Towards a Resiliency Decision Framework for Microservices

Johannes Busch
Faculty IV, Dept. of Computer Science
Univ. of Appl. Sciences&Arts Hannover
Hannover, Germany

Andreas Hausotter
Faculty IV, Dept. of Computer Science
Univ. of Appl. Sciences&Arts Hannover
Hannover, Germany
Email: Andreas.Hausotter@hs-hannover.de

Arne Koschel
Faculty IV, Dept. of Computer Science
Univ. of Appl. Sciences&Arts Hannover
Hannover, Germany
Email: Arne.Koschel@hs-hannover.de

***Abstract*—Microservices build a deeply distributed system. Although this offers significant flexibility for development teams and helps to find solutions for scalability or security questions, it also intensifies the drawbacks of a distributed system. This article offers a decision framework, which helps to increase the resiliency of microservices. A metamodel is used to represent services, resiliency patterns, and quality attributes. Furthermore, the general idea for a suggestion procedure is outlined.**

Keywords*—*Microservice; Resiliency; Software Architecture.

I. Introduction

Microservices are a current trend in software development. They divide complex systems into several independent and lightweight services [1]. This approach has several benefits over traditional architectures like monoliths or Service-Oriented Architectures (SOA). One major point is the independence of these services at runtime and in development. Furthermore scalability, security or equivalent questions can be answered on a per service basis, which offers a higher degree of flexibility [2][3].

Besides these benefits, microservices also come with various challenges, one important of them being their distributed nature. To compute non-trivial business functions, several microservices have to work together. Thus, they have to communicate over networks, which cannot guarantee complete availability. Modern cloud based systems further increase this problem by their volatile nature.

Key contribution of this article is a decision framework, which offers suggestions to increase the resiliency of specific microservices. To achieve this goal services and resiliency patterns need to be put in perspective. This is done through the resiliency decision framework metamodel, a novel way to describe services, patterns, quality attributes and their interconnections. Furthermore, a suggestion procedure will be presented. Its purpose is to analyze service requirements and pattern effects to compute a list of suggestions to strengthen these requirements.

In this article resiliency is defined as the ability of a software to handle failures in a meaningful way or recover completely from it without human intervention. The context for resiliency in this article is based upon safety and not security. The definition is based upon [4][5].

This work was developed under the *Competence Center Information Technology and Management* (CC_ITM), which is part of the *University of Applied Sciences and Arts Hannover*. Its main objective is the transfer of knowledge between university and the insurance industry.

The remainder of this article is organized as follows: First related work is presented in Section II. An application scenario based on the german insurance sector is presented in Section III. The core of the Resiliency Decision Framework is described in Section IV. Section V summarizes this article and gives an outlook on future work.

II. Related Work

While resiliency is important in microservice based architectures, it is certainly not a new topic in general. In Service-Oriented Architectures resiliency is often realized by means of fault tolerant services. These can be designed either through specific middleware [6][7] or alternative implementations [8]. Furthermore, fault tolerance can be evaluated over several services [9], which can increase the resiliency of complex business functions or processes.

Another field besides fault tolerance are self healing systems. Self healing can be achieved either through internal techniques [10] or external components [11]. Furthermore, self healing can be build into the architecture itself [12].

In [13], a catalogue of resiliency patterns is described. Furthermore, a framework for resiliency in high performance computing is defined. In contrast we provide a full resiliency decision framework for microservices, which is a novelty to the best of our knowledge.

III. Application Scenario

To stress the importance of resiliency in microservice-based systems and to evaluate the suggestions given by the Resiliency Decision Framework following application scenario will be used. It is based on prior CC_ITM [14] work. The described `Partner Management System` is expanded further by a business process from the reference architecture for the German insurance companies (VAA) [15].

An overview of the application scenario is given in Figure 1. It consists of several microservices grouped into different systems. Each system represents a bounded context as described

 ISBN: 978-1-61208-844-0

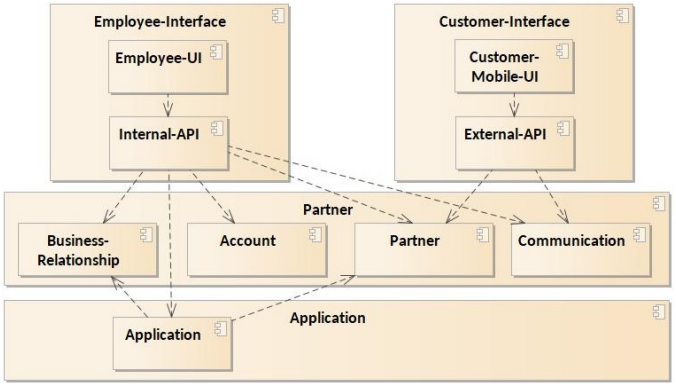

Figure 1. Overview of the application scenario

by Evans in [16]. The goal was to develop a realistic and complex enough system, which could be found at partner companies of the CC_ITM.

The `Partner Management System` offers several CRUD operations to manage insurance partners. It is used throughout the scenario. The `Application System` implements the business process `Manage Application` ('Antrag verwalten') from the VAA. It processes applications for a new insurance contract. Thus it retrieves information about partners and adds a new insurance contract, if an application finishes successful. The `Employee-Interface System` contains a UI and API for internal use. It offers a comprehensive access to all insurance systems and is the sole way to work with the implemented business process. Customers and external companies can access insurance information through the `Customer-Interface System`. Both UIs are designed as modern Single Page Applications.

IV. Resiliency Decision Framework

The goal of the `Resiliency Decision Framework` (RDF) is to suggest resiliency design patterns based upon the quality requirements of the analyzed microservices. To achieve this goal the different patterns, services and quality requirements need to be presented in a well defined way. This is done through a metamodel. To evaluate these information and provide a list of suggestions a suggestion procedure is described.

A. Resiliency Decision Framework Metamodel

An overview of the `Resiliency Decision Framework Metamodel` is given in Figure 2. Base is the `Framework` element, which is the root for a given RDF instance. The metamodel consists of three major parts:

`Quality Tree`: Services and patterns are described through a quality attributes defined in a quality tree. The `QualityTree` element offers, for example, to define a resilient specific one. The metamodel structure is based on well know quality trees like ISO 25010 [17].

`Services`: The `Service` element describes a specific microservice. Each service has several characteristics, which

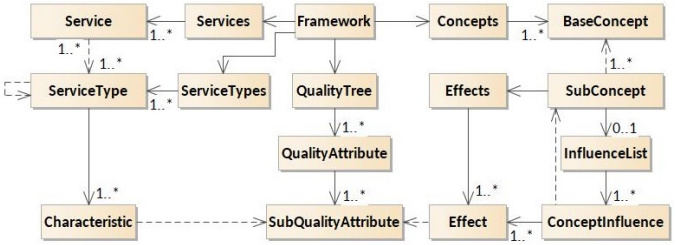

Figure 2. Overview of the developed metamodel

correspond to a specific requirement described by an quality attribute. Each characteristic also contains a numeric value, representing its importance. Currently these have to be defined by the architect. `ServiceType` elements are used to organize these characteristics and enable reusability in complex systems.

`Concepts`: This element defines resilient patterns in a consistent way. Each pattern consists of a base concept and one or more sub concepts. The base concept describes a general problem and solution (for example, Monitoring). A Sub concept narrows this to a specific pattern (for example, Logging) with its effects on service quality and requirements. The effect is described through a numeric value. Currently these are given by the authors based upon research into the corresponding patterns. Furthermore, influences amongst patterns can be defined in order to describe how well certain patterns work together.

B. Resiliency Decision Framework Suggestion Procedure

The `Suggestion Procedure` analyzes a defined model and computes a list of resiliency pattern suggestions for each defined service. It consists of several steps to determine and enhance the list of suggestions for a given service. Especially the filter steps are highly configurable. All steps are based on the numeric values given to pattern effects and service characteristics. Currently these represent qualitative values and not quantitative ones based on concrete measurements. The different steps are:

1) `Collect requirements`: Collect the requirements for a service by collecting all service types.
2) `Determine positive patterns`: Determine all resiliency patterns, which have a positive effect (for example, a corresponding positive numeric value) on at least one of the service requirements.
3) `Domain filter`: Filter these patterns by domain rules, for example, that would cause a negative impact onto the service requirements.
4) `Threshold filter`: Additionally filter these patterns by thresholds, for example, the combined negative effects of a pattern. A simple approach to combine these effects is to add corresponding numeric values but more complex approaches (for example, weighing) are also possible.
5) `Sorting`: Evaluate each pattern against the service requirements and sort the pattern list based on this eval-

 ISBN: 978-1-61208-844-0

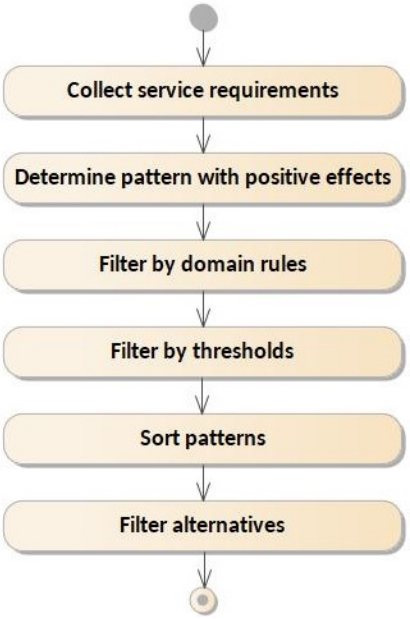

Figure 3. Overview of the developed procedure

uation. A simple approach for evaluation is to combine the positive and negative effect values in correspondence to the service requirements.

6) `Filter alternatives`: Filter less fitting pattern alternatives to diversify the list of suggestions, for example, filter less fitting approaches to load balancing.

These steps will be repeated for each service defined in the Resiliency Decision Framework model. An overview of the procedure is given in Figure 3.

C. Applying the Resiliency Decision Framework

To apply the Resiliency Decision Framework to a microservice based system, several activities have to be performed. Besides configuring the procedure filters, a comprehensive pattern catalogue has to be developed. This was done by a comprehensive literature analysis into software patterns. Furthermore, the different microservices have to be defined in the framework model.

After applying the procedure (cf. Figure 3) to this model, the last activities are implementing the suggestions and evaluating the improved microservices. These activities especially have to be repeated each time the catalogue of patterns or the service requirements change. All these activities form the Resiliency Decision Framework process.

How the microservices are evaluated, is out of scope of this contribution. One approach could be the QoS Measurement Model described in [18].

D. Resiliency Decision Framework Example

As described above, the basis for all suggestions is the definition of a service. For example, the partner service needs to be highly available because of its central role. Also it has to offer minimal latencies for user interfaces and external partner companies. Resiliency patterns are analyzed by the framework procedure based on these service `Characteristics`. A possible list of patterns could include automatic scaling, escalation and monitoring. These are ordered by their positive impact onto the service requirements. Besides this, all effects of required patterns and pattern influences are also analyzed.

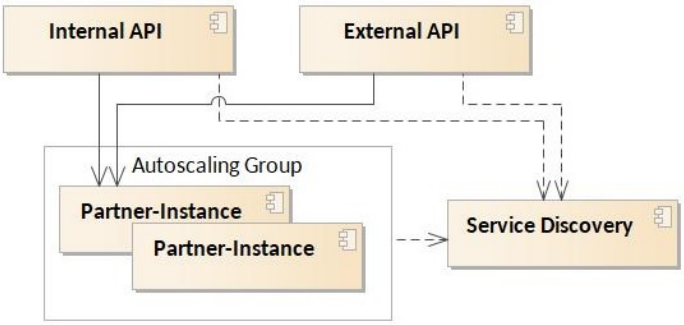

Figure 4. RDF applied onto the Partner service

`Automatic scaling` increases the availability of the service, but has a list of requirements. To distribute the requests and support scaling, a load balancing pattern is needed. Several load balancing patterns are known, thus the suggestions procedure evaluates the different alternatives. An external load balancer would, for example, add another hop, thus load balancing based upon service discovery is suggested by the framework.

The enhanced `Partner-Service` is given in Figure 4. The partner service is now deployed through an autoscaling group. Requests to a partner service a distributed through a service discovery, which could implement different algorithms (for example, round robin). To minimize the impact on client services, the service discovery should implement DNS as its API. Thus, no changes to the API services are needed.

V. Conclusion and Future Work

The basis for the `Resiliency Decision Framework` was described in this article. By using a well-defined metamodel, the RDF can be applied to different sectors and areas. The `Suggestion Algorithm` uses the metamodel to create a list of diverse suggestions with maximum positive effect on the resiliency of a microservice.

Future work will revolve around the development of a quality tree for software resiliency. Furthermore, an extensive library of resiliency patterns will be developed. An evaluation will be done by applying the framework to the complete application scenario. The quality of the suggestions is directly dependent on the quality of the service requirements and pattern catalogue definitions. Thus, a way to evaluate the quality of service requirements and pattern definitions need to developed.

References

[1] M. Fowler and J. Lewis, “Microservices a definition of this new architectural term,” [retrieved 11, 2020]. [Online]. Available: https://martinfowler.com/articles/microservices.html

 ISBN: 978-1-61208-844-0

[2] E. Wolf, *Microservices: Grundlagen flexibler Softwarearchitekturen*. dpunkt.verlag, 2016.
[3] M. Richards, *Microservices vs. Service-Oriented Architecture*. O'Reilly Media Inc., 2016.
[4] U. Friedrichsen, "Unkaputtbar: Eine kurze Einführung in Resilient Software Design," *Business Technology Magazin*, vol. 19, 4 2014.
[5] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, V. Sekar, "Gremlin: Systematic resilience testing of microservices," in *Proc. 36th IEEE International Conference on Distributed Computing Systems*, 2016, pp. 57–66.
[6] Z. Zheng and M. R. Lyu, "A qos-aware middleware for fault tolerant web services," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, 2008, pp. 97–106.
[7] I. Chen, G. Ni, and C. Lin, "A service-oriented fault-tolerant environment for telecom operation support systems," in *2008 IEEE International Symposium on Service-Oriented System Engineering*, 2008, pp. 208–214.
[8] A. S. Nascimento, C. MF. Rubira, R. Burrows, F. Castor, and P. HS Brito, "Designing fault-tolerant soa based on design diversity," *Journal of Software Engineering Research and Development*, vol. 2, no. 1, p. 13, 2014.
[9] K. Peng and C. Huang, "Reliability evaluation of service-oriented architecture systems considering fault-tolerance designs," *Journal of Applied Mathematics*, vol. 2014, pp. 1–11, 01 2014.
[10] A. Carzaniga, A. Gorla, A. Mattavelli, and N. Perino, "A self-healing technique for java applications," in *Proc. ICSE '12: International Conference on Software Engineering*, 2012, pp. 1445–1446.
[11] . K. Ravi and V. Sathyanarayana, "Container based framework for self-healing software system," in *Proc. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004. FTDCS 2004.*, 2004, pp. 306–310.
[12] Y. Qun, Y. Xian-chun, and X. Man-wu, "A framework for dynamic software architecture-based self-healing," in *Proc. 2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, 2005, pp. 2968–2972 Vol. 3.
[13] S. Hukerikar and C. Engelmann, "Resilience design patterns: A structured approach to resilience at extreme scale," *Supercomputing Frontiers and Innovations*, vol. 4, 08 2017.
[14] A. Koschel, A. Hausotter, M. Lange, and S. Gottwald, "Keep it in sync! consistency approaches for microservices: An insurance case study," *Service Computation*, pp. 7–14, 10 2020.
[15] GDV, "Vaa-fachliches referenzmodell," [retrieved 07, 2017]. [Online]. Available: http://www.gdv-online.de/vaa
[16] E. Evans, *Domain Driven Design — Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.
[17] iso25000.com, "Iso/iec 25010," [retrieved 3, 2021]. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=0
[18] A. Hausotter, A. Koschel, , J. Busch, and M. Zuch, "A generic measurement model for service-based systems," *Service Computation*, pp. 12–18, 2 2018.

 ISBN: 978-1-61208-844-0