Executive Summary of the Thesis

Engineering microservice-based Self-Adaptive Systems: the case of RAMSES

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Authors: Giancarlo Sorrentino, Vincenzo Riccio

Advisor: Prof. Raffaela Mirandola

Co-advisor: Prof. Matteo Camilli

Academic year: 2021-2022

1. Introduction

As the size and complexity of software systems keep increasing, a microservice architecture is often chosen as an architectural style for service-based systems (SBSs). These systems integrate several (micro)services, which are commonly located in multiple computing infrastructures, spread across the world. In this context, the problem of managing such big and complex systems is noteworthy, since direct human intervention is challenging, time-consuming and error-prone.
Self-Adaptive Systems (SASs) tackle this problem by autonomously adapting themselves, in order to achieve user-defined goals in response to changes in the underlying environment or in the system itself, without human intervention [6]. However, SASs are usually challenging and expensive to develop, both in terms of time and resources.
To address this problem, we developed an extensible software framework called RAMSES, a Reusable Autonomic Manager for microSerivcES. The goal of RAMSES is to enforce the satisfaction of user-defined Quality-of-Service (QoS) specifications for the target SBS at runtime, while improving its overall performance. Our work also includes a standalone microservice-based application – SEFA – that serves as a managed subsystem, providing the scientific community with a fully-implemented SAS, which comprises two reusable and independent subsystems.
RAMSES implements a microservice-based MAPE-K loop, and it has been designed to ease the (re)use of the system by a user who wants to adapt a preexisting service-based application. As a consequence, a common technology stack was chosen for the implementation (e.g., Java, Spring).
We corroborate our work with an experimental campaign, to analyse how RAMSES behaves in specific scenarios that synthetically reproduce relevant operating conditions.
Finally, we present the conclusion of our work and the future research directions.

2. Background and Related Work

During the last years, the complexity of software systems started increasing fast. They became no longer restricted to few components located in small and easily controllable areas, but made of a huge number of interconnected and distributed

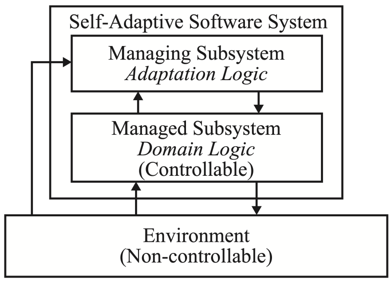

Figure 1: A Conceptual Architecture for Self-Adaptive Software Systems [1]

devices, as in the Internet of Things field. Hence, the need for automatic (re)configuration and optimization mechanisms for those systems arose, aiming at satisfying their admin's goals without human intervention. Self-adaptation is commonly recognised as an effective approach to cope with this need [6, 7].

Self-Adaptive Systems are systems capable of adjusting their behaviour in response to their perception of the environment and of the overall system [2]. As outlined in Figure 1, they promote separation of concerns between a *Managing (Sub)system* and a *Managed (Sub)system*. The Managing Subsystem implements the adaptation logic to manage the Managed Subsystem, which encloses the domain logic. The self-adaptive system operates in an observable environment, which might affect the adaptation logic. The concerns of the managing system over the managed system are known as *adaptation goals* and can be grouped into four categories: self-configuration (i.e., to configure themselves automatically given a set of high-level policies), self-optimisation (i.e., to adapt themselves to improve their performance or cost), self-healing (to detect and repair internal problems), and self-protection (to protect themselves from malicious attacks or harmful problems).

The managing subsystem is often modelled as a feedback loop made of four stages:

1. monitoring (M) of the environment and of the managed subsystem;
2. analysis (A) of the data;
3. plan (P) of the adaptation strategies;
4. execution (E) the adaptation.

This kind of loop, shown in Figure 2, is referred to as MAPE-K, and it involves a shared Knowledge (K) serving as a repository for the data needed by the loop execution. Self-adaptive systems gained more and more attention from the

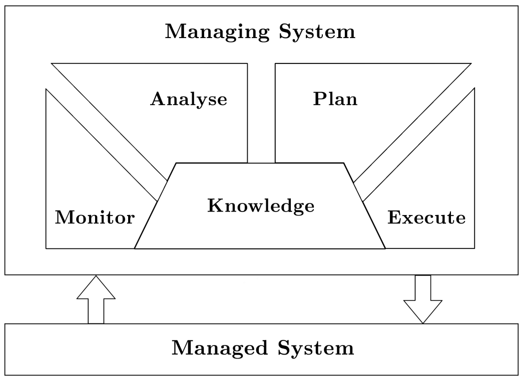

Figure 2: Structure of a SAS implementing a MAPE-K loop (based on [6])

scientific community, whose research focused on different applications of such systems. To analyse the studies conducted so far, the systematic review of SASs proposed in [7] by Wong et al. was used as a reference. In addition, some exemplars published on the website on *Software Engineering for Self-Adaptive Systems*[1] were examined.

Recent surveys and systematic reviews [4, 6, 7] stress the problem of engineering such systems. The ideal solution to build a Self-Adaptive System would be to design both the Managing and the Managed System together, optimizing their interaction: indeed, the Managing subsystem would be tailored to the Managed one, that in turn would be designed to efficiently and effectively interact with the former. However, this approach is expensive, both in terms of time and resources.

Frameworks like RAINBOW [3] address this problem by proposing a reusable infrastructure, which still requires some effort to fully understand how to set it up and combine it with a pre-existing web or service-based application. Moreover, it needs both a formal definition of the model – using the ad-hoc *Stitch* self-adaptation language – and a translation infrastructure for communications from and to the managed system, which implies both a high design and development complexity.

When dealing with SBSs, different Self-Adaptive Systems usually share part of their adaptation goals. More specifically, they usually address the problem of dynamically ensuring some Quality-of-Service (QoS) specifications while optimizing the overall system's performance. Differently

[1]`http://www.self-adaptive.org/exemplars`

Scenario	Observable Properties	Examples of Adaptation
S1: Violation of QoS specifications	Values of the QoS indicators of the service over time (e.g., availability, average response time)	– Change the current service implementation – Add instances in parallel – Shutdown of an instance with low performance – Change configuration properties
S2: Service unavailable	Success or failure of each service invocation	– Change the current service implementation – Add instances in parallel
S3: Better service implementation available	Properties of the service implementations	– Change the current service implementation

Table 1: Adaptation scenarios

from the studies analysed during our work, our solution is not limited to domain-specific adaptations or to a specific technology stack but aims at providing a flexible and reusable Managing System, focused on dynamic QoS management and optimization for different SBSs.

Our domain of interest is the one in which an already existing service-based application needs to be extended with a managing subsystem to perform self-adaptation, implementing a managing subsystem. In this case, the best solution would be to design an ad-hoc Managing System, that suits the specific domain and needs of the Managed one. However, this approach introduces tight coupling in the overall SAS, which could lead to maintainability issues and increasing costs.

To overcome these drawbacks, we propose the engineering of a reusable self-adaptive system. To bring this approach to fruition, we have implemented:

- SEFA, a SErvice-based eFood Application implementing the managed subsystem;
- RAMSES, a Reusable Autonomic Manager for microServicES implementing the application logic.

As the name suggests, SEFA is a Java-based microservice application that allows customers to browse a list of restaurants and their respective menus, choose some dishes, and finally place orders, paying for them by credit card and getting them delivered to a specific address.

On the other side, RAMSES represents the managing subsystem, and it is made of configurable and extendable components, whose behaviour does not depend on the specific Managed System to be adapted. The design of RAMSES was driven by the adaptation scenarios described in Table 1.

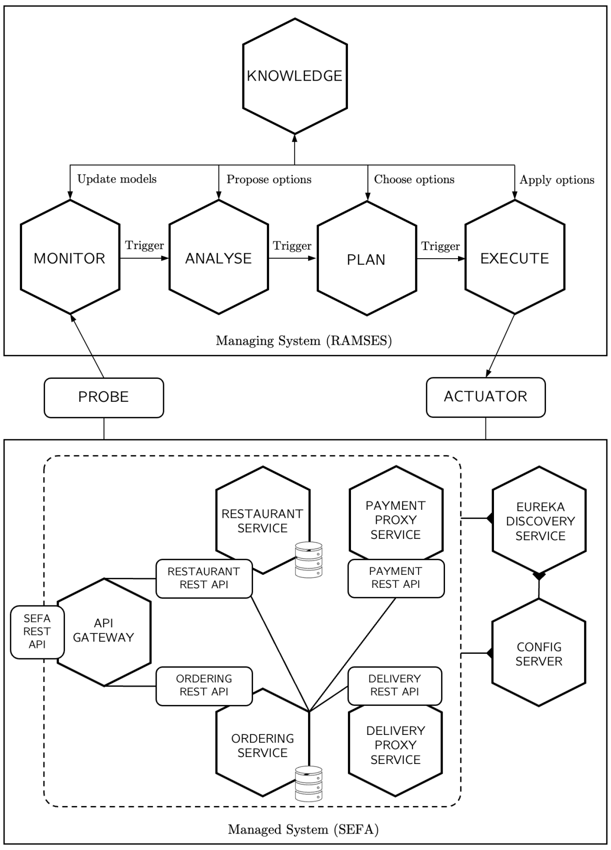

Figure 3: High-level architecture of the proposed SAS

The complete architecture is summarized in Figure 3.

3. SEFA

To test the proposed solution and to build a microservice-based system that could be easily reused and adapted for research purposes, we developed SEFA. It is implemented in Java and developed using the Spring Boot and Spring Cloud frameworks. As the architecture is designed according to the *microservice pattern*, the server-side logic is made up of multiple microservices, which expose REST APIs using the JSON format. Such services are the following:

- the *Restaurant Service*, in charge of managing the restaurants available on the application and their respective menus;
- the *Ordering Service*, in charge of managing all the customers' carts, and of allowing them to place their orders; it interacts with the Restaurant Service and with the two proxy services (*adapter pattern*) to reach its goal;
- the *Payment Proxy Service*, in charge of mediating with a third-party payment service provider (i.e., who processes the payment) during the elaboration of the order;
- the *Delivery Proxy Service*, in charge of mediating with a third-party delivery service provider (i.e., who delivers the order to the customer) during the elaboration of the order.

Since SEFA highly depends on the service providers, the *circuit breaker pattern* is used, in order to improve the resilience of the application and to mitigate the impact that potential failures or abnormal response times of the third-party services may have on the end-user.

To exploit the advantages brought by the microservice architecture, multiple instances of each service can run in parallel. This introduces the problem of applying the same configuration to all the instances of the same service, that has been addressed by using a centralised configuration server.

Being in a distributed setting, in order to discover the location of the instances and to choose the one to make a request to, client-side service discovery and load-balancing are applied. The load balancer algorithm used is based on *fitness proportionate selection*, also known as *roulette wheel selection* [5].

Finally, in order to ease the communication, SEFA includes a single entry point in charge of processing and routing the end users' requests to the internal microservices, according to the *API gateway pattern*.

4. RAMSES

RAMSES is a managing subsystem designed as a distributed MAPE-K loop, implemented in Java using the Spring Boot framework. Each stage of the loop is implemented as a standalone microservice and is described in its respective section.

As anticipated before, the main characteristic of RAMSES is being reusable with different service-based Managed Systems. To enforce this feature, RAMSES is designed to interact with the Managed System via two components: a Probe component, that allows RAMSES to retrieve all the relevant data from the Managed System, and an Actuator component, that allows RAMSES to effectively perform operations on the system. These components must be provided together with the Managed System itself, and must offer a specific set of APIs, defined by RAMSES.

4.1. Knowledge

The Knowledge component is the one holding the system model. For this reason, it is the source of truth for the entire loop. Indeed, the Knowledge interacts with the other loop components to maintain and provide them with an up-to-date runtime model of the system.

4.2. Monitor

The Monitor component is in charge of collecting data from the Managed System periodically. At each iteration it queries the Probe component provided by the Managed System, asking to perform a snapshot of all the instances of each service. Each snapshot contains statistics about the resource usage of the instance (e.g., CPU and disk usage), about the processed HTTP requests (e.g., response time, number of errors) and about the circuit breakers, if any.

The Monitor routine runs periodically and asynchronously with respect to the rest of the loop: it accumulates all the snapshots in a temporally ordered buffer, and it stores them in the knowledge-base as soon as a new loop iteration starts. When this happens, the Analyse component is notified to start its iteration.

4.3. Analyse

The Analyse component is in charge of processing the latest snapshots of all the service instances and of updating the Knowledge with the new values for the QoS indicators handled by RAMSES, for all the services and their instances. Furthermore, it is in charge of determining the adaptation options to force or pro-

pose for each managed service.
The analysis starts as soon as this component is notified by the Monitor. The first step is to analyse the status of each instance i. If i is considered suitable for the next steps, the latest snapshots are processed in order to compute a new value for each QoS property.
If undesired behaviours are detected during the analysis of a service, RAMSES may impose some *forced* adaptation options, that will be applied in any case at the end of the current loop iteration. Otherwise, for each QoS indicator, their latest values are processed in order to determine whether a service requires adaptation. If so, the Analyse component proposes some adaptation options, that will be evaluated during the Plan stage.
For each service implementation s, the current RAMSES implementation includes four different types of adaptation options:

- the *Add Instance* option, which represents the action of adding a new instance of s;
- the *Shutdown Instance* option, which represents the action of shutting down the specific instance it refers to;
- the *Change Implementation* option, which represents the action of replacing the instances of s with instances of the service implementation specified by the option;
- the *Change Load Balancer Weights* option, which, for a service balanced using a *fitness proportionate selection* algorithm, represents the action of redistributing the weights associated with all the running instances of s.

Future versions of RAMSES may extend this list by including new adaptation options.

4.4. Plan

The goal of the Plan component is to determine which are the best adaptation options among all the ones proposed during the current loop iteration. For each managed service, if there is at least one forced option, the non-forced ones are discarded, while all the forced ones are directly chosen. Conversely, all the options are processed and compared to extract the one estimated to bring more benefits to the service it refers to.
The benefit of each option is computed by estimating the value that each QoS indicator is expected to have after applying option. When a service is load balanced using a *fitness proportionate selection* algorithm, the weights of the instances of the service involved are modified depending on the option to apply:

- when an instance should be added, the Plan assigns a fraction of the total weight to the new instance, resizing the weight of the other instances;
- when an instance should be shut down, the Plan equally redistributes its weight among the other instances;
- when the service implementation should be changed, the Plan equally splits the total weight between the instances of the new service implementation;
- when the weights of the running instances should be changed, the Plan redistributes the instance weights by solving a mixed integer linear programming (M-ILP) optimization problem.

4.5. Execute

The Execute component retrieves from the Knowledge all the adaptation options chosen by the Plan during the current loop iteration, if any. For each of them, according to their type, the Execute contacts the Actuator component to effectively apply the changes required by the considered adaptation option.
The Execute component, and consequently RAMSES itself, assumes that all the operations requested to the Actuator are eventually executed, and that all the changes of service configurations are performed within a reasonably short amount of time.

5. Experimental Evaluation

An experimental campaign was conducted in order to assess how RAMSES behaves in specific scenarios that synthetically reproduce relevant operating conditions, while at the same time determining the impact that its configuration parameters have on the overall adaptation process. In particular, SEFA was selected as the system to be managed by RAMSES.
In order to create an experimental environment in which the managed services exhibited unwanted behaviours (e.g., QoS constraints not satisfied, high failure rate, etc.), we synthetically injected issues in the managed system by manipulating the services instances, artificially slow-

ing their execution or causing failures. Moreover, to analyse the results in a quantitative way, we adopt the so-called QoS Degradation Area (QoSSDA) indicator. Given a managed service and the plot of its values for the selected QoS indicator, we define the QoSDA as the total area between the considered QoS threshold and its corresponding QoS value trend, in the portion of the graph where the QoS specification is not satisfied.

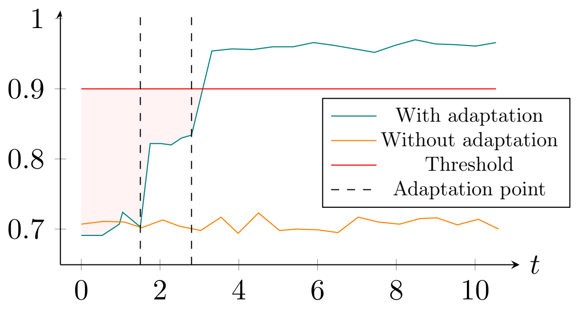

Figure 4: Restaurant Service availability

With respect to the adaptation scenarios proposed in Table 1, we noticed that our system is capable of performing adaptation options that actually improve the performance of SEFA. As shown in Figure 4, concerning scenario S1 we experienced an improvement in the QoSDA value for the availability of the Restaurant Service (i.e., the red area) of 88%, with respect to the case in which no adaptation is performed.

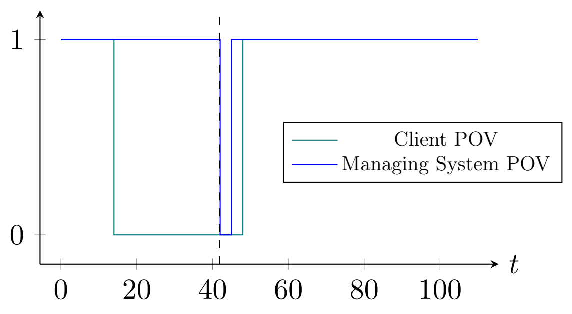

Figure 5: Payment Proxy Service running instances

As shown in Figure 5, concerning scenario S2, we assessed that RAMSES is actually capable of detecting failures, replacing the unavailable instances when needed. Moreover, it is also able to effectively detect the instances prone to end up in a failed or unreachable status, replacing them with new ones.

Finally, concerning scenario S3, we assessed the self-optimization capabilities of RAMSES. In our experiment, RAMSES improved the performance of the system by changing the service implementation of the Payment Proxy Service, which increased its availability from ≈ 0.96 to ≈ 1.0, even if its requirement was already satisfied.

To conclude our campaign, a final experiment was conducted, replacing SEFA with a different managed system, composed by two services:

- the *Randint Producer Service*, which generates random integer numbers upon request;
- the *Randint Vendor Service*, which contacts the Randint Producer Service to obtain a new random integer.

As shown in Figure 6, RAMSES was able to adapt the new system and make it satisfy its specification, without requiring any modification to its adaptation logic.

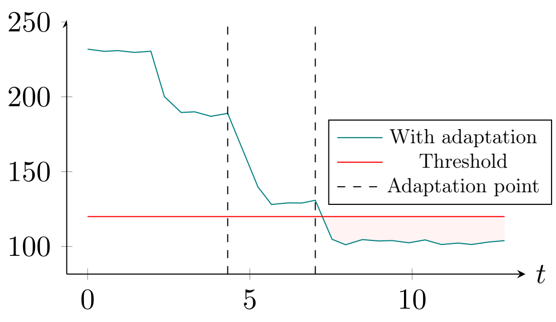

Figure 6: Randint Vendor Service average response time

6. Conclusions and Future Work

Developing Self-Adaptive Systems is a challenging task. Their development process is cost and time demanding, whether developing both the Managing and Managed subsystems from scratch or starting from a preexisting application to be adapted.

Even if some works have already addressed this problem by proposing a reusable infrastructure to ease the engineering of SASs (e.g., the RAINBOW framework [3]), they are not ready-to-use, due to their abstraction and generality, and also a significant amount of time and effort is required to set them up.

RAMSES aims at providing a Managing System that is easily reusable for service-based applications. Given a probe and an actuator component

offering specific interfaces and satisfying a set of prerequisites, RAMSES is able to adapt different applications without changing its managing logic. Moreover, since it was designed according to the microservice architectural pattern, it also allows to exploit the advantages brought by the use of this pattern, such as modularity, decoupling and easy maintainability.
The proposed solution is a first attempt at realising a modular and reusable Managing System, which is open to further improvements. Future versions of RAMSES could propose a deeper analysis process or extend its set of adaptation options. In particular, an enhanced analysis routine could also take into account other metrics (e.g., resources usage and circuit breakers metrics) in order to build more reliable indicators of the managed services' performances.
At the time of writing, the adaptation options to apply are chosen by RAMSES according to the benefit they are estimated to bring to the system. Further versions of RAMSES could encompass a more complex decision-making process, enriching the existing one by taking into account the costs and the risks deriving from the application of an adapt option. Moreover, the benefit estimation could also consider analysing the history of performed adaptation options, using machine-learning techniques to quantify the actual benefits they brought to the managed system.

References

[1] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. Software engineering processes for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 51–75. Springer, 2013.

[2] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pages 1–26. Springer Berlin Heidelberg, 2009.

[3] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[4] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[5] Melanie Mitchell. *An Introduction to Genetic Algorithms*, pages 124–125. The MIT Press, 1999.

[6] Danny Weyns. Engineering self-adaptive software systems – an organized tour. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 1–2, 2018.

[7] Terence Wong, Markus Wagner, and Christoph Treude. Self-adaptive systems: A systematic literature review across categories and domains. *Information and Software Technology*, 148:106934, 2022.