Architectural Solutions for Self-Adaptive Systems

Lina Garcés*, Silverio Martínez-Fernández†, Valdemar Vicente Graciano Neto‡, and Elisa Yumi Nakagawa*
*Department of Computer Systems, University of São Paulo - USP, São Carlos, Brazil.
†Universitat Politècnica de Catalunya (UPC) – BarcelonaTech, Barcelona, Spain
‡Federal University of Goiás, Goiânia, Brazil.
linamgr@icmc.usp.br, smartinez@essi.upc.edu, valdemarneto@inf.ufg.br, elisa@icmc.usp.br

***Abstract*—Increasingly adopted in critical application domains, self-adaptive systems (SaS) present a particular ability to modify their behavior or configuration at runtime autonomously. The architectural activity of decision-making in an SaS requires the selection of the best software structures configuration. At the same time, requirements of quality attribute (e.g., interoperability, maintainability, reliability), adaptive capabilities (e.g., self-management, self-organization), control approaches (e.g., centralized, distributed), and human interventions must be balanced. This work presents Four4SaS, a collection of the main rationale and knowledge to architect SaS. Four4SaS' solutions encompass well-known architectural patterns and their possible benefits and drawbacks of their use in SaS. Four4SaS' architectural knowledge was reused for designing a river monitoring SaS. Domain-independent architectural solutions of Four4SaS can be used as an initial backbone to design future SaS and development frameworks for such systems.**

***Index Terms*—self-adaptive system; software architecture; reference architecture; quality attribute; architectural pattern.**

I. Introduction

Self-adaptive Systems (SaS) have increased their importance in the past few years, mostly because of their impact in several critical application domains, such as autonomous vehicles, smart cities, security surveillance, avionics, and health-care. SaS autonomously modify their behavior at run-time responding to operating environment changes [1].

An SaS is usually constituted by two types of systems [2]:

- **Managed systems**, which monitor and affect the external world with which the SaS interacts, and comprise the application logic that provides the system's functionalities; and
- **Managing systems**, which encompass the adaptation logic that deals with one or more concerns, monitor both the environment and the managed systems, and adapt the latter when necessary to achieve SaS goals.

The managing system usually imposes control over the managed systems through the use of autonomic managers [3] or MAPE-K feedback loops that contain components executing activities of *m*onitoring, *a*nalyzing, *p*lanning, and *e*xecuting, and sharing domain and control *k*nowledge [4]. During the architectural process, software architects must consider how control activities are coordinated and deployed, the most significant quality attribute requirements (e.g., performance, reliability, safety, availability, scalability), and the type of adaptive capabilities required by the SaS [5], e.g., context-awareness, situation-awareness, self-configuration, self-healing/protecting, self-optimizing, self-managing, self-organizing, and reflection.

Adaptive capabilities allow decreasing human intervention when systems' modifications are required at run-time. Adaptive capabilities range from changes of specific data types to the reconfiguration of the complete SaS architecture in response to environmental changes, internal faults, unexpected constituents' behaviors, integration of new constituents, new requirements, or changes in business goals. Architects must select the adequate strategies (e.g., architectural configurations, patterns, styles, and tactics, or even technologies) that enable the desired modifications without stopping the system's operations and with the minimum human intervention. Architects' decisions making is a challenging activity during the SaS engineering, since the success or failure of an SaS software project mostly depends upon the correctness of its architecture for considering control activities, adaptive characteristics, and quality attribute requirements. However, the rationale behind architectural decisions is frequently known only by the SaS' software team (including architect, developers, testers), making difficult the reuse of such knowledge in other SaS projects. In this context, one question arises: how can we support architects to design SaS architectures based on the reuse of architectural knowledge?

The main contribution of this work is *Four4SaS*, a collection of four domain-independent solutions that guide and facilitate the architectural design of SaS. In Section II, we identify the recurrent SaS architectural configurations (i.e., arrangements of software structures) and investigate how they have been used to address different control, quality attributes, and adaptivity requirements. Based on these findings, *Four4SaS* is defined in Section III. To show its feasibility, *Four4SaS* was applied to architect a river monitoring SaS (RMS), as detailed in Section IV. *Four4SaS*' solutions were assessed by fifteen SaS architects and results of such evaluation are explained in Section V. Section VI describes threats to the validity and limitations of this work. Finally, Section VII presents the related work highlighting the contribution of *Four4SaS* to state of the art and details further works.

This work was supported by Brazilian funding agencies FAPESP (Grants: 2018/07437-9 and 2017/06195-9) and CNPq (Grant: 312634/2018-8) and by the GENESIS Spanish project under contract TIN2016-79269-R and the "Beatriz Galindo" programme. Authors thank all researchers and practitioners that participated in the evaluation study.
Manuscript received ; revised .

II. MINING SAS ARCHITECTURES

This section presents the research methods conducted to identify *the main software building blocks of SaS architectures and their variation depending upon adaptive capabilities and control characteristics.* This knowledge is the basis of the *Four4SaS*' architectural solutions. Bearing this goal in mind, we performed the following steps:

1) Recurrent architectural solutions were extracted from 13 existing reference architectures (RAs) for SaS. Shortly, a RA presents the most relevant decisions (e.g., selection and arrangement of patterns and tactics, and description of the rationale behind the proposed solutions) for designing software systems architectures in specific domains [6]. RAs for SaS were identified through the conduction of a systematic mapping study following the guidelines found in [7]. The systematic mapping protocol and extracted data used in this work are detailed in [8]. The interested reader is referred to [8] which contains IDs (RA1 to RA13) and the complete reference for each architecture.
2) The recurrent SaS' architectural solutions were extracted from the 13 RAs for SaS listed in [8]. Solutions were classified and summarized according to the main elements of an SaS architecture, namely, information about SaS' constituent systems, control characteristics, adaptive capabilities, and architectural patterns, detailed, respectively, in Sections II-A to II-D, and summarized in Table I and mentioned in the remainder of this section.

A. Constituents Systems of SaS

An SaS is formed of two types of constituent systems: the managed and the managing systems. Each constituent system can present diverse adaptive capabilities; hence it can also be considered as an SaS. This characteristic requires architectures with multiple adaptation levels, i.e., SaS' composition is based on managing systems hierarchies [2], as those proposed by RA2, RA6, RA8, RA10, and RA12.

B. Control Characteristics

Control in SaS is related to the distribution level of the managed and managing systems and the decentralization level of the control activities (i.e., monitoring, analysis, planning, and execution) [2]. Hence, the following control strategies can be adopted to address adaptive capabilities in SaS:

- **Category 1: (The managed and managing systems, and the control activities are centralized)**. This strategy is common in SaS with capabilities as situation-awareness (RA1 and RA5) and self-configuration (RA3, RA6, RA8, and RA10);
- **Category 2: (The managed systems are distributed, while the managing systems are centralized and the control activities are decentralized.)** This category supports capabilities as self-configuration (RA12) and self-management (RA9 and RA11) in SaS;
- **Category 3: (Both managed, and managing systems are distributed, and the control activities are decentralized)**. This strategy has been used in SaS with capabilities of self-management (RA2) and self-optimization (RA7); and
- **Category 4: (Both managed, and managing systems are distributed, and the control activities are full-decentralized)**. SaS with capabilities of self-organization can be addressed using this control category (RA4, RA13).

The implementation of these control strategies is mostly based on the MAPE-K loop and its variations, as detailed in Table I. To represent domain and control knowledge, repositories, ontologies, and conceptual models are commonly used by the RAs to represent such knowledge.

C. Requirements of Adaptive Capabilities

Architectural solutions proposed in RAs have a focus on adaptive capabilities as *situation-awareness, self-configuration, self-management, and self-organization. Reflection* is considered in SaS' architectures with *self-management and self-configuration* characteristics. Most RAs (i.e., 8/13) consider the occurrence of changes, simultaneously, in both managed and managing systems. Different SaS' structures can change, ranging from managed systems' entities (e.g., components, services, or interfaces) to managing systems' plans, policies, and goals. No human involvement allows to execute close adaptations (as in RA1 and RA10), i.e., the SaS themselves manage a number of predefined adaptive actions, and no new behaviors and alternatives can be introduced at run-time [5]. Moreover, most architectures define open adaptations, i.e., new goals, requirements, and policies can be added, and even new adaptable entities can be introduced by humans [5].

D. Architectural Patterns and Quality Attribute Requirements

The most employed architectural patterns to design SaS and address diverse adaptivity and quality attribute requirements are described as follows.

Layers have been widely used regardless of the desired adaptive capabilities. Layers allow complex behaviors through hierarchies (RA1, RA3, RA4, RA5, and RA11). Lower layers implement fast adaptations (i.e., reconfigurations of the managed system), and higher layers are responsible for time demanding adaptations (i.e., selection of the best policy or plan to achieve missions based on current system status). Lower layers in RAs achieve **performance** requirements (RA1), and higher layers address **reliability** properties (RA10). **Interoperability** can also be supported by establishing generic connections between managed systems and managers (RA2 and RA5). Layers' separation of concerns enhances **maintainability** and **modifiability** requirements [9].

Shared-data Repository is commonly presented in SaS architectures with self-configuration, self-management, or self-organization capabilities. It allows access to persistent data, ensuring the **availability** of context, control, configuration, and domain information. Also, it avoids undesired changes on data

TABLE I: Information Extracted from Reference Architectures for Self-adaptive Systems

ID	Constituents of SaS		Control Characteristics of SaS			Adaptivity Requirements		
	Managing system	**Managed system**	**Strategy**	**Control approach**	**Knowledge representation approach**	**Capabilities**	**Reflection**	**Human Involvement**
RA1	IPM system	IPM system	Category 1	Triggering conditions and events	Shared ontology between layers	Situation-aware		
RA2	Touch-points and autonomic managers	Any self-adaptive IT system	Category 3	Hierarchical MAPE-K	Distributed and shared repositories between layers	Management		✓
RA3	The AComponent	Any adaptive component	Category 1	Adaptive component paradigm	Components internal registers	Configuration		✓
RA4	Management SOA-based system	Any SOA-based system	Category 4	Observer/Controller architecture	Central knowledge repository	Organization		✓
RA5	Touch-points and autonomic managers	Any component-based system	Category 1	MAPE-K	Central knowledge repository	Situation-aware		✓
RA6	Reflection-based SaS	Any SaS	Category 1	MAPE-K	Meta model	Configuration	✓	✓
RA7	Grid controller component	Micro-grid	Category 3	Hierarchical MAPE-K	Shared ontology	Optimization		✓
RA8	Meta controller subsystem	mobile robot's control application	Category 1	Epistemic Control Loop	Shared ontology	Configuration	✓	✓
RA9	Runtime environment	Runtime application	Category 2	Hierarchical Feedback Loop	Distributed and shared repositories between layers	Management		✓
RA10	models@run.time system	models@run.time system, CPS, or safety SaS	Category 1	MAPE-K	Multiple models	Configuration	✓	
RA11	Enactors, managers and solvers	System components architecture	Category 2	Hierarchical MAPE-K	Central knowledge repository shared between layers	Management	✓	✓
RA12	Self-adaptive middleware	WSN nodes	Category 2	Hierarchical MAPE-K	Repository by layer	Configuration		✓
RA13	HIIC* component	Any SaS	Category 4	Hierarchical MAPE-K	Shared Repository	Management		✓

* Hierarchical inter-intra collaborative pattern

due to modifications in managed and managing systems, contributing to **modifiability** [9] of SaS (RA2, RA4, RA9, RA11, and RA12). Finally, repositories reduce data exchange between managed and managing systems, and between hierarchies of these systems.

Blackboard and ***Pipes and Filters*** have been applied in SaS' architectures with situation-awareness capability (RA1, RA5). Together, both patterns benefit **performance** since the blackboard allows efficient delivering of data between system's layers [9], and the pipe and filters pattern grants concurrent execution of adaptations by high-level layers [9].

Publish-Subscribe is used in SaS' architectures for self-optimization (RA7) with the following benefits: (i) **performance** to communicate data among SaS entities; (ii) **modifiability** due to low coupling between entities; and (iii) dynamic **scalability**, since managed systems can enter or exit without affecting other parts of the SaS.

Service-Oriented Architecture (SOA) has been applied in SaS with capabilities of self-organization and self-configuration (RA4, RA12). SOA benefits are: - **interoperability** of managed systems; - evolution and dynamic **scalability** of SaS according to new demands of resources; - **availability** of managing systems through their replication for processing monitored data of managed systems. The combination of SOA and ***Enterprise Service Bus (ESB)*** facilitates the **integration** of multiple managed systems, mediating and transferring monitored data to all the managing systems interested in such data.

Master-Slave has been adopted in SaS' architectures with self-management properties (RA11). It benefits **performance** providing low response times to control changes in the managed system.

Broker facilitates mediation, communication, **interoperability** and **integration** of heterogeneous managed systems. Brokers have been considered for SaS with self-configuration requirements (RA12).

Decorator pattern allows behavior adaptations at fine granularity level, e.g., adaptations of components objects or parameters values. Decorator can improve the **reliability** of adaptations in SaS with self-configuration properties (RA12).

HIIC (Hierarchical Inter-Intra Collaborative) proposes hierarchies of MAPE-K loops and their coordination to make possible a decentralized control in SaS [10]. HIIC has been used in SaS with high-level adaptive capabilities as self-configuring or self-management (RA13).

III. FOUR4SAS: ARCHITECTURAL SOLUTIONS FOR SAS

Adaptive capabilities and distributed levels of control and constituent systems have a strong influence at determining SaS' architectures. *Four4SaS* defines four generic and reusable solutions based on the architectural knowledge mined from RAs for SaS. Table II summarizes each *Four4SaS*' solution, detailing information about:

- Control strategies adopted by the solutions (C1 to C4) and mentioned in Sections III.A to III.D;
- Monitored elements in the SaS (i.e., managed systems layer or manager systems layer);
- Reasons ([**R**]) why adaptations are required;
- SaS' elements ([**E**]) that need to be adapted;
- Adaptive requirements that are possible to address with the *Four4SaS*' solutions;
- Adaptation type, depending on whether the architecture allows open or close adaptations;
- Benefits ([**B**]) regarding quality attribute requirements and capacities achieved with the solutions; and
- Possible drawbacks ([**D**]) that the solutions could bring to an SaS.

In this table, in the cases where the reasons for adaptations ([**R**]), elements to be adapted ([**E**]), benefits ([**B**]), and

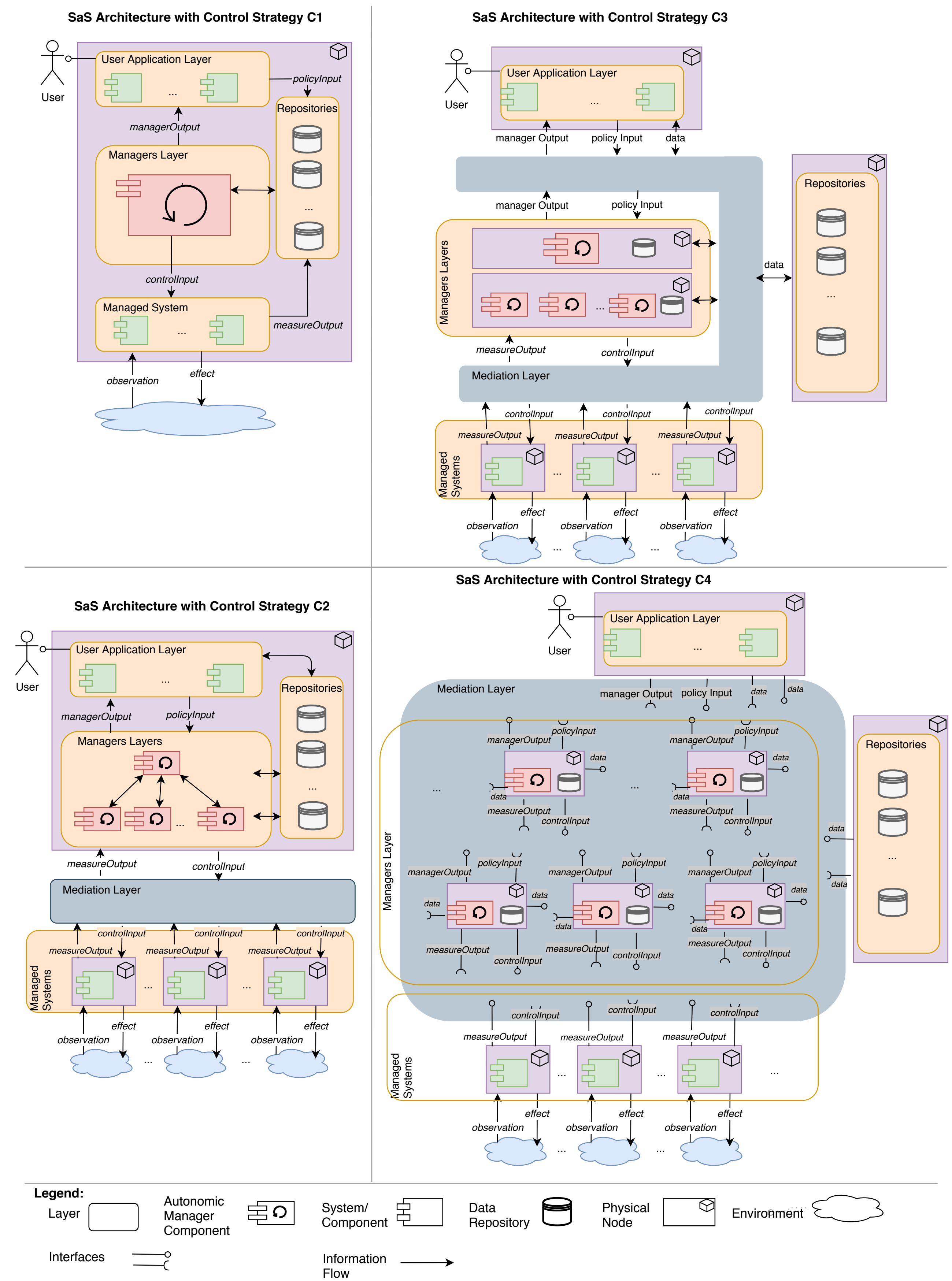

Fig. 1: Architectural solution for SaS with control strategies C1, C2, C3, and C4

TABLE II: Description of *Four4SaS*' Architectural Solutions

***Four4SaS*' Solution**	**Monitored Elements**	**Reason for Adaptation**	**Elements to be Adapted**	**Adaptivity Requirements**	**Adaptation Type**	**Benefits**	**Drawbacks**
Control Strategy C1	Managed systems layer	[**R1**] New adaptation plans or policies; [**R2**] undesired situations detected; [**R3**] installation, update, integration of systems or components.	[**E1**] Adaptation plans or policies; [**E2**] Behavior, states, or configuration of the managed systems;	Situation-aware; self-configuration of managed systems.	Close adaptations	[**B1**] Fast SaS' adaptations; [**B2**] Fast communication of SaS' situations or events. [**B3**] Easy maintainability or modifiability of components.	[**D1**] Monolith architecture; [**D2**] No scalable architecture; [**D3**] Managers can be a single point of failure; [**D4**] Oriented to address individual adaptive capabilities or self-* properties.
Control Strategy C2	Managed and managers systems layers	[**R1, R3**]; [**R4**] Faults discovery or diagnosis.	[**E1, E2**]; [**E3**] Behavior, states, or configuration of systems in the managers layer.	Self-configuration and self-management of the SaS	Close and open adaptations	[**B3]; [B4**] Systems at the managed systems layer can scale; [**B5**] Redundancy of monitored or managed systems; [**B6**] Hierarchies of manager systems allow complex SaS' reconfigurations or adaptations.	[**D3, D4**]; [**D5**] The mediation layer can be a single point of failure; [**D6**] High-level managers are no scalable;
Control Strategy C3	Managed and managers systems layers	[**R1, R4**]; [**R5**] Management of performance and resource allocation.	[**E1, E2, E3**]; [**E4**] Mediation layer.	Self-optimization and self-management of the SaS	Close and open adaptations	[**B3, B4, B5, B6**]; [**B7**] It enables the identification, prevention, and recovery from faults, at all layers, due to problems in physical nodes.	[**D4, D5, D6**]; [**D7**] Possible bottleneck in the mediation layer due the increment of data to be transferred.
Control Strategy C4	Managed and managers systems layers	[**R1**]; [**R6**] Instantiate, activate, deactivate, remove, update elements located in the managers systems layer; [**R7**] Scale elements in the manager systems layer.	[**E1, E2, E3, E4**].	Self-organization and scalability of the SaS	Close and open adaptations	[**B3, B5, B6, B7**]; [**B8**] Various high-level managers can address multiple adaptive capabilities at the same time; [**B9**] Monitored data can be distributed to multiple managers for different purposes; [**B10**] Systems at the managed and manager layers can scale;	[**D5, D7**]; [**D8**] Possible conflict between adaptive properties; [**D9**] Additional strategies for coordinating highly distributed autonomic managers located in the manager systems layer.

drawbacks ([**D**]) are presented in more than one solution, they are described in their first occurrence, and only their codes are used in the text. Architects can use this table as a guide to select the adequate solution for architecting an SaS. Each solution is depicted in Figure 1 and explained as follows.

A. Architectural Solution for SaS with control strategy of Category 1 (C1)

To design SaS with adaptive requirements of situation-aware and self-configuration, architects can structure their systems, as shown at the upper-left of Figure 1. This architecture is adequate for SaS that, based on measures obtained from the managed system, require to make changes in the adaptation plans or policies, or even in the behavior, state, or configuration of such systems. SaS modifications occur when the final user (SaS administrator) updates the adaptation plans and policies, faults are detected, or managed systems change their state and configuration. This architecture allows close adaptations, i.e., the manager system (e.g., autonomic manager) decides which type of reconfiguration should be executed based on policies or plans previously stored in a *shared repository*.

One benefit, shared by all *Four4SaS*' solutions, is to enhance modifications and maintenance of SaS' elements thanks to the low-coupling that *Layers* provide. This first solution also makes it possible to execute fast reconfiguration of the managed system when the control activities (i.e., monitor, analyze, plan, and execute) are organized as a centralized MAPE-K feedback loop that follows the *Pipes and Filters* pattern. Fast communications are as well possible when the *Blackboard* pattern (allocated in the Repositories layer) is used to store and communicate all measures obtained from the managed systems.

A drawback of this solution is that the SaS will have a monolithic and no scalable architecture. Besides, the existence of a unique autonomic manager can result in a single point of failure that could completely stop the SaS' operations. Additionally, the centralized control made by the autonomic manager only allows addressing an individual adaptive property at the time, limiting the type of adaptations that an SaS can perform.

B. Architectural Solution for SaS with control strategy of Category 2 (C2)

The second solution of *Four4SaS* (presented at the bottom-left side of Figure 1) allows adaptive capabilities of self-configuration and self-management. Comparing with the previous alternative, this solution requires monitoring both managed and managers systems. The information obtained from monitored systems is used by high-level managers to define which elements to change, i.e., behavior, state, and configuration of managed systems and low-level managers, due to faults presented by the monitored systems. It is possible to execute close adaptations in an SaS. Open adaptations are also possible since final users (SaS administrators) can send new policies directly to high-level managers. The distributed property of managed systems requires a Mediation layer to support data transfer between managed systems and managers.

One benefit of this architecture is the possibility to scale and replicate managed systems and low-level managers, increasing the reliability of monitored data. This is possible by the allocation of managed systems in distributed physical nodes. Additionally, the distribution of managers, following a *hierarchy of MAPE-K* components, allows to plan and execute more complex adaptations by high-level autonomic managers. This characteristic also favors the SaS' reliability. In this architecture, fast changes are possible when the *Master-Slave* or *Decorator* patterns are used as a basis to organize MAPE-K hierarchies. The combination of *Broker* and *Publish-Subscribe* patterns can be used to structure the Mediation layer, improving interoperability between managed systems and managers and the performance when data is transferred.

One drawback is that Mediation layer can be a single point of failure, disconnecting the managed systems and stopping the SaS operation. Moreover, managers at the high level are not scalable, hindering the achievement of multiple adaptive properties.

C. Architectural Solution for SaS with control strategy of Category 3 (C3)

This solution aims both self-optimization and self-management capabilities. As shown in the upper-right side of Figure 1, the managed and managing systems are distributed. They can be modified at run-time and, together with the Mediation layer, are the elements being monitored. Hierarchies of *MAPE-K components* are allocated in different nodes; hence, control activities are decentralized. This characteristic benefits the execution of complex reconfiguration and the management of SaS performance and resource allocation capabilities. This architecture presents similar benefits to the previous solutions. An additional benefit is the identification, prevention of faults and recovering from the failures caused by the physical decoupling of MAPE-K hierarchies. This property increases the SaS' reliability. An important drawback is the possible bottleneck in the Mediation layer due to the increase of the transferred data [9].

D. Architectural Solution for SaS with control strategy of Category 4 (C4)

The last strategy distributes and monitors both managed systems and managers. Hence, SaS with requirements of high scalability, flexibility to add/activate/deactivate/update elements, and self-organization can use this solution as a blueprint for their architecture.

This solution benefits diverse types of adaptations with distinct complexity performed by fully-decentralized managers deployed in multiples nodes, following the *Hierarchical MAPE-K* or *HIIC* [15] patterns. As a complement, the use of *SOA, Broker, or ESB* favors the scalability [9] of managed systems and managers in an SaS. These characteristics differentiate this solution from the previous three, as depicted at the bottom-right side of Figure 1.

This solution is the most complete among the *Four4SaS*' solutions, enhancing quality attributes of maintainability, reliability, and interoperability, as listed in Table II. Possible drawbacks are related to the simultaneous execution of multiple adaptive requirements that can generate conflicts [5], [11]. This drawback can be overcome through the coordination of highly distributed and hierarchical managers [11]. Additionally, the use of a Mediation layer to communicate data, events, and control can generate problems in the performance and reliability of the SaS.

IV. APPLYING *Four4SaS*: THE RMS CASE

A River Monitoring System (RMS) is responsible for monitoring river levels in a given region. It consists of systems/devices, such as motes, gateways, river monitors, sensor observation services, web map services, and emergency services. Motes and gateways are distributed over the river banks. Each mote measures and communicates its observations (e.g., water level, temperature, pressure, or pollutants) or its configuration information (e.g., description, identification, classification of the mote) to the closest gateway. Gateways establish, at run-time, specific river areas situation, detecting unexpected events (e.g., floods in a region), defining plans to overcome problematic situations (e.g., to restart unavailable motes), and executing such plans. Gateways share a central repository containing control information, such as adaptation plans, emergency situations, motes network configuration, and policies. Gateways communicate information of the river areas and systems situations to *River Monitor* components, which aggregate the information received from all gateways to establish an overall panorama of the river and send this information to the user applications. The River Monitor receives reconfiguration requests from user applications (e.g., requesting water and temperature levels of the entire or parts of the river) and communicates new behavior or configuration policies to gateways and motes. More information about RMS can be found in [12].

Selecting a Four4SaS' Architectural Solution

Before selecting a *Four4SaS*' solution, the RMS must be characterized according to SaS properties defined in Section III and Table II:

- **Monitored elements:** motes and gateways, both distributed over the environment (i.e., river);
- **Managers:** gateways (physically distributed) and river monitors (both possibilities: centralized or distributed);
- **Reasons for adaptations in the RMS:**
  - *Interoperability and Flexibility:* It is required to add, remove, instantiate, deactivate, and activate motes and gateways without requiring manual adaptations;
  - *Scalability:* It is required to continuously add more motes and gateways to cover new river(s) areas independently of the city;
  - *Reliability:* It is important to identify when motes and gateways are unavailable (e.g., hardware problems). Additionally, it is required to detect, prevent faults and recover from faults occurred in motes and gateways; and
  - *Maintainability:* Modifications in policies, requirements, adaptation plans, or even in individual motes and gateways must not affect the RMS operation.
- **Elements to be adapted:**
  - Adaptation plans, policies, and requirements;
  - Motes' behaviors, states, and configuration (e.g., measurement rates of water level);

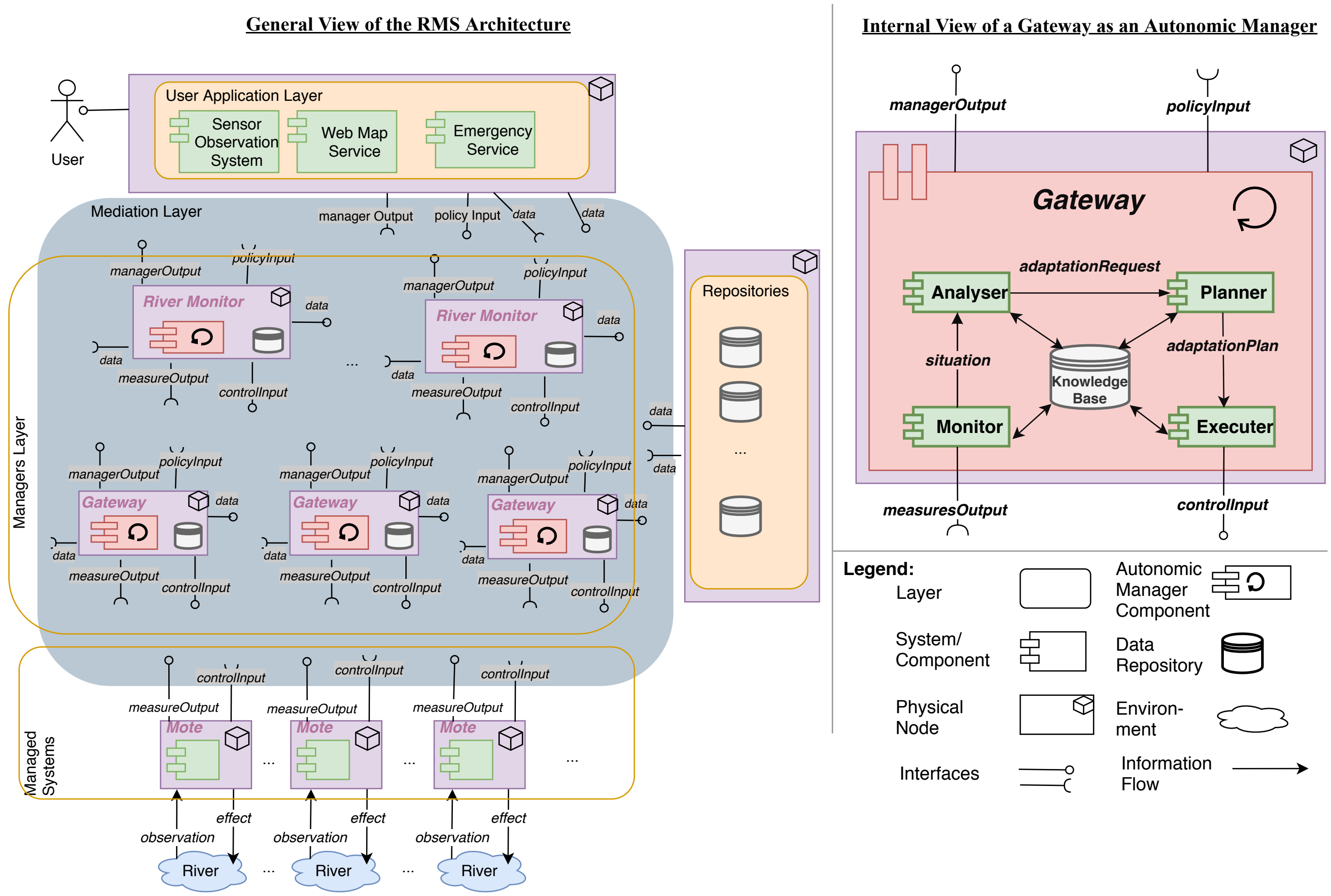

Fig. 2: RMS Architecture as an Instance of the *Four4SaS*' Architectural Solution with Control Strategy C4

  – Gateways' behaviors, states, and configuration; and
  – Capacity of supporting multiple connections and large amount of data transferred among systems.
- **Type of adaptation:** The RMS requires to adopt close adaptations in motes and gateways, and open adaptations (involving final users) to make possible new policies of reconfiguration (e.g., increase in the water levels measurement rates of motes in a region).

Therefore, the RMS' architecture needs to address **adaptive capabilities** of self-configuration and self-organization. Considering that multiple adaptive properties are necessary, managed systems and managers are distributed, and scalability is an important requirement of this SaS, it is required full-decentralization of autonomic managers' control activities. Analyzing *Four4SaS*' solutions in Table II, the best option for the RMS' architecture is to follow the solution with control strategy C4. Figure 2 presents the final RMS architecture as an instance of this solution.

V. Four4SaS Assessment

To assess *Four4SaS*' quality characteristics (e.g., usefulness) to design software architectures of SaS, we ran an evaluation with architects that could use *Four4SaS* in their projects. They were trained in *Four4SaS* and they answered an online questionnaire (available at [8]) designed based on the TAM evaluation questionnaire [13]. The questionnaire included validated constructs from TAM to evaluate five criteria: usefulness, ease of use, demonstrability, feasibility, and the quality of architectures created with *Four4SaS*. An example of using an SaS architecture with *Four4SaS* was discussed during the training, specifically the RMS presented in Section IV.

Fifteen (15) architects participated in this study. They had 2 to 8 years (N=15, Min=2, Max=8, Mean = 4) of experience working with SaS for different application domains, including IoT, embedded, healthcare, crisis and emergency, robotics, banking, and spacial systems.

Answers to each question were scored from 1 (strongly disagree) to 7 (strongly agree). Table III summarizes the results of *Four4SaS*' assessment. For each question, the amount of answers scored in a specific value (from 1 to 7) are given. Score tendencies are highlighted in gray-color scale[1]. For each criterion, aggregated results are presented, detailing the mean, standard deviation, median, mode, and minimum and maximum scores given by respondents.

The majority of responses positively scored all evaluation criteria. We only obtained a negative result for the feasibility criterion, regarding *"All information that is necessary to*

[1] A cell in white color represents that no architect scored a question with such a value

TABLE III: Criteria, Questions and Answers to Evaluate *Four4SaS*.
Amount of participants: 15. Answers were rated in the scale 1 to 7 with the following meaning: 1 - strongly disagree; 2 - moderately disagree, 3 - somewhat disagree, 4 - neutral (neither disagree nor agree), 5 - somewhat agree, 6 - moderately agree, and 7 - strongly agree. Questions marked with $^{(*)}$ were asked to architects in it negative form. Herein, they are presented in its positive form to facilitate results analysis.

Criteria	Question	Amount of Answers by Scale						
		1	2	3	4	5	6	7
Perceived usefulness	Using *Four4SaS* could improve my performance in my job/research	0	0	0	0	3	9	3
	Using *Four4SaS* in my job/research could increase my productivity.	0	0	0	1	4	5	5
	Using *Four4SaS* could enhance my effectiveness in my job/research.	0	0	0	3	7	4	1
	I find *Four4SaS* could be useful in my job/research.	0	0	0	1	5	6	3
	Aggregated Results by Criterion: ***Mean = 6 | St. Dev = 0.87 | Median = 6 | Mode = 6 | Min = 4 | Max = 7***							
Perceived Ease of Use	I could use *Four4SaS* in a clear and understandable way.	0	1	1	0	3	6	4
	Using *Four4SaS* would not require a lot of my mental effort.	0	1	0	2	7	5	0
	I find *Four4SaS* to be potentially easy to use.	0	0	1	3	4	4	3
	I find it is easy to get from *Four4SaS* the knowledge to construct new SaS architectures.	0	0	0	3	6	5	1
	Aggregated Results by Criterion: ***Mean = 5 | St. Dev = 1,16 | Median = 5 | Mode = 5 | Min = 2 | Max = 7***							
Result Demonstrability	I have no difficulty explaining to others the benefits of using *Four4SaS*.	0	0	1	2	3	4	5
	I believe I could communicate to others the consequences (drawbacks) of using *Four4SaS*.	0	0	0	1	2	6	6
	The results of using *Four4SaS* are clear to me.	0	0	1	0	3	8	3
	I would have no difficulty explaining why using *Four4SaS* may or may not be beneficial$^{(*)}$.	1	2	0	3	2	6	1
	Aggregated Results by Criterion: ***Mean = 6 | St. Dev = 1.3 | Median = 6 | Mode = 6 | Min = 1 | Max = 7***							
Feasibility	All information that is necessary to understand *Four4SaS* are available$^{(*)}$.	2	3	3	2	0	4	1
	Making available more information, even if possible, would not be too costly$^{(*)}$.	0	1	0	5	5	2	1
	The quality of the information provided to understand *Four4SaS* is good$^{(*)}$.	0	0	3	1	3	2	6
	All necessary information to fully understand *Four4SaS* can be used$^{(*)}$.	0	0	0	5	2	4	4
	Aggregated Results by Criterion: ***Mean = 5 | St. Dev = 1.6 | Median = 5 | Mode = 4 | Min = 1 | Max = 7***							
Quality of architectures created with *Four4SaS*	The quality of the RMS architecture obtained from *Four4SaS* can be considered as high.	0	0	0	1	3	10	1
	I have no problem with the quality of the RMS architecture.	0	0	0	3	5	5	2
	I rate as excellent the results from using *Four4SaS* to create RMS architecture.	0	0	0	3	1	9	2
	Aggregated Results by Criterion: ***Mean = 6 | St. Dev = 0.87 | Median = 6 | Mode = 6 | Min = 4 | Max = 7***							

understand Four4SaS are available". The 53% (i.e., 8/15) of architects' answered this question with score less or equal to 3 and median of 3 (somewhat disagree).

We also provided the possibility to respondents include free text. The main architects' feedbacks were: (i) systematization of the selection of *Four4SaS*' solutions and support to the architectural decision-making using automated tools; (ii) offering of metrics on possible trade-offs of using *Four4SaS*' solutions in specific projects; (iii) improvement of the *Four4SaS*' description using different architectural views, implementation details, and a web site to link additional information; (iv) automatic code generation based on *Four4SaS*' architectures to support productivity; and (v) use of *Four4SaS* as a basis to product-line architectures in specific SaS domains, e.g., IoT.

VI. Limitations

Threats to search, data, and research validity of our work were mitigated by following the guidelines to conduct systematic mapping study [7] and to execute an on-line questionnaire with SaS architects following the TAM approach [13]. In [8] are presented the protocol, execution, and extracted data of our systematic mapping study, as well as the form and results of conducting the TAM questionnaire.

The main limitation of *Four4SaS* is that its architectural solutions are described in a higher abstraction level. Hence, architects need to refine the *Four4SaS*' structures for achieving more detailed architectures of their concrete SaS projects. To overcome this limitation, *Four4SaS* must be also disseminated to the SaS practitioners and researchers to be used in different projects, making possible the identification of *Four4SaS* variations and possibly complementary solutions.

VII. Final Remarks

Several strategies for designing SaS architectures have been proposed during the last decade. Most of them are focused on investigating feedback loops (as MAPE-K) as core elements of SaS architectures [14], [15], [16], [17]. Researchers also have proposed some patterns for self-adaptation [18], [19] and runtime software evolution [20], some of them oriented to SOA-based SaS [19].

Four4SaS solutions advance the state of the art by bringing to the light reusable architectural knowledge about how to design SaS considering variations of adaptive capabilities, quality attribute requirements, necessity of human involvement to perform open or closed adaptations, and the distribution level of SaS constituents (i.e., managed and managing systems) and control operations. *Four4SaS* solutions are based on evidence obtained from 13 published SaS reference architectures, listed in [8].

As future work, *Four4SaS*' solutions could be formalized in an SaS architectural framework to guide decision-making and allow automatic code generation of SaS. More efforts must be also invested to better understand the trade-offs arising from addressing simultaneously multiple adaptive characteristics in SaS architectures (e.g., following the C4 strategy of *Four4SaS*).

About the Authors

Lina Garcés: is a post-doctorate researcher at the Institute of Mathematics and Computer Sciences (ICMC) of the University of São Paulo (USP). Her research interests include software engineering, (dynamic) software architectures, reference architectures, architectural decision making, software quality, self-adaptive systems, Systems-of-Systems, e-Health, and Ambient Assisted Living. Garcés received a Ph.D. in Computer Science from USP and the University of Southern Brittany (UBS), France, in 2018. She is a member of the IEEE, IEEE SA, SBC, and SBIS. Contact her at *linamgr@icmc.usp.br*.

Silverio Martínez-Fernández: is assistant professor at UPC-BarcelonaTech . His research interests include empirical software engineering, big data, technical debt, maintainability metrics, and software architectures. Martínez-Fernández received a Ph.D in Computer Sciences from the Universitat Politècnica de Catalunya (UPC) in 2016. From 2020, he is a distinguished researcher as part of the "Beatriz Galindo" programme. Contact him at *smartinez@essi.upc.edu*.

Valdemar Vicente Graciano Neto: ia a ternured professor at the Informatics Institute (INF) of the Federal University of Goiás (UFG), Brazil. His research interests include software engineering, information systems, software architecture, model-driven software engineering, and simulation. Neto received a Ph.D in Computer Science from USP and the University of Southern Brittany (UBS), France, in 2018. He is a member of the SBC and SCS. Contact him at *valdemarneto@inf.ufg.br*.

Elisa Yumi Nakagawa: is associate professor in the Department of Computer Systems at USP. Her research interests include software architecture, reference architectures, systems-of-systems, software testing, and evidence-based software engineering. Nakagawa received a in Ph.D. in Computer Science from USP in 2006. She is a member of the IEEE and SBC. Contact her at *elisa@icmc.usp.br*.

References

[1] Peyman Oreizy, Michael Gorlick, Richard Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David Rosenblum, and Alexander Wolf. *An Architecture-Based Approach to Self-Adaptive Software*. IEEE Intelligent Systems. vol.14, 3, pp. 54-62. (1999).

[2] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl Goschka. On Patterns for Decentralized Control in Self-Adaptive Systems. In: Rogerio de Lemos, Holger Giese, Hausi A. Müller and Mary Shaw (eds.). *Software Engineering for Self-Adaptive Systems II*. vol. 7475, LNCS. pp. 76-107. Springer Berlin Heidelberg. (2013).

[3] IBM Corporation. An architectural blueprint for autonomic computing. Technical report, IBM Corporation, p. 1-34, 2005.

[4] Hausi A. Müller, Holger M. Kienle, and Ulrike Stege. *Autonomic Computing Now You See It, Now You Don't*. In Software Engineering, Andrea Lucia and Filomena Ferrucci (Eds.) vol. 5413, LNCS. pp. 32-54. Springer Berlin Heidelberg (2009).

[5] Mazeiar Salehie and Ladan Tahvildari. *Self-adaptive software: Landscape and research challenges*. ACM Trans. Auton. Adapt. Syst. vol. 4, 2, pp. 1-42 (2009).

[6] Eduardo Guerra and Elisa Yumi Nakagawa. *Relating Patterns and Reference Architectures.* In: PLoP. pp. 1-9. (2015)

[7] Kai Petersen, Sairam Vakkalanka, Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, Vol. 64, pp. 1–18 (2015).

[8] Lina Garcés. *Research on Reference Architectures for Self-Adaptive Systems: A Systematic Mapping Protocol and Extracted Data.*. pp. 1-54. (2020). Mendeley Data, v1. https://data.mendeley.com/datasets/w55h5cwyzc/draft?a=c9f71899-d8aa-494d-964a-9a717709dd5f

[9] Paris Avgeriou and Uwe Zdun. Architectural patterns revisited – a pattern language. In 10th EuroPlop, Irsee. (2005), pp. 1-39.

[10] Edith Zavala, Xavier Franch, Jordi Marco, Alessia Knauss, Daniela Damian. SACRE: Supporting contextual requirements' adaptation in modern self-adaptive systems in the presence of uncertainty at runtime, Expert Systems with Applications, Vol. 98, 2018, pp. 166-188.

[11] Danny Weyns. Software engineering of self-adaptive systems: An organised tour and future challenges. Handbook of software engineering. Springer. (2018)

[12] Livia C. Degrossi, Guilherme G. do Amaral, Eduardo S. M. de Vasconcelos, João P. de Albuquerque, Jó Ueyama. *Using Wireless Sensor Networks in the Sensor Web for Flood Monitoring in Brazil*. In ISCRAM, pp. 458-462 (2013).

[13] Davis, F.F.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly, 13(3), 1989.

[14] Danny Weyns and Tanvir Ahmad; *Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review*. ECSA, pp. 249–265 (2013).

[15] Edith Zavala, Xavier Franch, Jordi Marco, Christian Berger. HAFLoop: An architecture for supporting Highly Adaptive Feedback Loops in self-adaptive systems, *Future Generation Computer Systems*, Vol. 105, 2020, pp. 607-630.

[16] Henry Muccini, Mohammad Sharaf, and Danny Weyns. *Self-adaptation for cyber-physical systems: a systematic literature review*. SEAMS, p. 75-81 (2016).

[17] P. Arcaini, R. Mirandola, E. Riccobene and P. Scandurra, *A Pattern-Oriented Design Framework for Self-Adaptive Software Systems*, ICSA-C, 2019, pp. 166-169.

[18] Muccini, H., Spalazzese, R., Moghaddam, M. T., and Sharaf, M. (2018). Self-adaptive IoT architectures: An emergency handling case study. ACM International Conference Proceeding Series.

[19] Hassan Gomaa, Koji Hashimoto, Minseong Kim, Sam Malek, Daniel A. Menascé. *Software Adaptation Patterns for Service-Oriented Architectures*. In SAC. pp.462-469 (2010).

[20] Richard N. Taylor, Nenad Medvidovic, Peyman Oreizy. *Architectural Styles for Runtime Software Adaptation*. In: WICSA/ECSA. pp. 171 - 180.(2009)