constituents must be low coupled without harnessing operations of their peers;
- *Managerial independence.* Constituent systems are governed by their own rules, rather than by external ones when they participate in an SoS. This characteristic raises challenges for the creation of SoS, due to uncertainties on constituents operations reliability, which can negatively or positively impact SoS behaviors and the achievement of their missions.
- *Distribution.* Constituent systems are dispersed; therefore, a type of mediator is required to allow the communication of their operations results with other constituents and, depending on the type of SoS, with the central authority. Mediators must enable communication, coordination, and (if required) translation of messages to support inter-operations between constituents and the SoS.
- *Evolutionary development.* SoS can be under constant changes due to modifications in missions, e.g., market tendencies can change business strategies, which results in changes of missions, and inclusion of new constituents or removal of those that are not longer required.
- *Emergent behavior.* An SoS behavior emerges as a result of the synergistic collaboration of its constituent systems. An emergent behavior addresses a global mission, rather than individual missions accomplished by the constituent systems separately. Depending on the complexity to characterize, measure, and predict emergent behaviors, an SoS can be modeled as deterministic or stochastic systems [48].

Depending on the independence level of constituents, an SoS can be classified as follows [15, 38]: (1) *directed*, if constituent systems are controlled by a central authority to satisfy the SoS missions; (2) *acknowledged*, if constituent systems maintain independent management and missions, but collaborate with the SoS to achieve its missions; (3) *collaborative*, when constituent systems are not forced to follow a central management, but voluntarily collaborate to achieve the SoS missions; and (iv) *virtual*, if the SoS has neither a central authority, nor clear missions, its constituent systems are unaware of their participation in the SoS; therefore, its behaviors are highly emergent.

SoS characteristics and types impose the following properties to their architectures [31, 32, 52]: (a) continuous integration of constituent systems and capabilities, as required by the SoS to achieve its missions; (b) definition of design decisions to assure predicted emergent behaviors [48]; (c) establishment of decision-making authorities that control SoS operations and govern (if possible) alliances of constituent systems; (d) strategies for SoS evolution, such as new missions, requirements, and other significant changes appear; and (e) resilience and adaptability of SoS operations for ensuring desired quality levels and avoiding unexpected emergent behaviors.

Characteristics of distribution and operational and managerial independence of constituent systems hamper the construction of SoS, mainly due to interoperability barriers. In SoS, interoperability can be defined as the ability of constituent systems alliances to exchange specified state and data and interact following a commanding authority or predefined agreements [49]. The following four levels of interoperability have been considered in distributed software systems, and also are required in SoS:

- *Technical interoperability*, related to establish links and transmit data among constituent systems. It is domain-independent and does not know about the meaning of data exchanged [9]. Therefore, technical interoperability only guarantees the correct transmission of bits, but it does not communicate anything about the meaning of these bits and what they represent [41]. It is often focused on communication protocols and infrastructure necessary for the protocols to operate.
- *Syntactic interoperability*, related to the syntax that defines rules on the type of data exchanged, structures, and organization. It provides data formats, well-defined syntax, and encoding (e.g., message content structure, size of headers, size of the message body, and fields contained into a message) [41].
- *Semantic interoperability*, related to semantics that defines the exact meaning of the data exchanged (e.g., clear definitions of domain terminology or vocabulary). Data are conceived, in the semantic layer, as information to be shared, processed, and well-understood (with no ambiguity) by systems. Semantic interoperability is specific to domain and context and requires the use of unambiguous codes and identifiers [9].
- *Organizational interoperability*, concerned with the interaction of multiple organizations to obtain their respective business goals [11]. It is focused on the coordination of independent and distributed workflows and activities that are well understood by constituent systems, their organizations, and users. Organizations participating in a business process, at SoS level, must commit to perform several activities

(as part of their own workflows), and such commitments are specified in contracts.

Software mediator

Issues on interoperability among software entities (e.g., components, systems, services), such as exchanged data and behavioral heterogeneity, have been overcome through intermediary software entities called mediators [36]. Mediators perform the necessary translations of the data exchanged and appropriately coordinate interactions among software entities [71]. Similarly to connectors in software architectures, mediators can be seen as first-class software entities, that specify their internal behaviors, protocols, duties, commitments, and roles, making possible the definition of mediators patterns to be reused by software architects in different software projects [23, 62].

Because of their benefits, mediators have been studied in other types of complex systems, for instance, in services-based systems on distributed environments. Several types of mediators (some of them considered as patterns) [8, 36, 43, 62, 63] to improve interoperation among heterogeneous components and services.

In an SoS perspective, mediators are considered architectural elements that connect constituent systems to allow their communication, coordination, cooperation, and collaboration [54]. SoS mediators must be defined in an abstract way and instantiated at run-time, for enabling SoS emergent behaviors and missions accomplishment [55]. In contrast to software connectors, which act as communication channels to transfer data or control among constituents, mediators have communication, coordination, and conversion roles [55] and facilitate SoS control over dependent entities. Moreover, since they can be created or destroyed by the SoS at run-time, they must be under SoS management.

Figure 1 shows a representation of an SoS mediator used in this work. In its simple form, a mediator is configured to receive input data (through its *fromEx* interface) and send output data (through its *toEy* interface).

A mediator has a *gate* that provides an interface (i.e., interaction points) among systems (either SoS, or their constituents) and their local environment. Each gate specifies *duties* (or obligations) to be fulfilled by the entity connected with the gate. A gate is governed by a *protocol* that defines a set of rules to guide interactions among the interconnected entities through the gate. The gate also defines some *assumptions* and *guarantees* on behaviors, properties, and environment of the entity connected through the mediator's gate. Therefore, the interested entity must provide a respective gate that *commits* to offer duties specified in the mediator's gate towards a successful interaction. Most of the times, *commitments* or contracts are dynamically created at run-time to achieve *assumptions* and *guarantees* specified in the gates of the mediator and the other software entity (e.g., constituent system or another mediator) interested in being connected. More details about mediators' internal composition and formal specification can be found in [54].

Fig. 1 General representation of a mediator. Source [23]

Materials and methods

This section presents the methods used for the construction and assessment of *MediArch*, a mediators-oriented architecture for SoS.

Identification of mediation requirements in SoS

As an initial step, the literature on SoS fundamentals [46, 52] was read and SoS properties proposed in [32] were studied. Important requirements of quality attributes for each SoS property were defined to understand architectural challenges that mediators must address. The mapping between properties (P) [32] and quality attributes requirements of SoS [59] are detailed in the first two columns of Table 1.

Additionally, we studied architectural strategies to address both SoS' quality attributes requirements and properties. We searched for architectural patterns and styles used for constructing SoS architectures. For this, the search string *("architectural pattern?" OR "architectural style?") AND ("System-of-Systems" OR "Systems-of-Systems" OR "System of Systems")* was executed in data libraries Scopus and Google Scholar. Such libraries were selected since they index research published in other important computer science academic libraries, such as ACM, SpringerLink, and IEEEXplorer. The search was limited to Title-Abstract-Keywords. Studies [5, 12, 13, 31, 32, 39, 50, 53, 57, 58, 69] were returned, and, to the best of our knowledge, they are the only studies on architectural patterns and styles for SoS architectures. We also looked for architectural patterns and styles for self-organizing systems, since they share characteristics of dynamism and emergent behaviors with SoS [16, 53, 56]. Architectural patterns and styles related to quality attributes and properties of SoS are shown in the third column of Table 1.

Furthermore, research on software connectors [3, 25, 40, 42, 44, 47] and software mediators [8, 36, 43, 62, 63] were analyzed to understand the concept and capabilities of software entities responsible for mediating interactions in, and among, software systems. Based on that

Table 1 Relationship among properties (P), quality attributes, architectural patterns and styles, and mediation categories in SoS

SoS properties	Quality attributes	Architectural patterns/styles	Mediator categories
P1—continuous accommodation or integration of constituent systems.	Cooperation, integration, interoperability (i.e., technical, semantic, and syntactic), portability, and flexibility of constituents, and low-coupling between constituents and the SoS	Centralized architecture, publish-subscribe, contract monitoring, ESB, trickle-up, pace layering, MAPE-K, share data, broker, and observer-controller	Communication and conversion
P2—assurance of emergent behaviors	Coordination of constituents' capabilities, organizational interoperability, and situation-aware	Centralized architecture, SOA, supply chain, reconfiguration control architecture, contract monitor, trickle-up, evolution styles, reflection, MAPE-K, observer-controller, and master-slave	Coordination, conversion, and control
P3—necessity of a decision-making or control authority	Centralized, decentralized, or full-decentralized control, and accountability of decisions	Centralized architecture, SOA, publish-subscribe, pipe-filter, supply chain, reconfiguration control architecture, infrastructure grid, trickle-up, MAPE-K, shared-data, observer-controller, and master-slave	Control
P4—evolutionary development.	Maintainability, modifiability, separation of concerns, adaptability, portability, low-coupling, flexibility, scalability, integration, and interoperability	SOA, publish-subscribe, supply chain, infrastructure grid, ESB, pace layering, evolution styles, reflection, and MAPE-K	Coordination, conversion, control
P5—resilience or adaptability of SoS architecture (e.g., dynamic reconfigurations)	Adaptability, fault-tolerance, self-organization, robustness, dynamic configurations, availability, and security	SOA, publish-subscribe, blackboard, pipe-filter, supply chain, reconfiguration control architecture, ESB, pace layering, evolution styles, reflection, and MAPE-K	Coordination and control

understanding, we abstracted mediation capabilities presented in architectural patterns and styles used in SoS architectures.

Categorization of mediators

Similar mediation capabilities (identified in architectural patterns and styles) were grouped to define types of mediation. Three categories of mediators, namely, communication & coordination, conversion, and control, were defined as important for SoS architectures. The last column of Table 1 shows them and their relationship with SoS properties, quality attributes, and architectural patterns/styles.

Specification of mediators

The description of each mediator was based on specifications proposed in SosADL, an ADL (Architectural Description Language) defined to systematically represent software architectures of SoS [55]. SosADL proposes the following structure for the specification of mediators [55]:

- Duties, which are obligations to be fulfilled by mediator's gates when the mediator is connected to a constituent system through an interface
- Behaviors, which are capabilities offered by the mediator to satisfy duties specified in the interface
- Assumptions, which are conjectures about the interface offered by the mediator for the desired behavior
- Guarantees, which are assertions about the results of the mediator's behavior to satisfy the duties specified in the interface.

Definition of *MediArch*

Since SoS architectures must be conceived to address a continuous integration of constituent systems, dynamic reconfigurations, evolution, scalability, and resilience, mediators must be organized towards satisfying such requirements. Therefore, they must be structured in a way they can be created, composed, removed, modified, and reconfigured at without affecting SoS behaviors or constituent systems operations.

Interactions among mediators and dependencies among their behaviors were identified for the establishment of mediators organization in SoS architectures. Categories of mediators were organized in a three-layered architecture, called *MediArch*, detailed in the "Software mediators for SoS" section. The structure of this mediators-oriented architecture was based on the architectural pattern of *pace layering*, which enables the construction of complex behaviors through layer hierarchies [57]. In this pattern, lower layers implement fast adaptations (e.g., reconfigurations of constituent systems) and support interoperability of constituent systems, and higher layers are responsible for time-demanding adaptations (i.e., selection of the best policy or plan for reconfiguring the SoS) and address reliability requirements [57].

Evaluation of *MediArch*

To obtain evidence of the applicability of *MediArch*, the mediators contained in this architecture were used as first-class software entities for the construction of the software architecture of two SoS, namely, FMSoS (Flood Monitoring SoS) and HSH-SoS (Healthcare Supportive Home SoS). The FMSoS is an SoS whose main missions are the continuous monitoring of rivers in a city and emergency situations detection (e.g., floods). The HSH-SoS' mission is to assist patients diagnosed with chronic diseases (e.g., diabetes mellitus) in the management of their conditions during their daily life activities. Both SoS have been a focus of research of our group over the past years, and their mediators-based architectures are detailed in the "Evaluation of *MediArch*" section.

The creation of FMSoS and HSH-SoS architectures followed the guidelines for architecting software systems in [29], using *MediArch* as backbone. The mediators-oriented architectures of FMSoS and HSH-SoS were assessed regarding their feasibility to allow SoS properties and their related quality attributes described in the two first columns of Table 1. For this, the following research questions were investigated:

RQ1 - How do SoS architectures based on MediArch enable continuous accommodation or integration of constituent systems?

RQ2 - How do SoS architectures based on *MediArch* assure emergent behaviors?

RQ3 - How can different strategies of control authorities be established in SoS architectures based on *MediArch*?

RQ4 - How can SoS architectures based on *MediArch* evolve?

RQ5 - How do SoS architectures based on *MediArch* address resilience or adaptability?

Software mediators for SoS

Figure 2a illustrates *MediArch*, a three-layered architecture that provides a general structure to design SoS architectures for SoS considering mediators as first-class software entities. Mediators in *MediArch* were categorized into three types, namely, communication, conversion, and control. Altogether, they support different capabilities required during SoS operations, such as information exchange, request and reply messages transmission, coordination of capabilities, conversion of messages, and reconfigurations, among others. Mediators allow interactions between the following software entities: (a) constituent and constituent, (b) SoS and constituent, (c) constituent and SoS, (d) consumers and SoS, (e) SoS and consumers, and (f) mediators and mediators. For the sake of clarity and simplicity, constituent systems, consumer systems, and mediators are referred to as software entities, i.e., *E*, that can be mediated.

MediArch was defined following the *pace layering* pattern [57] and is composed of a hierarchy of mediators that establishes dependencies among all type of software entities found in SoS, as illustrated in Fig. 2. Therefore, mediators in upper layers depend on data coming from entities in lower layers. Mediators in the *control layer* use data provided by mediators in the *communication, conversion, & coordination layer* to define the current status of the SoS and establish, for instance, emergency and reconfiguration plans. Similarly, this middle layer depends on

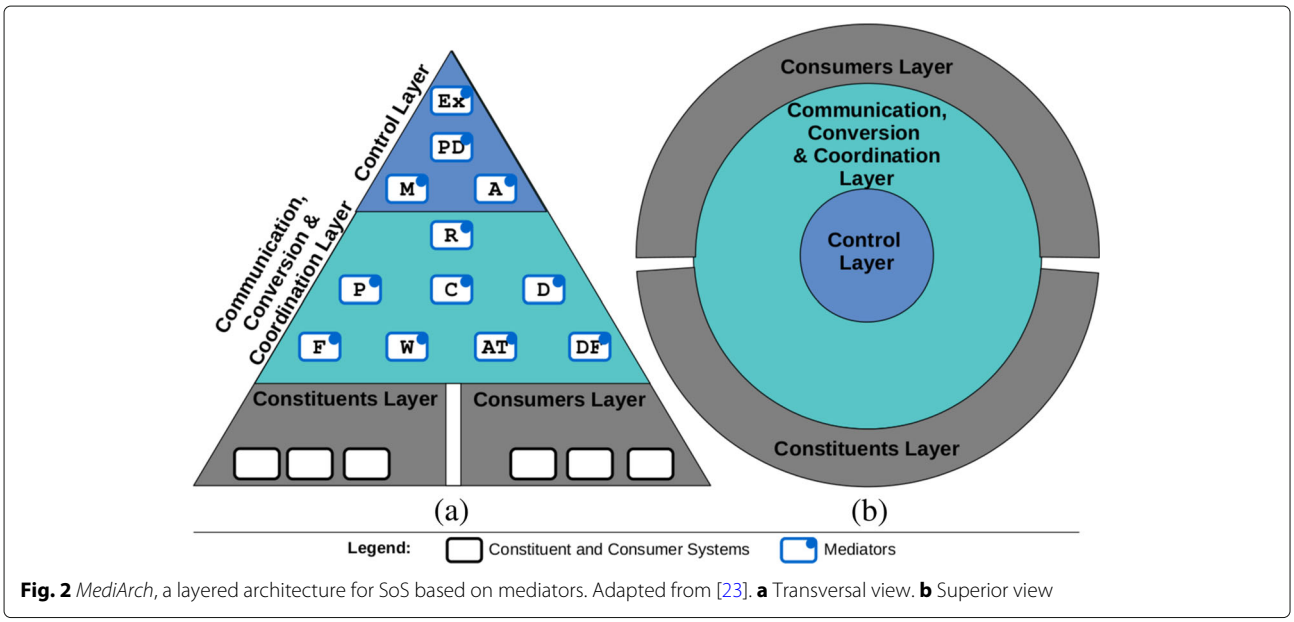

Fig. 2 *MediArch*, a layered architecture for SoS based on mediators. Adapted from [23]. **a** Transversal view. **b** Superior view

data supplied by the *constituent and consumer systems layers* to compose new information that will allow the execution of emergent behaviors and, thus, to enable the accomplishment of SoS missions. Moreover, the mediators' hierarchy in *MediArch* allows SoS managers to specify reconfigurations commands at different levels. In this context, mediators in upper layers can define, propose, and send emergency and reconfigurations plans. Those plans detail which types of modifications or actions (preventive or corrective actions) to be executed by entities in lower layers.

Elements in *MediArch* are described in the remainder of this section.

Constituents and consumers layers

The lowest layer of *MediArch* is composed of constituents and consumers systems. Constituents are important systems for SoS operations. Consumers are systems that interact with the SoS, utilizing results of SoS operations, which are the result of interactions among constituents and SoS controller.

As shown at the right side of Fig. 2, although both types of systems (i.e., consumer and constituent systems) are in the same level, they do not communicate directly while participating of the SoS. This is mainly because the information required by consumers can only be produced during interactions between constituent systems and through the transformations and communication of exchanged data by SoS mediators. Such systems are out of the SoS control, operationally and managerially independent, and distributed through the environment. Moreover, their interfaces, transferred data, data formats, and technologies are heterogeneous.

Communication, conversion, & coordination layer

This layer is the intermediate layer between constituent and consumer systems, and also mediators in the control layer. Mediators, in this layer, are concerned with interoperability, communication, and coordination issues. All systems participating in the SoS (i.e., consumers and constituents) only can request and exchange interoperable information through mediators in this layer. Mediators for communication and conversion purposes are explained as follows. For each mediator, it is offered its general description, duties, behaviors, assumptions, and guarantees, as proposed in [54], and detailed in "Software mediator" section.

Communication & coordination mediators

Mediators for communication allow (synchronous or asynchronous) transmission of data, messages, events, or operation results among entities E_i, e.g., constituent systems, consumers systems, and software mediators, and coordination of interactions among SoS entities. Four mediators for communication & coordination purposes, namely, pipe, collaborator, distributor, and router are defined in this intermediary layer, as showed in Fig. 3a–d. They were abstracted from architectural patterns/styles pipe-filter, publish-subscribe, observer-controller, and pace layering, which are some approaches used for communication & coordination purposes in SoS [32, 39, 57], as shown in Table 1.

Pipe (P) Communicates data d from E_1 to E_2. It is an unidirectional transfer, as shown in Fig. 3 a. Pipes can be used when an SoS just receives output messages from constituent systems, but no request can be sent to their

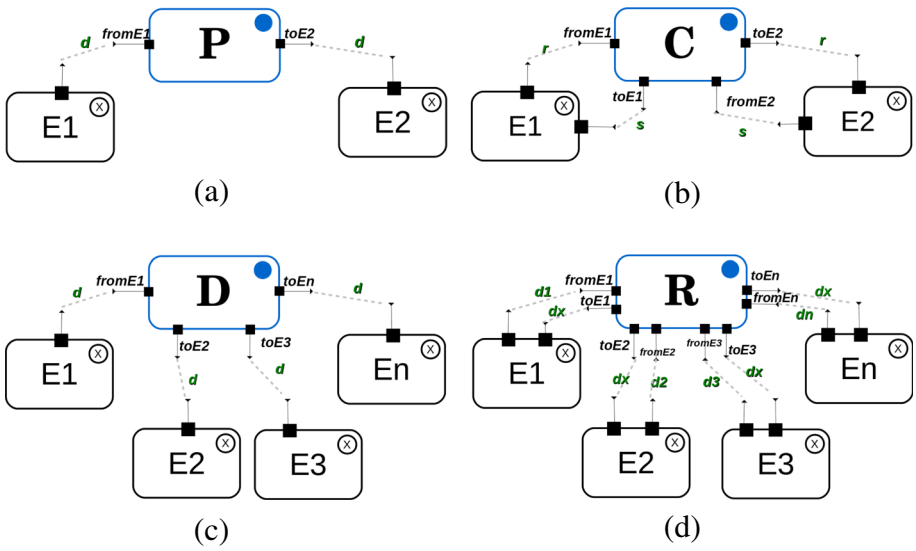

Fig. 3 Communication mediators. Adapted from [23]. **a** Pipe. **b** Collaborator. **c** Distributor. **d** Router

constituents. Pipes can also be used in some virtual SoS, in which no central control exists to request data from constituent systems, which are not aware of their participation in the SoS.

Specifications:

- Duties: To transmit d from E_1 to E_2.
- Behaviors: $Transmit(d)$ using $output$ interface in E_1 to $input$ interface in E_2.
- Assumptions: E_1 and E_2 interfaces are known, and d can be processed or understood by E_2.
- Guarantees: d is transmitted to E_2.

Collaborator (C) Used to request/supply services/data between entities, E_1 and E_2, as illustrated in Fig. 3b. Collaborators can be used in SoS with certain control level over their constituent systems, e.g., in directed or acknowledged SoS.

Specifications:

- Duties: To process requests and provide responses.
- Behaviors: $requestToE2(r)$, $supplyToE1(s)$, where r is the requested service or data from E_1 to E_2, and s is the response to that request from E_2 to E_1.
- Assumptions: Requested services are provided by suppliers and answers can be understood by requesters.
- Guarantees: Synchronous/asynchronous delivery of requests and answers. Buffering is necessary when requests are streams.

Distributor (D) Useful in distributed systems to communicate data d to several receptors (i.e., broadcasting). Distributors can be used when an SoS must communicate reconfiguration requests or for coordination purposes among multiple entities. Distributor is depicted in Fig. 3c.

Specifications:

- Duties: To distribute d to interested entities.
- Behaviors: $distribute(d)$ to an entities' list $entityList$ interested in receiving the data.
- Assumptions: Type of distribution is broadcast or directed, and receptors are previously known by emissor.
- Guarantees: Delivery of d to interested entities, and d can be understood by receptors.

Router (R) Coordinates data flow when multiple entities are transmitting and receiving data in an SoS. As presented in Fig. 3d, R receives data $d_1, d_2, d_3, ..., d_n$ from multiple entities $(E_1, E_2, E_3, ..., E_n)$ (differently from a distributor, which has an unique input interface), and coordinately, sends data d_x to interested entities registered in an *entityList*.

Specifications:

- Duties: To route data d_j among entities.
- Behaviors: $route(d_j)$ to $entityList$.
- Assumptions: d_j from E_i can be understood by other entities.
- Guarantees: Transmission of data among multiple entities.

Conversion mediators

This type of mediators is concerned with interoperability among constituent and consumer systems, and security issues when data is transmitted. Four mediators, namely, filter, wrapper, adapter, and data fusion, showed in Fig. 4 a–d, were defined to support interoperable and secure communications and collaborations among entities, through operations on exchanged data or protocols. These mediators were abstracted from architectural styles/patterns trickle-up, SOA, shared-data, and broker, which have been applied for conversion purposes in SoS [32, 37, 57, 58].

Filter (F) Selects and removes parts of input data D simplifying its structure. Filters can select relevant information from outputs of constituent systems that contain more information than that required for SoS operations. Filters can be used to decrypt messages for security purposes, obtain data related with specific message headings for syntactic interoperability intentions, or to eliminate noise in data provided by constituent systems before their

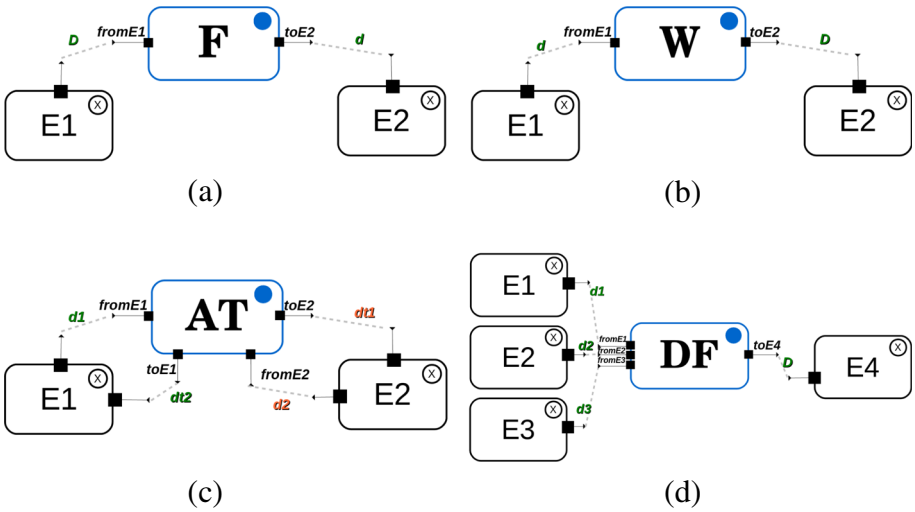

Fig. 4 Conversion mediators. Adapted from [23]. **a** Filter. **b** Wrapper. **c** Adapter. **d** Data fusion

use in SoS operations. This mediator is presented in Fig. 4a.

Specifications:

- Duties: To identify and filter d from input data D.
- Behaviors: $d = filter(D)$.
- Assumptions: D is understood by the filter, and d is contained in D.
- Guarantees: Selection and filtration of data d from D.

Wrapper (W) Enables technical and syntactic interoperability among constituent systems, consumer systems, and mediators that use different communication or transport protocols, or distinct data formats. Moreover, it adds encryption data or authentication information for security purposes. This mediator can add extra information w to data exchanged d to construct D, following a wrapping rule wr. Wrapper is depicted in Fig. 4 b.

Specifications:

- Duties: To create D adding w to d following the wrapping rule wr.
- Behaviors: $D = wrap(d, w, wr)$; $d = unWrap(D, w, wr)$.
- Assumptions: It exists a rule wr to wrap and unwrap data.
- Guarantees: D containing d plus additional information w. d can be recovered from D.

Adapter (AT) Known as *Translator*, it supports semantic interoperability of exchanged data d_i among constituent systems, consumer systems, and mediators. This type of interoperability is done through the translation or mapping between different data formats *Datatype*. Moreover, adapters can be used to match interaction protocols among constituent systems. Figure 4c shows this mediator.

Specifications:

- Duties: To receive d_1 from E_1; To adapt d_1 into dt_1, where $d_1.Datatype \neq dt_1.Datatype$, using the transformation rule tr; To communicate dt_1 to $E2$.
- Behaviors: $dt_1 = Adapt(d_1, tr)$.
- Assumptions: There exists a transformation rule tr to allow translation between exchanged data formats.
- Guarantees: Correct transformation of data transmitted between two entities. dt_1 can be successfully understood by E_2.

Data fusion (DF) Sometimes referred to as *Aggregator*, it collects and merges individual data d_1, d_2, d_n from different entities E_1, E_2, E_n and creates a single aggregated output D for further transmission, following a predefined aggregation rule agr. D contains (parts of) d_1, d_2, d_n and, if needed, some additional information w for SoS control purposes. Aggregated data is important to SoS to understand about its current status and decide future behaviors or reconfigurations in its structure. Data fusion mediator is presented in Fig. 4 d.

Specifications:

- Duties: To receive d_1, d_2, d_n from E_1, E_2, E_n. To merge d_1, d_2, d_n into D. To communicate D.
- Behaviors: $D = aggregate(d_1, d_2, ..., d_n, w, agr)$.
- Assumptions: d_1, d_2, d_n are semantically and syntactically interoperable. It exist a data aggregation rule agr to coordinate the creation of D.
- Guarantees: D as a fusion of $d_1, d_2, ..., d_n, w$.

Control layer

The upper layer of *MediArch* is focused on control issues, ensuring the identification and execution of emergent behaviors, dynamicity of the SoS, and missions accomplishment.

Mediators in this layer only have direct interaction with mediators located in the communication and conversion layer. Hence, a direct connection between control mediators and constituent or consumer systems is not possible.

Mediators in the control layer are concerned with behavioral and control issues of SoS software architectures. Four mediators, namely, monitor, analyzer, planner, and executer, were defined to deal with architectural reconfigurations for the execution of emergent behaviors, availability of constituent systems capabilities, or changes in SoS missions or its environment. Mediators for control purposes are presented in Fig. 5a–d. These mediators were abstracted from architectural styles/patterns Observer, Publish-subscribe, reflection, MAPE-K, SOA, centralized architecture, reconfiguration control architecture, contract monitor, pace layering, and evolution styles, which have been utilized in SoS for control, emergent behaviors identification, evolutionary development, and dynamic reconfiguration purposes [12, 13, 32, 50, 57, 58, 69], as described in Table 1.

Monitor (M) Collects[1] information d_i from groups of SoS entities E_j registered in an *entityList* and provides their status $s_1, s_2, ..., s_j$ (e.g., constituents behavior under execution, current environment status, constituent availability). Entities status s_j is aggregated to estimate entities situations *eSituation* during SoS operations. Figure 5a depicts the monitor mediator.

Specifications:

- Duties: To collect entities' information d_i;
  To define entities' status s_j;
  To infer and communicate entities' situation $eSituation$.
- Behaviors: $s_j = collect(E_j.d_i)$, where E_j is in $entityList$;
  $eSituation = aggregate(s_1, s_2, ..., s_j)$;
  $sendSituation(eSituation)$.
- Assumptions: Entities E_j to be monitored are registered in the $entityList$;
  Information d_i received from entities is semantically interoperable;
- Guarantees: Situation inference of entities participating of the SoS.

Analyzer (A) Establishes the situation of an SoS, *SoSituation*, based on historical knowledge on SoS (which is presented as model *SoSModel*) and its entities' current situation, *eSituation*. Examples of SoS situations are current emergent behavior under execution, SoS mission addressed, or if the SoS is presenting problems in its operation. This mediator is presented in Fig. 5 b.

[1] For this reason the monitor is also called collecter or observer.

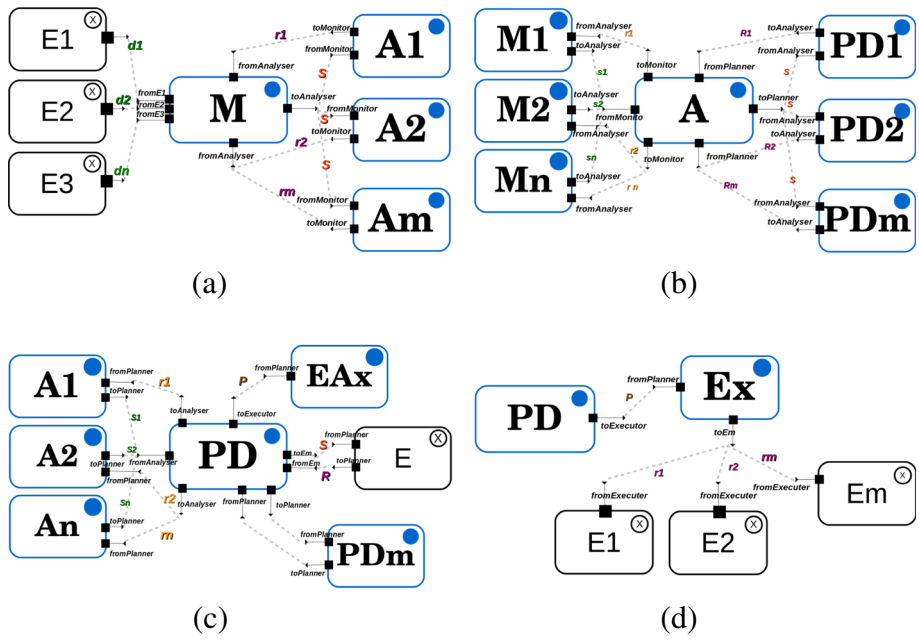

Fig. 5 Control mediators. Adapted from [23]. **a** Monitor. **b** Analyzer. **c** Planner. **d** Executer

Specifications:

- Duties: To infer *SoSituation* based on its model *SoSModel* and entities situation *eSituation*.
- Behaviors: $SoSituation = situation(SoSModel, eSituation)$.
- Assumptions: There is historical knowledge on the SoS previous status, prediction models, situation of entities, and the SoS models.
- Guarantees: Possible situations of the SoS.

Planner (PD) Also known as *Decider*, it selects a better configuration plan, *confPlan*, to be executed by an SoS (or by parts of it). Configuration plans can change an SoS architecture (e.g., adding or removing mediators, connecting or disconnecting constituents) and its behaviors, or request new functionalities to constituent systems. For this, the planner must request reconfiguration policies and to decide which configuration plan, *confPlan*, is the most adequate for the current SoS situation, *SoSituation*, and based on SoS historical behaviors, *SoSHystory*. The selected plan is communicated for further execution by responsible parts. Figure 5c illustrates this mediator.

Specifications:

- Duties: To establish and provide reconfiguration plans ,$confPlan$, based on current SoS situation, *SoSituation*, and history of SoS behaviors, *SoSHistory*.
- Behaviors: $policies = requestPlan()$;
  $SoSHystory = requestHistory()$;
  $confPlan = Planning(SoSituation, policies, SoSHystory)$;
  $providePlan(confPlan)$.
- Assumptions: A knowledge base must be defined containing the SoS current and historical status, its reconfiguration plans, and its policies.
- Guarantees: Reconfiguration plans.

Executer (Ex) Occasionally referred to as *Actuator*, it realizes the reconfiguration plan, *confPlan*, provided by planners, and distributes reconfiguration requests, *reconfRequest*, to specific SoS entities (i.e., entities under control of the SoS) registered in a *entityList*. The executer is showed in Fig. 5 d.

Specifications:

- Duties: To distribute reconfiguration requests, $reconfRequest$, to entities in $entityList$.
- Behaviors: $sendReconf(entityList, reconfRequest)$.
- Assumptions: Plans are semantically interoperable to be understood by entities.
- Guarantees: All reconfiguration requests in a reconfiguration plan are performed.

Evaluation of *MediArch*

To assess the applicability of *MediArch* , presented in the "Software mediators for SoS" section, the software architectures of two SoS, FMSoS and HSH-SoS, were structured considering mediators as first-class entities and are presented in the "Examples of SoS architectures application based on *MediArch*" section. The "Results of *MediArch* assessment" section provides answers to the five research questions introduced in the "Materials and methods" section.

Examples of SoS architectures application based on *MediArch*

To design software architectures of FMSoS and HSH-SoS, the guidelines proposed by Hofmeister et al. [29] were followed. These guidelines define that, in a general way, the architecting process comprises the stages of domain analysis, architectural analysis, architectural synthesis, and architectural evaluation. Specifically, in the

stage of synthesis, software architectures of both SoS were built following the structure proposed in *MediArch*, depicted in Fig. 2. In the remainder of this section, both architectures for FMSoS and HSH-SoS are presented in the "Flood monitoring systems-of-systems—FMSoS" and "Healthcare supportive home systems-of-systems —HSH-SoS" sections, respectively.

Flood monitoring systems-of-systems—FMSoS

FMSoS monitors river levels in a given region and uses the information provided by heterogeneous and independent water level sensors (or motes) distributed on the banks of the river [14]. Besides these sensors, other types of systems participate of the FMSoS, such as, pollution level sensors, crowd-sourcing systems, telecommunication gateways, unmanned aerial vehicles (UAVs), vehicular ad hoc networks (VANETs), meteorological centers, fire and rescue services, hospital centers, police departments, short message service centers, and social networks, as described in [26, 54].

In a general way, the FMSoS missions are detection of floods, prediction of evacuation situations, alert to respective authorities, and minimization of impacts caused by floods [27].

Two versions of architectures for FMSoS were preliminarily defined and assessed in [26, 27]. This section presents a new version of the FMSoS architecture based on mediators. Figure 6 a shows the layered view of FMSoS software architecture, as an instance *MediArch*.

- *Constituent and consumer systems layer*. In this version of FMSoS, constituent systems are sensors S_j and Gateways G_i. This layer also contains consumer systems, namely, sensor observation, web map, and emergency services. FMSoS does not control any systems in this layer, since their operation and management are independent of this SoS. Constituent systems are distributed through the river. Interfaces, transferred data, data formats, and technologies of consumer systems are heterogeneous, since they were developed following neither predefined communication protocols, nor semantic standards. Therefore, constituent systems do not address technical and semantic interoperability. As shown in Fig. 6 a, constituent and consumer systems do not establish direct communication; therefore , they only can interact through mediation layers of communication, conversion & coordination, and control layers.
- *Communication, conversion, & coordination layer*. Based on mediators proposed in the "Software mediators for SoS" section and on non-functional requirements of FMSoS presented in [26, 27], the following mediators were selected to grant communication, conversion, and coordination capacities: Collaborators (C), Distributer (D), Routers (R), and Adapters (AT). These mediators create the composed mediators C-AT and R-AT. C-AT is a mediator that aggregates the capabilities of mediators Collaborator (C) and Adapter (AT). In the FMSoS, mediator C-AT was allocated for each constituent system to communicate, in a bidirectional way, constituent systems data (e.g., river levels, river temperature, water velocity) with other mediators and systems and, at the same time, allows semantic interoperability among connected entities. R-AT was

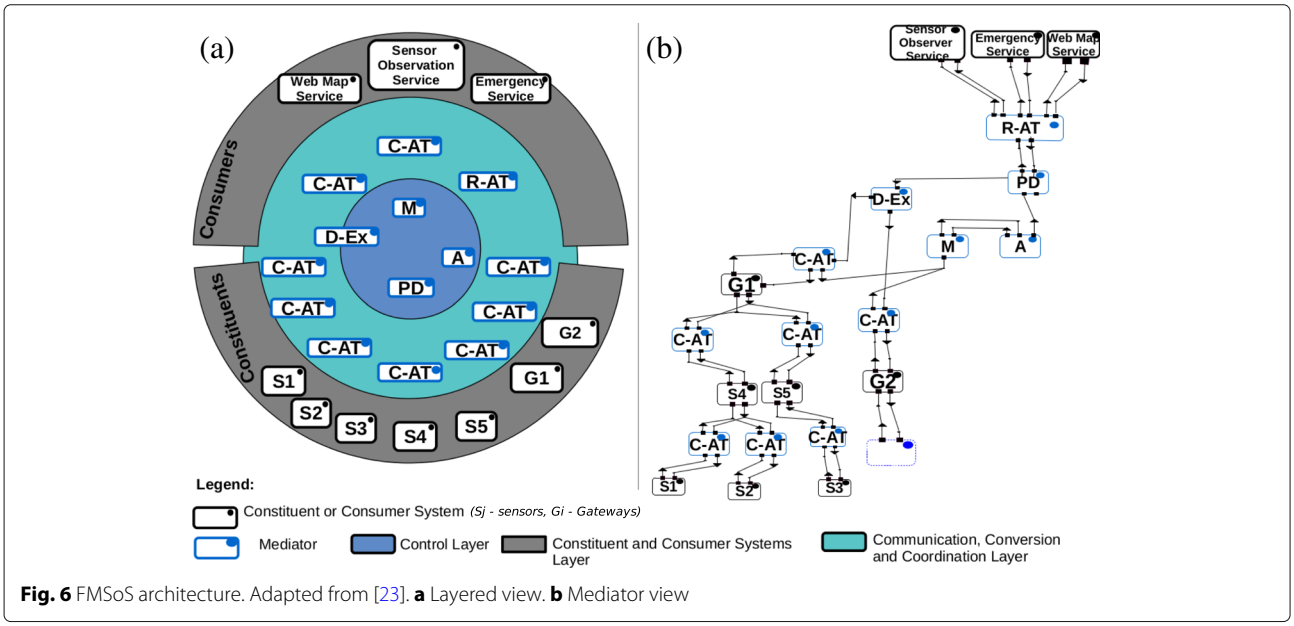

Fig. 6 FMSoS architecture. Adapted from [23]. **a** Layered view. **b** Mediator view

defined as a mediator that contains capabilities of routing (R) and adaptation (AT), and it is used to deliver data to consumer systems in a comprehensible way. In FMSoS, mediator R-AT communicates flood alerts, river situations, or SoS situations to sensor observation, web map, and emergency services. This communication is interoperable, since this mediator transforms heterogeneous data in standardized data formats and communication protocols, enhancing data comprehension by all consumer systems.

- *Control layer.* This layer is mainly formed by instances of mediators Monitor (M), Analyzer (A), Planner (PD), and Executor (Ex). Monitor collects data from sensors and gateways and establishes the current river and constituents status (e.g., river levels by region, river velocity, or contamination index). Such a status is directly communicated to the Analyzer that identifies possible situations of the FMSoS or its constituents (e.g., the incidence of floods in a region, unavailability of constituents or mediators, or normal operation). To predict FMSoS situations, the Analyzer uses a probabilistic situation model (*SoSModel*) stored in a repository. The Analyzer communicates the FMSoS situation to the Planner that establishes a reconfiguration plan for the SoS. These plans can define commands for deleting/instantiating new mediators, restarting of constituents, communicating emergency situations to different systems, or increasing sensing rates of whole sensors to improve river situation reliability. Reconfiguration plans are sent to the Executor that identifies software entities involved in a reconfiguration, organizes activities as workflows commands to be executed by each entity, and sends those commands to be delivered by the Distributor. Hence, the composed mediator D-Ex guarantees the correct distribution of reconfiguration commands in the FMSoS. Finally, mediators in the control layer do not interact directly with constituents and with consumer systems, therefore, requests are made by mediators in the middle layer.

Healthcare supportive home systems-of-systems—HSH-SoS

HSH-SoS has been designed to overcome the increasing demand for tele-homecare due to the growth of diseases and disabilities and avoid long-term hospitalization [20, 22]. HSH-SoS involves a variety of technologies and provides an autonomous life in residences to patients suffering from chronic diseases (e.g., diabetes, hypertension, Alzheimer's, Parkinson's), handicapped individuals, and dependent elderly people [19].

HSH systems are collaborative SoS, as each constituent system maintains its operational and managerial independence; however, constituents collaborate with the HSH system to achieve and evolve global missions. Examples of constituent systems are smart homes, domotic systems, electronic health records (EHR) systems, monitoring systems, smart devices, and rehabilitation systems [20]. In these SoS, collaborations among constituent systems grant responses to alarm situations and trigger preventive actions based on perceived trends and behavioral patterns. Such systems are based on intelligent situation analysis, help in the assessment of patient health conditions, and assist on fitted interventions. Moreover, HSH-SoS can consider local interventions (e.g., resorting to robotics, opening/closing doors or windows, emergency callings, activity monitoring, oxygen provision), tele-presence (in connection with carers, family, or community), or interventions provided by active supporting teams from the care network [21].

Figure 7 depicts the architecture of HSH-SoS, whose construction followed *MediArch*. The HSH-SoS has a mission to support patients diagnosed with diabetes mellitus during the self-management of their diseases at home. Requirements of this SoS can be consulted in [67], and a preliminary version of its software architecture is detailed in [24]. In a general way, the HSH-SoS architecture is specified as follows.

- *Constituents and consumers systems layer.* The bottom layer contains the following three types of services:
  - Services provided by constituent systems that are of interest of HSH systems to achieve their missions, but that are managerially and operationally independent of this SoS, e.g., services furnished by domotic system, sleep monitor, heart monitor, weight assessment system, body measurement system, blood pressure monitor, insulin, and blood glucose monitor. These systems are presented at the bottom-left of Fig. 7.
  - Services offered by constituent systems that are under the SoS control and depend on the information provided by external systems. These constituents, shown at the bottom-right of Fig. 7, are dietetic assistant systems, diabetes monitor, emergency manager, AoDL (activities of daily life) monitor, patient's physical signs and symptoms monitor, pharmacological treatment manager, disease intervention manager, and physical exercise monitor.
  - Consumer services that interact with the SoS, requesting and offering information from constituent systems and results of SoS operations. Examples include ambulance services, nutrition plan manager, patient

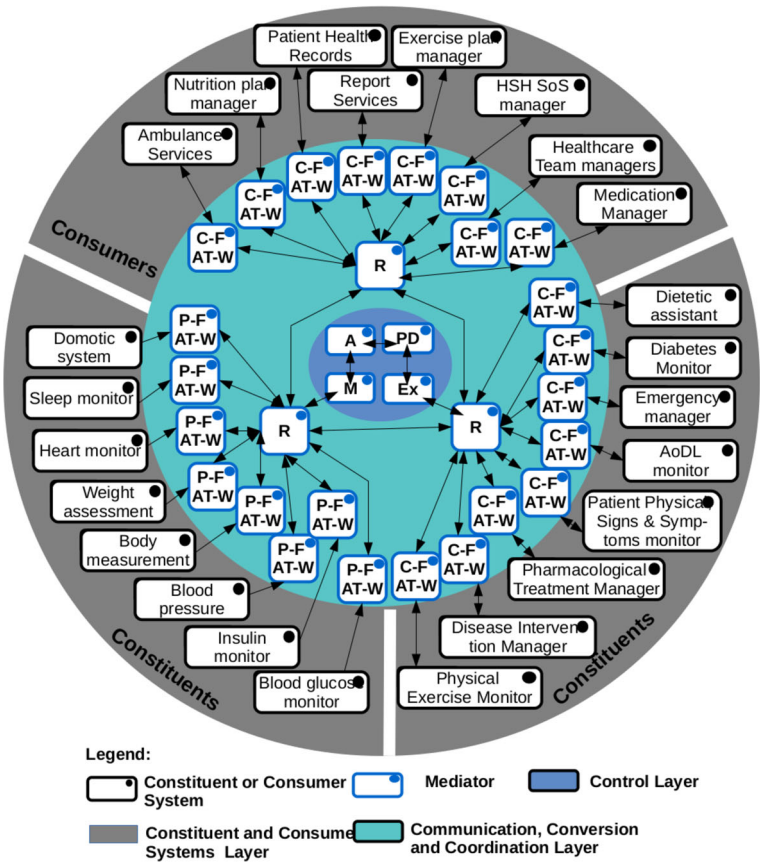

Fig. 7 Layered view of HSH-SoS

health records, report services, exercise plan manager, HSH SoS manager, health-care team managers, and medication managers, presented at the top of Fig. 7.

- *Communication, conversion, & coordination layer.* The middle layer is composed of mediators of type *Pipe (P)*, *Collaborator (C)*, and *Router (R)* for communication and coordination purposes, and *Filter (F)*, *Adapter (AT)*, and *Wrapper (W)* for conversion. Based on these mediators, two composed mediators were used in the HSH-SoS: The *P-F-AT-W*, which transfers data, unidirectionally, from constituent systems that are totally independent of the SoS to elements in upper layers. Constituent systems connected to the SoS through composed mediators *P-F-AT-W* are located at bottom-left in Fig. 7. Moreover, *P-F-AT-W* selects only important information for SoS operations from data obtained from constituents, as well as the translation and modification of such data to be communicated in an interoperable way to other entities participating in the SoS. In this SoS, a mediator *P-F-AT-W* was allocated for each constituent system. The another composed mediator is *C-F-AT-W*, which differs from the *P-F-AT-W* only in the sense that allows bidirectional communications among constituent and consumer systems and the SoS control. Then, *C-F-AT-W* also facilitates multiple transformations of data among constituents, consumers, and the SoS, while *Routers (R)* were used to communicate all transformed and standardized data to elements in the three layers, namely, constituent systems, consumers systems, and mediators.
- *Control layer.* The higher layer is responsible for coordinating HSH-SoS operations and executing their missions. This layer is composed of mediators of type *Monitor (M)*, *Analyzer (A)*, *Planner (P)*, and *Executer (Ex)*. As showed in Fig. 7, only mediators *Monitor* and *Executer* have bidirectional communication with *Router* (from the communication, conversion, & coordination layer). *Monitor* receives standardized data from constituent and consumers systems through *Router* and sends the status of both patients and the HSH-SoS to the *Analyzer*. The *Analyzer* infers possible situations

(e.g., systems faults, patient emergency situation) and communicates them to the *Planner*, that establishes different plans of action for avoiding or recovering from the predicted situations. Plans are sent to the *Executer* that defines the entities (i.e., mediators, constituent systems, consumer systems) involved in the execution of these plans. *Executer* communicates actions to be fulfilled by entities through the *Router*.

Results of *MediArch* assessment

Based on the experience of using software mediators defined in *MediArch* to construct FMSoS and HSH-SoS, we present a descriptive analysis about how *MediArch* can contribute to address important properties of SoS architectures, namely, integration of constituent systems, emergent behavior assurance, decision-making or control strategies, evolutionary development, and adaptability or resilience [32]. In the remainder of this section, we answer the five research questions (RQs) proposed for our study, which were introduced in the "Evaluation of *MediArch*" section.

RQ1—How do SoS architectures based on MediArch allow continuous accommodation or integration of constituent systems?

To permit continuous integration of constituent systems, a SoS architecture must offer entities to ensure *interoperability* at technical, semantic, and syntactic levels. Only when interoperability is granted, *cooperation* among constituents and among other SoS entities is possible. Moreover, the fact that constituents participate in an SoS must not affect their operational and managerial independence. Therefore, SoS architectures must allow *portability* and *flexibility* when constituents are plugged to or unplugged from the SoS without affecting their internal structure and configuration.

SoS architectures based on *MediArch* can address *interoperability* among constituent systems through the composition of communication, conversion, & coordination mediators. For instance, in FMSoS (see Fig. 6b) for each constituent system (e.g., sensors *Sj* and gateways *Gi*), a composed mediator C-AT (including capabilities of collaborator and adapter mediators) was allocated with the aim of modifying data formats, vocabularies, and protocols used by each system into a common standard. Similarly, a composed mediator R-AT (containing capabilities of router and adapter mediators) was allocated to establish communication and information adaptations to consumer systems, such as sensor observer, emergency, and web map services, among others. For the same purpose, in HSH-SoS (see Fig. 7) two composed mediators, namely P-F-AT-W and C-F-AT-W were defined. The former aggregates capabilities from pipes (P), to obtain data from constituents; filters (F), to select only relevant data for SoS operations; adapter (AT), to translate concepts contained in constituents data to an unified vocabulary managed by the SoS; and wrapper (W), to transform constituents data formats into standardized data format to be managed in the SoS. The mediator C-F-AT-W also promotes data filtering, concepts translation, and data format transformation; however, it permits bidirectional data exchange among SoS entities, differently from P-F-AT-W, whose interactions are unidirectional.

Mediators in the communication, conversion, & coordination layer allow interoperability and coordination of data exchanged among SoS entities. Specifically, mediators Collaborator (C), Distributer (D), and Router (R) are responsible for delivering data, messages, events from constituent systems, consumer systems, and mediators to the SoS controllers, and vice-versa, promoting *cooperation* of constituents with other SoS entities.

In FMSoS and HSH-SoS architectures, constituent systems were not directly connected to SoS controllers, but through mediators from the communication, conversion, & coordination layer, that were allocated to each constituent system. In *MediArch*, these mediators allow *low coupling* among constituent systems, consumer systems, and SoS controllers, since they can cooperate making use of mediation capabilities, and avoid adaptations of their interfaces for communication purposes. Moreover, *MediArch* promotes *separation-of-concerns*, since each type of mediator has unique responsibilities that can be reused at connecting constituent or consumer systems, saving these systems to include mediation capabilities in their operations. Moreover, once constituent or consumer systems are removed from the SoS, they can be fully operational and used in other contexts. In this scenario, *MediArch* enables *portability* and *flexibility* of constituent and consumer systems to be used by the same SoS in different scenarios, or by purposes outside of SoS scope.

Trade-offs Significant performance overhead can be imposed on SoS operations during data transformations executed by conversion mediators. For this, a work-force must be dedicated to structure interoperation standards governing the SoS. Finally, transformation rules among data structures must be created when a constituent system does not follow a specific interoperability standard, for instance, in the HSH-SoS transformation rules must be defined to map segments of the XML-based messages to segments of HL7-based messages. HL7 (Health Level 7) is a standard for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery, and evaluation of health services [28].

RQ2—How do SoS architectures based on MediArch assure emergent behaviors?

SoS must predict or evaluate possible emergent behaviors and estimate their impact over operations and missions achievement [32, 48, 52]. Therefore, SoS architectures must be constructed to facilitate the reasoning about emergent behaviors [32]. For this, SoS architectures must allow *organizational interoperability* allowing coordinated interactions among participants, and *awareness of SoS situations* to perceive possible unexpected behaviors.

In *MediArch*, mediators of type Collaborator (C), Distributor (D), and Router (R) are responsible to coordinate interactions among entities participating in a SoS. Hence, these mediators can be specialized for executing workflows required during SoS operations. These workflows are a combination of capabilities (functionalities, operations, processes) offered by constituent systems that are arranged in a systematic way to allow the execution of SoS missions, achieving *organizational interoperability*. It is only through the coordination of constituents capabilities that new SoS behaviors emerge [46, 48, 52]. In the FMSoS, the communication, conversion, & coordination layer contains all mediators needed to allow the cooperation of systems in a crisis situation. For instance, consumer systems (web map service, sensor observation service, and emergency service) can exchange information (in an interoperable way) making use of the mediator R-AT to coordinate efforts to assist people during a disaster. Moreover, consumer systems can obtain updated information from constituent systems to support their decisions.

Through mediators located in the control layer of *MediArch*, it is possible to obtain a partial situation of each participant in a SoS. These situations are useful to establish possible behaviors and their effects on SoS operations. This is allowed thanks to the interaction between mediators of type Monitor (M) and Analyzer (A). For instance, in the HSH-SoS, the Monitor, through the Router (R), can obtain different information from constituent systems (e.g., domotic system, sleep monitor, heart monitor, insulin monitor); define the status of each system (active, inactive), patient parameters (quality of sleep, heartbeats, blood sugar levels), house condition (empty, occupied); and communicate such information to the Analyzer (A). The Analyzer then defines possible situations for both patient's health and the SoS. In this context, the monitor-analyzer duo offers *situation-aware* capabilities required to measure risks in SoS operations or patient's health status.

Trade-offs Having continuous monitoring and analysis of constituent systems and SoS operations to establish situations and predict emergent behaviors might negatively impact time and resource consumption [32]. Models of possible SoS situations are required, demanding high efforts in SoS architectural analysis stage. Mostly, these models are modified at runtime, requiring the adoption of challenging strategies, as models@run.time [1].

RQ3—How can different strategies of control authorities be established in SoS architectures based on MediArch?

Depending of the SoS type, namely, directed, acknowledge, or collaborative, control authority varies, e.g., a directed SoS can require a *centralized* control, differently from acknowledge or collaborative SoS whose control authorities can be *decentralized* or *fully-decentralized* [32, 46, 52, 57]. Therefore, SoS architectures must be flexible regarding the composition of entities *accountable for decision-making*.

In *MediArch*, mediators in the control layer, namely, Monitor (M), Analyzer (A), Planner (PD), and Executer (Ex), are low-coupled making it possible to create different configurations of composed mediators. Hence, for directed SoS, the composed controller mediator M-A-PD-EX can be arranged to control all operations, configurations, and qualities in a *centralized* way. In this scenario, the mediator M-A-PD-Ex can be set up following the pattern MAPE-K, also known as autonomic manager [61].

Moreover, different control activities (e.g., monitoring operations, analyzing systems situation, planning of reconfigurations, or executing of control commands) can be performed individually by specialized mediators following workflows configured to pursuit SoS missions. In FMSoS and HSH-SoS, decisions were made by control mediators arranged in *a decentralized* way. For instance, in FMSoS (see Fig. 6b), control mediators were configured to enable (1) bidirectional interactions between the monitor and the analyzer, (2) unidirectional communication from the analyzer to the planner, and (3) unidirectional communication between the planner and the executer. Whereas, in the HSH-SoS (see Fig. 7), control mediators were set up to have bidirectional communication between monitor-analyzer, analyzer-planner, and planner-executer.

Additionally, control mediators can be organized in a *fully-decentralized* configuration enabling concurrent analysis of different situations. For instance, a composed control mediator M-A-PD-Ex can be created to manage each desired quality in an SoS, e.g., performance, reliability, and safety. Each composed mediator can identify the SoS level-of-agreement regarding each quality property and define different reconfiguration strategies to ensure or recover an expected quality level.

The assurance of an *accountable decision-making* requires the establishment of reconfiguration policies to be performed by the planner. These policies must specify possible trade-offs in SoS operations (e.g., decreasing performance level) and strategies to mitigate them.

Trade-offs The definition of reconfiguration policies is a time-demanding task, since a deep analysis and formal representation of architectural trade-offs related to reconfigurations are required. Moreover, over reconfigurations of the SoS, participant systems, and mediators can appear, as more composed controllers are defined. This can be mitigated by the reflection strategy [12], in which a higher layer (out of the SoS control) monitors the SoS as a whole, prioritizing types of reconfigurations and quality properties.

RQ4—How can SoS architectures based on MediArch evolve?

New missions, requirements, and policies can be required in an operating SoS. This can imply adding or removing constituent systems, requiring new capabilities from participants, or creating, removing, or modifying mediators configurations. Therefore, entities in SoS architectures must be *adaptable*, *scalable*, and *modifiable* for considering unforeseen changes without affecting the SoS quality.

The strategy adopted in *MediArch* for addressing *adaptability* and *modifiability* and facilitating *maintainability* of SoS architectures and its mediators were proposed considering separation-of-concerns and low-coupling, avoiding possible interdependencies among constituent systems, consumer systems, and SoS controllers. In this scenario, SoS entities can be added/removed and SoS operations can change, as required by the controller, without affecting participant systems configurations. These characteristics are required to allow SoS *evolution* and *scalability*. For instance, in the FMSoS, new constituent systems (e.g., sensors or gateways) can be located in a different region of a river to increase the area monitored by this SoS. For this, mediators C-AT can be created as new constituents are included in the SoS. Information from new constituents is aggregated to offer updated information to consumer systems, e.g., web map service or sensor observation service.

Trade-offs Reconfigurations of an SoS require to preserve the current execution state (e.g., data values, active operations, interdependencies) of all entities (e.g., consumers, constituents, mediators, controllers) involved in the reconfiguration plan. Strategies that ensure safety and non-disruptive reconfiguration approaches [66] must be defined and considered in *MediArch*.

RQ5—How do SoS architectures based on MediArch address resilience or adaptability?

Most operational and managerial independence and distribution of constituent systems imply inaccessibility of constituents capabilities (due to network problems or unavailability of systems) required in SoS operations. Therefore, SoS architectures must be *resilient* to support SoS operations in challenging environments. Resilience can be considered in software architectures for enabling the *self-organization* of their structures and *dynamic reconfiguration* of their operations. Resilience is an important requirement for SoS and assure *reliable* and *robust* operations [52, 64].

In *MediArch*, *self-organization* and *dynamic configuration* can be achieved by two strategies. The first one is to organize *MediArch* mediators following the *pace layering* pattern [57], also known as layers of changes. The second strategy was to allow the composition of mediators located in the control layer. These mediators can be aggregated to realize the autonomic manager pattern (or MAPE-K) [61], which is a strategy in self-adaptive systems to achieve architecture reconfigurations in run-time [2, 69]. Moreover, control mediators can identify faults in constituent systems and SoS operations and, through a predefined set of policies, propose changes in structure and setting of the SoS to prevent or recover from faults. This scenario brings a strategy to achieve *reliability* and *robustness* in SoS architectures [32, 52, 64].

Trade-offs SoS resilience and adaptability at runtime depend on the quality of reconfiguration policies specifications. As in *MediArch*, the policies logics is independent of SoS controllers capabilities, accessing and processing such policies could raise bottlenecks during reconfigurations execution, thus affecting the overall SoS performance.

Discussion

SoS impose challenges to software architects, mainly because their construction is based on collaboration among software systems geographically distributed and managed by different providers. Constituent systems are often engineered by distinct software organizations for specific purposes without considering their further integration in more complex systems. Therefore, constituents are operationally independent, and their data models are heterogeneous, which requires to refactor their code towards interoperability capabilities. To conform a SoS, it is required a successful collaboration among their internal entities, constituent systems, and consumer systems. Then, it is important to have well-defined interfaces, standardized data structures, a common vocabulary, clear workflows, communication protocols, and entities responsibilities. Moreover, collaborations among constituents must be flexible to permit adding or removing new participants, requirements, and activities during an SoS operation. The traditional integration of systems in which participants are highly connected is not favorable in SoS.

Related work

Different strategies have been proposed to support integration, interoperability, coordination, and collaboration of distributed software systems [10]. These solutions vary from specific architectural structures such as connectors and mediators, to more complete architectural solutions, e.g., middlewares, frameworks, and platforms.

Mehta et al. [47] proposed, by the early 2000s, the first initiative to consolidate a taxonomy of software connectors for component-based systems. This taxonomy comprises four categories of connectors, i.e., communication, coordination, conversion, and facilitation. Since then, connectors started to be considered as first-software entities for component-based architectures, separating communication logic (provided by connectors behaviors) from business logic (implemented in components). However, connectors are limited when dynamic alliances between heterogeneous components (developed in different programming languages or running in different platforms) are required. Despite connectors which enhance reuse in component-based systems, they are in some ways high-coupled since they are set up to components they connect, limiting their use in more dynamic settings, such as in SoS. This is evidenced in two works [17, 51] who focused in directed SoS in the military domain. Combs and Vaggle [51], describe a directed service-oriented SoS, that dynamically creates connectors to integrate services in the architecture. These connectors are only focused on communication protocol conversions. Similarly, Ehrmantraut [17] proposes to use software connectors for integrating constituent systems. For each system, a connector is set up for allowing the transformation of communication protocols. In both works, the connector logic of communication and conversion is implemented in a high-coupled way.

The increasing need for creating dynamic alliances of distributed and heterogeneous systems lead the creation of software mediators, initially referred to as mediation connectors. Mediators were initially defined in [70, 73] as an intermediate layer to promote communication, simplification, abstraction, reduction, merging, communication, and intelligence during interaction among highly distributed systems and heterogeneous data representation. Different types of mediators have been proposed in complex systems during the last fifteen years [8, 18, 36, 43, 62]. Ege et al. [18] define a three-layered framework composed of three types of mediators (i.e., presence, integration, and homogenization) for information systems. This framework is oriented to collect data from different databases and integrate such data to be presented in an aggregated way to final users. However, this framework does not consider interactions, collaboration, or coordination among data providers which is a common characteristic in SoS. Li et al. [43] define a mediator to allow cooperation among different web services, focusing on the conversion of individual services work-flows to collaborate in the execution of composed business processes. Spalazzese et al. [62] is one of the first works proposing the dynamic synthesis of mediators in networked systems. The creation of mediators at runtime is made by a central controller and supports interoperability at technical and organizational levels since communication and operational protocols of distributed systems are homogenized. Despite works of Li et al. [43] and Spalazzese et al. [62] enhancing organizational interoperability in service-oriented, networked, and component-based systems, solutions for important issues, such as technical, semantic, syntactic interoperability, and control of distributed, heterogeneous, and independent systems (as those found in SoS) are not described.

To solve mediation issues in SoS, during the last five years, middleware [8, 36, 45] and framework [63, 65, 68] solutions have been suggested. The middleware in Lopes et al. [45] was proposed for IoT-based SoS. This middleware considers a central service control for decision-making and a central bus and proxy for constituents coordination purposes. Despite to enhance constituent alliances in a SoS, strategies to integrate constituent systems are only oriented to directed SoS, where constituents interfaces must be modified and the SoS has control over their operations, limiting the participation of more operational and independent constituent systems. Tomson and Preden [63] designed MACE, a simulation framework based on multi-agents for modeling proactive middleware in directed SoS and assessing possible emergent behaviors. MACE internal structures are not clear, harnessing the understanding on which and how SoS architectures properties are met by this framework and the simulated middleware. The framework defined by Varga et al. [65] supports cloud-based and event-oriented SoS and defines a central orchestrator for decision-making in directed SoS. This framework supports interoperability at technical (following cloud-based architecture) and organizational (proposing services orchestration) levels. However, the composition of services is coupled, since they must be adapted to participate in the SoS, limiting the operational and managerial independence of constituent systems; therefore, a new SoS must be configured (through the configuration of services adapters) when new requirements/missions are required. Wanderley et al. [68] proposed the MBA (Model-Broker-Agent) framework based on multi-agent systems that are interconnected through composed connectors named facilitators-broker. This framework defines a central broker to coordinate constituent systems and agents to transform or facilitate communication protocol and data semantic conversions. The use of multi-agents systems promotes SoS properties of robustness, low-coupling, and technical and syntactic interoperability. However, the MBA framework lacks

strategies to support important properties of SoS architectures, such as dynamism, evolution, and organizational interoperability.

Finally, the emergent middleware, proposed by Issarny and Bennaceur [36] and Bennaceur and Issarny [8], is the most related work found in the literature. This middleware enables the on-the-fly creation, remotion of mediators allowing adaptability and dynamic configuration of SoS architectures. This middleware has learning enablers to continuously determine components behaviors, it achieves semantic interoperability through ontologies, and it was conceived to support alliances of components in directed, acknowledge, and collaborative SoS.

MediArch contributions

Last column of Table 2, summarizes *MediArch* contributions to support architectural properties (P), quality attributes, and SoS types. Properties and qualities of SoS architectures were introduced in the "Identification of mediation requirements in SoS" section and types of SoS were defined in the "Systems-of-systems" section. Table 2 compares contributions made in *MediArch* and related works.

MediArch advances the state-of-the art, since from our knowledge, is the second approach proposed to overcome problems in SoS architectures that considers mediators as first-class entities. The first work in this line is reported in [36]. *MediArch* also proposes mediators organizations that can be configured to support (at least partially) all SoS properties. Therefore, SoS architectures based on *MediArch* could be able to dynamically integrate constituent systems, identify possible emergent behaviors, set up different strategies for control and decision-making (i.e., centralized, decentralized, full-decentralized), evolve over time including or modifying new requirements, policies, or constituents, and decide and execute reconfigurations at runtime. This is possible, due to mediators were proposed in *MediArch* to address important qualities, such as interoperability (in all its levels), low-coupling, separation of concerns, scalability, and maintainability. Moreover, based on experiences at architecting FMSoS and HSH-SoS, we are confident that *MediArch* can be used as backbone to create software architectures for directed, acknowledge, and collaborative SoS.

Limitations and directions for future studies

The SoS construction requires investments of time, economical resources, human resources, and the establishment of partnerships and collaborations among owners and managers of the SoS and its constituent systems. Hence, architectures of both FMSoS and HSH-SoS were conceptually validated in this work. *MediArch* was assessed regarding organizations of SoS entities (constituent systems, consumer systems, and mediators) for achieving requirements of interoperability, integration, modifiability, low-coupling, and flexibility. Low-coupling capabilities brought by mediators allow us to visualize how SoS architectures could support dynamic reconfigurations; however, more investigations about mediators synthesis at runtime must be conducted to verify this hypothesis. Moreover, additional studies about how *MediArch* could overcome issues related to performance, non-disruptive configurations, safety, and security must be done. These issues were identified as trade-offs during *MediArch* assessment.

Complementary decisions to define internal structures and behaviors of each mediator must be made. For instance, for communication mediators, *Collaborator (C)* can implement the *RPC*[2] pattern, *Router (R)* can be structured following *publish-subscribe*, and *Distributor (D)* can be assembled as a *service bus*. For control mediators, *Monitor (M)* can be instantiated either as an *Observer* when no clear contract exists between SoS and monitored constituent systems, or as part of a *publish-subscribe* strategy when *Monitor (M)* subscribes to messages sent by constituent systems. For *Analyzer (A)*, additional decisions are related to approaches for predicting situations (or emergent behaviors) based on historic SoS status. Situations can be predicted using machine learning algorithms, such as those proposed in [30], namely, dimension reduction, clustering, classification, or regression. For *Planner (P)* a standardized template to represent reconfiguration plans must be established, since reconfiguration strategies could be modified at runtime by the SoS manager. Both reconfiguration plans (used by *Planner (P)*) as situations records (used by *Analyzer (A)*) can be modeled through ontologies, XML schemes, graphs, entity-relationship, key-value, or NoSQL schemes, and stored in dedicated repositories. Decisions to implement mediators will depend on (1) the nature of each SoS, i.e., directed, acknowledged, collaborative, and virtual [46, 52]; (2) categories of emergent behaviors that the SoS could present, i.e., simple, weak, strong, or spooky [48]; and (3) SoS requirements related to quality attributes, e.g., performance, reliability, or security.

Besides using *MediArch*, SoS architects will need to define strategies for mediators synthesis at runtime, to achieve requirements of reliability, dynamic evolution, and adaptivity. Automatic synthesis is required, as mediators must be created or removed during SoS operations. Some interesting approaches architects could follow are presented in [27, 33–35].

Initial research on requirements of quality attributes and architectural styles/patterns for SoS and self-adaptive, self-management, self-organizing (self-*) systems served

[2] Remote procedure call

Table 2 Comparison between *MediArch* and related work

Comparison criteria	Related work reference											
	[51]	[17]	[18]	[43]	[62]	[36]	[63]	[8]	[45]	[65]	[68]	*MediArch*
Oriented to SoS	X	X				X	X		X	X	X	X
Oriented to mediators			X	X	X	X		X				X
Property												
P1		X							X	X	X	X
P2						X						X
P3	X		X		X				X	X		X
P4					X						X	X
P5												X
Quality attributes												
Cooperation				X								X
Integration	X	X	X					X				X
Technical interoperability	X	X	X	X	X			X	X		X	X
Syntactic interoperability			X								X	X
Semantic interoperability						X		X				X
Organizational interop.				X	X	X			X			X
Portability												X
Flexibility	X											X
Low-coupling				X							X	X
Situation-aware												X
Centralized control	X		X		X					X	X	X
Decentralized control												X
Full-decentralized control												X
Accountability												X
Maintainability				X								X
Modifiability				X								X
Separation of concerns				X								X
Adaptability				X		X		X				X
Scalability			X	X								X
Fault-tolerance				X								X
Self-organization												X
Robustness											X	X
Dynamic configuration	X				X	X	X					X
Performance**												
Non-disruptive**												

Table 2 Comparison between *MediArch* and related work (*Continued*)

Comparison criteria	Related work reference										
	[51]	[17]	[18]	[43]	[62]	[36]	[63]	[8]	[45]	[65]	[68]	*MediArch*
Safety**												
Security**												
SoS												
Directed		X		X		X	X		X	X		X
Acknowledge				X		X						X
Collaborative						X						X
Virtual**												

Legend: SoS Properties: *P1* continuous integration of constituent systems, *P2* emergent behavior assurance, *P3* the need for a decision-making or control authority, *P4* evolutionary development, *P5* resilience or adaptability. **Limitations of *MediArch*

as a basis to identify and specify mediators contained in *MediArch*. For instance, we considered the following styles/patterns: pace layering, SOA, autonomic manager, central architecture, broker, and publish-subscribe, as the most important ones. Mapping among requirements, styles/patterns, SoS characteristics, and mediators types was presented in Table 1. We observed that existing patterns, proposed more than 15 years ago [4], are as relevant now to architectures of contemporary systems (e.g., SoS and self-* systems), as they are for other types of systems. However, it is important to investigate about pattern languages, architectural frameworks, and reference architectures, to establish guidelines for solving recurrent problems in SoS architectures. *MediArch* could be used as initial step to standardize architectural solutions for SoS, and its application in FMSoS and HSH-SoS is an important start point for its consolidation.

Threats to validity

Three aspects of validity of our work were considered, namely construct, internal, and external. These aspects ensure the trustworthiness of results obtained at applying *MediArch* for constructing the architectures of FMSoS and HSH-SoS and analyzing the evidence to answer our RQs.

Construct validity

This validity reflects in which measure the use of *MediArch* supported the construction of SoS architectures that achieve continuous integration of constituent systems, assurance of emergence behaviors, decision-making authorities, evolution, and dynamic configurations.

To avoid threats to construction validity, *MediArch* and its mediators were defined based on deep studies on software connectors and mediators in large-scale and complex systems. Moreover, SoS architectural patterns and styles were investigated to understand their mediation approaches and their support to quality attributes of integration, interoperability (in technical, syntactic, semantic, and organizational levels) situation-aware, accountability of decisions, maintainability, modifiability, separation of concerns, portability, low-coupling, flexibility, scalability, adaptability, fault-tolerance, robustness, and dynamic configurations (see Table 1). Additionally, software architectures of FMSoS and HSH-SoS were constructed as instances of *MediArch*, following the guidelines proposed in [29]. More information about architectures of FMSoS and HSH-SoS can be consulted in [21–23, 26, 67].

Internal validity

This validity considers whether all causal relations that affect the application of *MediArch* were defined and handled. The following factors that could prejudice the construction of SoS architectures based on *MediArch* were identified: (1) the understanding of FMSoS and HSH-SoS scope and their requirements, which had no impact on results, since authors had previous experience on working with both SoS; (2) the difficulty to understand mediators and their application in SoS architectures, that was avoided, since mediators and the layered architecture were proposed based on well-known architectural patterns and styles; and (3) problems to instantiate *MediArch*, that were mitigated by using well-understood guidelines for software architecting [29].

External validity

This validity is concerned with the generalization of results obtained at applying *MediArch*, i.e., it is possible to obtain similar results in other SoS using this architecture and mediators proposed in this work. To mitigate external validity, we presented two cases in different application domains, demonstrating how SoS architectures could be hierarchically organized, having mediators as first-class entities.

Conclusions and future works

In our preliminary work [23], we established a taxonomy to support the selection of software mediators during architectural design of SoS. In this work, we extend such taxonomy as basis to construct *MediArch*, a high-level architecture composed of three layers: the constituent and consumer systems layer; the communication, conversion, and coordination layer; and the control layer. The last two layers describe hierarchical organizations of three software mediators, namely, communication & coordination, conversion, and control.

MediArch considers mediators as first-class entities in SoS architectures and was used as a backbone during the architectural synthesis of two systems, FMSoS, in the domain of crisis and emergency management, and HSH-SoS, in the domain of smart homes for health-care. Software architectures for both SoS have already been studied by authors in [21, 22, 26, 27]. In particular, in this work, alternative architectures for these SoS were presented using software mediators as core entities to enhance continuous integration of constituent systems, assurance of emergence behaviors, decision-making authorities, evolution, and dynamic configurations.

As future work, validation and verification of SoS software architectures based on *MediArch* can be done using simulation approaches such as those presented in [26]. Mediators were defined in an abstract way to support low-coupled architectures; hence, it is expected mediators could be dynamically synthesized and deployed to conform to different architectural configurations of an SoS. Moreover, mediators described herein are fine grained, with clear objectives and specifications; hence, they can be also implemented following different architectural styles, for instance, SOA or microservices to construct service-oriented SoS. Additionally, *MediArch* can be used to design more complex SoS architectures and reference architectures in different domains, as presented in [21, 22]. Finally, we intend to develop and publish a mediators-based platform that implements *MediArch*. This platform will facilitate SoS mediation improving the development of those systems, reducing costs and time. Besides that, it is our goal to continue investigating on mechanisms to support interoperation and dynamic organization of SoS in different domains, for instance, education, defense [6], and big data [60].

Abbreviations

ADL: Architectural Description Language; AoDL: Activities of daily life; FMSoS: Flood monitoring system-of-system; HSH: Healthcare supportive home system; IRISA: Institut de Recherche en Informatique et Systèmes Aléatoires; NoSQL: Class of database management systems (DBMS) that do NOT follow all of the rules of a relational DBMS and cannot use traditional SQL to query data; Self-*: Self-adaptive, self-management, self-organizing; SOA: Service-oriented architecture; SoS: System-of-system; SosADL: Systems-of-systems Architectural Description Llanguage; XML: Extensible Markup Language

Acknowledgements

The Authors thank the Program Committee of SBCARS'18 (XII Brazilian Symposium on Software Components, Architectures, and Reuse) for the opportunity to extend their previous work and submit it to this journal.

Authors' contributions

The first author wrote this article contents. The second and third authors oriented sections organization and reviewed this work. All authors read and approved the final manuscript.

Authors' information

Lina Garcés is a postdoctoral fellow at the Institute of Computer and Mathematic Sciences (ICMC) - University of São Paulo (USP), Brazil. She received her BEng (2009) and MSc (2012) from the Industrial University of Santander (UIS), Colombia, and her PhD (2018) from University of São Paulo (USP), Brazil, and University of Southern Brittany (UBS), France. Her main research interests are reference architectures, software architectures, dynamic architectures, interoperability , systems-of-systems, e-Health, ambient assisted living, and health ecosystems. She is a member of the IEEE, SBC (Brazilian Computer Society), and SBIS (Brazilian Society of Health Informatics). Contact e-mail: linamgr@icmc.usp.br.

Flavio Oquendo is a full professor of Computer Science (holding a Research Excellence Award from the Ministry of Higher Education and Research of France) serving as Research Director at the UMR CNRS IRISA, in Brittany, France. He received his BEng from ITA, Sao José dos Campos, SP, Brazil, and his MSc, PhD, and HDR from the University of Grenoble, France. He has published over 200 refereed journal and conference papers and has been editor of over 15 journal special issues and research books. He has served on program committees of over 100 international conferences, e.g., ICSE, ESEC/FSE, has chaired more than 10 of them, of which are the French, European, and IEEE/IFIP International Conferences on Software Architecture (CAL, ECSA, ICSA). His research interests are centered on formal languages, processes, and tools to support the efficient architecture of complex software-intensive systems and systems-of-systems. Contact e-mail: flavio.oquendo@irisa.fr.

Elisa Yumi Nakagawa is a MS (1998) and PhD (2006) in Computer Science from the University of São Paulo (USP), Brazil. She conducted her Post-Doctoral in 2011–2012 in Fraunhofer IESE, Germany, and in 2014–2015 at the University of South Brittany, France. She is associate professor in the Department of Computer Systems at the University of São Paulo, Brazil. Her main research interests are software architecture, reference architectures, systems-of-systems, software testing, and evidence-based software engineering. She is a member of the IEEE and SBC (Brazilian Computer Society). Contact e-mail: elisa@icmc.usp.br.

Funding

The São Paulo Research Foundation, FAPESP Grants No. 2018/07437-9, 2017/06195-9, and 2013/20317-9.

Availability of data and materials

Nothing to be included.

Competing interests

The authors declare that they have no competing interests.

Received: 7 February 2019 Accepted: 24 July 2019
Published online: 20 August 2019

References

1. Abmann U, Gotz S, Jézéquel JM, Morin B, Trapp M (2014) A reference architecture and roadmap for Models@run.time systems. In: Bencomo N, France R, Cheng BHC, Abmann U (eds). Models@run.time. Lecture Notes in Computer Science, vol. 8378. Springer, Cham. pp 1–18
2. Affonso FJ, Nakagawa EY (2013) A Reference Architecture Based on Reflection for Self-Adaptive Software. In: VII Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS'2013), Brasilia. pp 129–138
3. Amirat A, Oussalah M (2009) First-class connectors to support systematic construction of hierarchical software architecture. J Object Technol 8(7):107–130
4. Avgeriou P, Zdun U (2005) Architectural Patterns Revisited - A Pattern Language. In: the European Pattern Languages of Programming (EuroPLOP), Irsee. pp 1–39

5. Barnes JM, Garlan D, Schmerl B (2014) Evolution styles: foundations and models for software architecture evolution. Softw Syst Model 13(2):649–678
6. de Barros Paes CE, Neto VVG, Moreira T, Nakagawa EY (2019) Conceptualization of a System-of-Systems in the Defense Domain: An Experience Report in the Brazilian Scenario. In: IEEE Systems Journal (Early Access). pp 1–10. https://doi.org/10.1109/JSYST.2018.2876836
7. Beale D, Bonometti J (2012) Chapter 2: systems engineering (SE) - the systems design process. In: Fundamentals of lunar and systems engineering for senior project teams, with application to a lunar excavator. Auburn University, ESMD Course Material
8. Bennaceur A, Issarny V (2015) Automated synthesis of mediators to support component interoperability. IEEE Trans Softw Eng 41(3):221–240. March 2015
9. Benson T, Grieve G (2016) Principles of health interoperability SNOMED CT, HL7 and FHIR. Third edition. Springer-Verlag London, London. (Health Information Technology Standards)
10. Blair GS, Paolucci M, Grace P, Georgantas N (2011) Interoperability in Complex Distributed Systems. In: Bernardo M, Issarny V (eds). Formal Methods for Eternal Networked Software Systems. SFM 2011. Lecture Notes in Computer Science, vol 6659. Springer, Berlin, Heidelberg
11. Chopra AK (2008) Business process interoperability: extended abstract. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: doctoral mentoring program (AAMAS '08), Estoril. pp 1730–1731
12. Cuesta C, Romay MP (2010) Elements of self-adaptive systems – a decentralized architectural perspective. In: First International Workshop on Self-Organizing Architectures, SOAR, 2009, Revised Selected and Invited Papers. Springer Berlin Heidelberg 2010, Berlin. pp 1–20
13. Cuesta CE, Navarro E, Dewayne E, Roda C (2013) Evolution styles: using architectural knowledge as an evolution driver. J Softw Evol: Evol Process 25:957–980
14. Degrossi L, Do Amaral G, De Albuquerque J, Ueyama J (2013) Using Wireless Sensor Networks in the Sensor Web for Flood Monitoring in Brazil. In: Comes T, Fiedrich F, Fortier S, Geldermann J, Muller T (eds). Proceedings of the 10th International Conference on Information Systems for Crisis Response and Management (ISCRAM), Baden-Baden. pp 458–462
15. DoD (2008) Systems Engineering Guide for Systems of Systems, version 1.0. Washington, DC, USA: US Department of Defense (DoD). Available: http://www.acq.osd.mil/se/docs/SE-Guide-for-SoS.pdf. Accessed 20 June 2019
16. Dove R, Jennifer B (2011) Systems of systems and self-organizing security. INSIGHT 14(2):7–10
17. Ehrmanntraut R (2003) System-of-systems integration of air-ground telecommunications with the software connector. In: The 22nd Digital Avionics Systems Conference (DASC '03), vol. 2, Indianapolis. pp 6.A.3-61-12
18. Ege RK, Yang L, Kharma Q, Ni X (2004) Three-layered mediator architecture based on DHT. In: 7th International Symposium on Parallel Architectures, Algorithms and Networks, (I-SPAN 2004), Hong Kong. pp 313–318
19. Garcés L, Ampatzoglou A, Avgeriou P, Nakagawa EY (2015) A Reference Architecture for Healthcare Supportive Home Systems. In: IEEE 28th International Symposium on Computer-Based Medical Systems (CBMS'15), Sao Carlos. pp 358–359
20. Garcés L, Nakagawa EY (2017) A process to establish, model and validate missions of systems-of-systems in reference architectures. In: Proceedings of the Symposium on Applied Computing (SAC '17). ACM, New York. pp 1765–1772
21. Garcés L (2018) A reference architecture for healthcare supportive home systems from a systems-of-systems perspective. Thesis. University of Sao Paulo and University of Southern-Brittany
22. Garcés L, Oquendo F, Nakagawa EY (2019) Uma arquitetura de referência para sistemas de casas inteligentes de apoio ao cuidado à saúde da perspectiva de sistemas-de-sistemas. In: Proceedings of the Simpósio Brasileiro de Computação Aplicada à Saúde (CTD-SBCAS'19), Niterói. pp 73–78. https://doi.org/10.5753/sbcas.2019.6287
23. Garcés L, Oquendo F, Nakagawa EY (2018) Towards a Taxonomy of Software Mediators for Systems-of-Systems. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '18). ACM, Sao Carlos. pp 53–62
24. Garcés L, Zanin Vicente I, Nakagawa EY (2019) Software Architecture for Health Care Supportive Home Systems to Assist Patients with Diabetes Mellitus. In: IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS'19), Cordoba. pp 249–252
25. Garlan D (1998) Higher-Order Connectors. In: Workshop on Compositional Software Architectures, Monterey. pp 1–4
26. Graciano Neto V, Barros Paes C, Garcés L, Guessi M, Manzano W, Oquendo F, Nakagawa EY (2017) Stimuli-SoS: a model-based approach to derive stimuli generators for simulations of systems-of-systems software architectures. J Braz Comput Soc 23(13):1–22
27. Guessi M (2017) Synthesis of software architectures for systems-of-systems: an automated method by constraint solving. Thesis. University of Sao Paulo
28. HL7 International. Health Level Seven International (HL7). ANSI. Online: https://www.hl7.org/. Last Access: 27 June 2019
29. Hofmeister C, Kruchten P, Nord RL, Obbik H, Ran A, America P (2005) Generalizing a model of software architecture design from five industrial approach. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, Pittsburgh. p 2005
30. Hui L Which machine learning algorithm should I use? The SAS Data Science Blog, April 12, 2017. Available: https://blogs.sas.com. Last Access: 21 Jan 2019
31. Ingram C, Payne R, Perry S, Holt J, Hansen FO, Couto LD (2014) Modelling patterns for systems of systems architectures. In: 2014 IEEE International Systems Conference Proceedings, Ottawa. pp 146–153
32. Ingram C, Payne R, Fitzgerald J (2015) Architectural modelling patterns for systems of systems. INCOSE 25(1):1177–1192
33. Inverardi P, Issarny V, Spalazzese R (2010) A theory of mediators for eternal connectors. In: Margaria T, Steffen B (eds). Leveraging applications of formal methods, verification, and validation. ISoLA 2010. Lecture Notes in Computer Science, vol. 6416. Springer, Berlin
34. Issarny V, et al (2009) CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In: 14th IEEE International Conference on Engineering of Complex Computer Systems, Potsdam. pp 154–161
35. Issarny V, Bennaceur A, Bromberg YD (2011) Middleware-layer connector synthesis: beyond state of the art in middleware interoperability. In: Bernardo M, Issarny V (eds). Formal methods for eternal networked software systems. SFM 2011. Lecture Notes in Computer Science, vol. 6659. Springer, Berlin
36. Issarny V, Bennaceur A (2013) Composing Distributed Systems: Overcoming the Interoperability Challenge. In: Giachino E, Hähnle R, de Boer FS, Bonsangue MM (eds). Formal Methods for Components and Objects. FMCO 2012. Lecture Notes in Computer Science, vol 7866. Springer, Berlin, Heidelberg
37. Josuttis N (2007) SOA in practice: the art of distributed system design. O'Reilly Media, Inc, Beijing
38. Judith D, et al. (2008) Systems Engineering for Capabilities. CrossTalk. J Def Softw Eng 21(11):4–9
39. Kazman R, Nielsen C, Schmid K (2013) Understanding Patterns for System-of-Systems Integration. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2013-TR-017. Available in: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=75750. Accessed 15 June 2019
40. Kiwelekar AW, Joshi RK (2010) Identifying Architectural Connectors through Formal Concept Analysis of Communication Primitives. In: Babar MA, Gorton I (eds). Software Architecture. ECSA 2010. Lecture Notes in Computer Science, vol 6285. Springer, Berlin, Heidelberg. pp 515–518
41. Kubicek H, Cimander R, Scholl HJ (2011) Layers of interoperability. Organizational interoperability in E-government: lessons from 77 European good-practice cases. Springer Berlin Heidelberg, 2011. chap. 7, Berlin
42. Lau KK, Velasco Elizondo P, Wang Z (2005) Exogenous Connectors for Software Components. In: Heineman GT, Crnkovic I, Schmidt HW, Stafford JA, Szyperski C, Wallnau K (eds). Component-Based Software Engineering. CBSE 2005. Lecture Notes in Computer Science, vol. 3489. Springer, Berlin, Heidelberg
43. Li X, Fan Y, Wang J, Wang L, Jiang F (2008) A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation. In: Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Vancouver. pp 137–146
44. Lopes A, Wermelinger M, Fiadeiro JL (2003) Higher-order architectural connectors. ACM Trans Softw Eng Methodol 12(1):64–104

45. Lopes F, Loss S, Batista T, Lea R (2016) SoS-centric Middleware Services for Interoperability in Smart Cities Systems. In: Proceedings of the 2nd International Workshop on Smart (SmartCities '16). ACM, New York. pp 1–6. Article 4
46. Maier MW (1999) Architecting principles for system-of-systems. Syst Eng 1:267–284
47. Mehta NR, Medvidovic N, Phadke S (2000) Towards a taxonomy of software connectors. In: Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium. IEEE, Limerick. pp 178–187
48. Mittal S, Rainey L (2015) Harnessing emergence: The control and design of emergent behavior in system of systems engineering. Society for Computer Simulation International, Chicago. 2015
49. Morris E, Levine L, Meyers C, Place P, Plakosh D (2004) Systems of Systems Interoperability (SOSI): Final report. CMU/SEI-2004-TR-004 ESC-TR-2004-004. April 2004. Carnegie Mellon University:67. Available via: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2004_005_001_14375.pdf. Accessed 15 May 2019
50. Muller HA, Kienle HM, Stege U (2009) Autonomic computing now you see it, now you don't. In: Lucia A, Ferruci F (eds). Software engineering. Springer Berlin Heidelberg, Berlin. pp 32–54. Lecture Notes in Computer Sciences v. 5413
51. Nathan C, Jeff V (2002) Adaptive mirroring of system of systems architectures. In: Garlan D, Kramer J, Wolf A (eds). Proceedings of the first workshop on Self-healing systems (WOSS '02). ACM, New York. pp 96–98
52. Nielsen C, Larsen PG, Fitzgerald J, Woodcock J, Peleska J (2015) System of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput Surv 48(2):1–41
53. Nichols C, Dove R (2011) 7.1.3 Architectural patterns for self-organizing systems-of-systems. INCOSE Int Symp 21(1):856–867
54. Oquendo F (2016) Formally describing the software architecture of Systems-of-Systems with SosADL. In: 11th System of Systems Engineering Conference (SoSE). IEEE, Kongsberg. pp 1–6
55. Oquendo F (2016) Formally Describing the Architectural Behavior of Software-Intensive Systems-of-Systems with SosADL. In: 21st International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, Dubai. pp 13–22
56. Oquendo F (2017) Software architecture of self-organizing systems-of-systems for the Internet-of-Things with SosADL. In: 12th System of Systems Engineering Conference (SoSE). IEEE, Waikoloa. pp 1–6
57. Romay MP, Cuesta CE, Fernández-Sanz L (2013) On self-adaptation in systems-of-systems. In: Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems. IEEE, Montpellier. pp 29–34
58. Rothenhaus KJ, Michael JB, SM T (2009) Architectural patterns and auto-fusion process for automated multisensor fusion in soa system-of-systems. IEEE Syst J 3(3):304–316
59. Santos D, Oliveira B, Duran A, Nakagawa EY (2015) Reporting an Experience on the Establishment of a Quality Model for Systems-of-Systems. In: The 27th International Conference on Software Engineering and Knowledge Engineering (SEKE'2015), Pittsburgh. pp 304–309
60. Sena B, Garcés L, Allian A, Nakagawa E (2018) Investigating the Applicability of Architectural Patterns in Big Data Systems. In: 25th Conference on Pattern Languages of Programs (PLoP 2018), Portland. pp 1–10. HILLSIDE 978-1-941652-03-9
61. Sinreich D (2006) An architectural blueprint for autonomic computing. Fourth Edition. IBM White Paper, IBM
62. Spalazzese R, Inverardi P, Issarny V (2019) Towards a formalization of mediating connectors for on the fly interoperability. In: Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA). IEEE, Cambridge. pp 345–348
63. Tomson T, Preden J (2013) Simulating System of Systems Using MACE. In: 15th International Conference on Computer Modelling and Simulation (UKSim), Cambridge. pp 155–160
64. Valerdi R, Axelband E, Baehren T, Boehm B, Dorenbos D, Jackson S, Settles S (2008) A research agenda for systems of systems architecting. Int J Syst Syst Eng 1(1/2):171
65. Varga P, Blomstedt F, Ferreira LL, Eliasson J, Johansson M, Delsing J, Martínez de Soria I (2017) Making system of systems interoperable – the core components of the arrowhead framework. J Netw Comput Appl 81:85–95
66. Vasconcelos RO, Vasconcelos I, Endler M (2016) Dynamic and coordinated software reconfiguration in distributed data stream systems. J Internet Serv Appl 7(1):8
67. Vicente I, Garcés L, Nakagawa EY (2017) Establishment of a Software Architecture for Health Care Supportive Home System to Assist Patients with Diabetes Mellitus: Functional and Non-Functional Requirement. Technical Report N. 421. University of Sao Paulo, São Carlos. Available via http://repositorio.icmc.usp.br//handle/RIICMC/6651. Accessed 10 May 2019
68. Wanderley GMP, Abel M, Paraiso EC, Barthes JA (2018) MBA: A Framework for Building Systems of Systems. In: 13th Annual Conference on System of Systems Engineering (SoSE), Paris. pp 358–364
69. Weyns D, Ahmad T (2013) Claims evidence for architecture-based self-adaptation: a systematic literature review. In: 7th European conference on Software Architecture. Springer Berlin Heidelberg, Berlin Heidelberg. pp 249–265
70. Wiederhold G (1992) Mediators in the architecture of future information systems. IEEE Comput 25:38–49
71. Wiederhold G (1988) Interoperation, mediation, and ontologies. In: Institute for New Generation Computer Technology (ed). Proceedings of the International Conference on Fifth Generation Computer Systems, Tokyio. pp 33–48
72. Wiederhold G (1995) Mediation in information systems. ACM Comput Surv 27(2):265–267
73. Wiederhold G, Genesereth M (1997) The conceptual basis for mediation services. IEEE Expert 12:5:38–47

Publisher's Note