Modelling and Evaluation of Microservice Granularity Adaptation Decisions

by

Sara Hassan

A thesis submitted to the
University of Birmingham for the degree of
Doctorate of Philosophy

School of Computer Science
College of Engineering and Physical Sciences University of Birmingham
March 2019

Abstract

Microservices have gained wide recognition and acceptance in software industries as an emerging architectural style for autonomic, scalable, and more reliable computing. In this thesis we target a critical, main problem related to the transition towards microservices: **reasoning about the suitable granularity level of a microservice (i.e. when and how to merge or decompose microservices).** We conduct a systematic mapping study and use it to identify inadequacies in the state-of-the-practice and -art related to this problem. The thesis addresses the following inadequacies: a relatively disciplined understanding of the transition to microservices and technical activities underlying it, systematic architecture-oriented modelling support for microservice granularity, a dynamic architectural evaluation process to reason about the cost and added value of granularity adaptation, and effective decision support to inform reasoning about microservice granularity at runtime.

To address the identified inadequacies, initially we contribute an architecture-centric modelling approach for microservices. Next, we contribute a dynamic evaluation process which links granularity adaptation to its added value under uncertainty. Next, we contribute an interactive, iterative planning engine to provide insight regarding which granularity adaptation strategy is suitable at runtime. We use a hypothetical microservice application — Filmflix — as a case study for evaluating each aforementioned contribution. Finally, this thesis contributes to microservice-specific guidance aiming to render scalability-aware granularity adaptation decisions. We evaluate this contribution by comparing its usage against ad-hoc scalability analysis.

To the soul of my late grandfather . . .

Acknowledgements

Firstly, I am extremely grateful for having Dr. Rami Bahsoon as my teacher and supervisor both in undergraduate and postgraduate studies. Without his continuous patience, flexibility, and support I would not have gained the opportunity for a PhD degree nor widen the scope of my research and gain more knowledge and skills throughout my degree. His guidance and care my research make me extremely thankful for working under his supervision over the past six years. Beside my supervisor I would like to thank all the respected academics I worked with during my research for there constructive feedback. I would like to thank Prof. Xin Yao and Dr. Dave Parker — my thesis group members — for their continuous feedback regarding my research. I would like to thank Dr. Leandro Minku, Dr. Nour Ali, and Dr. Rick Kazman for their effort, time, fruitful discussion and insightful comments regarding papers we co-authored. Also, I would like to thank my colleagues from the Software Engineering Research Group: Dalia Sobhi, Paola Yanez, Carlos Gomez, Dr. Abdessalam Elhabbash, and Dr. Maria Salama for always being there to answer my queries and advise me generously.

Last but not the least comes my family. Without my parents' belief in me, tolerance for my worries, and support in every possible way, I would not have made it to writing those lines. Without my grandparents' pride of me I would not have made it this far. For this and much more, I could not be more grateful for my family.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Several industries have recently migrated to or consider migrating to microservices [93, 324, 145, 273, 158, 268, 306, 353, 63, 268, 148]. Microservices are a more fine-grained and autonomic form of services [148] where we consider services as an entity that logically represents a business activity with a specified outcome [1]. A service is self-contained, a black box for its consumers, and it may consist of other underlying services [1]. We acknowledge that there are other definitions of services in the literature but we consider this definition to be fitting for the context of this thesis.

The difference between service-oriented and microservice architectures is still an open research area. At a high level, the microservices architectural style has its core aligning an industry's business objectives with technical ones thereby introducing added value to the architecture. Underlying this alignment is promoting more service autonomy and decentralised governance in the architecture than that called for in service-oriented architectures.

Servitization in the business domain is a shift towards utility-based engineering for software products into services; the migration is continuously motivated by delivering value to end users [309]. We therefore view the migration to microservices as a form of servitization where services/components are transformed into microservices — — to introduce added value to the software architecture.

The transition involves isolating business functionalities into microservices that interact through standardised interfaces; the isolation aims at optimising the autonomy and replaceability of the service(s). It can also promote autonomous management

(i.e., decentralised governance) of the service(s), better traceability, accountability and auditing for the service(s) and their provision in the event of failure. These are examples of value added to the software architecture through isolating business functionalities into microservices. This added value can be introduced as a result of the microservice architecture's ability to cope with operation, maintenance and evolution uncertainties. Ultimately, this can also relate to improved maintenance costs and cost-effective quality of service (QoS) provision to end users. Microservice adoption can be motivated by improving cost-effective QoS provision along a plethora of attributes including but not limited to: performance, scalability, and/or reliability.

Despite the industrial push and the accelerated research rate in microservices, to our knowledge there is no disciplined understanding to the trend [142]. *One of the thesis aims is to advance this understanding by consolidating the views from academia and industry about microservices. Consequently we provide a working definition for the transition towards microservices hoping to serve as a starting point to guide practitioners in the field. This consolidation of various views regarding the transition is a pre-requisite for addressing the main problem of concern in this thesis:* ***reasoning about the suitable granularity level of a microservice***. A granularity level determines "the service size and the scope of functionality a service exposes [194, p.426]." Granularity adaptation entails merging or decomposing microservices thereby moving to a finer or more coarse grained granularity level. This problem is critical because "splitting (microservices) too soon can make things very difficult to reason about. It will likely happen that you (the software architect) will learn in the process [93]." This problem is of significance to both brownfield and greenfield development [93] since it affects how an abstract architecture can be refined to a concrete configuration leveraging microservices.

Another aim of this thesis is to provide systematic architecture-centric modelling support for microservices to reason about their granularity at a suitable level of abstraction. This support should: 1) treat microservice boundaries as first-class entities and 2) facilitate defining (at design time) candidate granularity adaptation strategies. Each granularity adaptation strategy can be driven by introducing added value to the architecture along different dimensions (e.g., replaceability, decentralised

governance). This strategies can be translated into code and embedded in the microservice architecture at design time. Nevertheless, it is essential to facilitate modelling the strategies first at a higher level of abstraction to cater for the heterogeneity of development tools used to develop the concrete microservices. Our modelling support aids design time compilation of a bank which includes several alternative granularity adaptation strategies modelled at a suitable level of abstraction. This bank is not exhaustive since it is not possible to anticipate at design time every microservice whose granularity adaptation can introduce added value to the architecture. It is relatively easier to predict at design time (through expert knowledge, historical data and/or benchmarks) the most critical microservices whose granularity adaptation adaptation can introduce added value to the architecture. Those critical microservices are the targets of the granularity adaptation strategy bank compiled at design time using our modelling support. In other words, we are concerned in this thesis about cases where granularity adaptation is a strategic, potentially value-bearing exercise rather than a stylistic, straightforward one.

Given candidate granularity adaptation strategies, *the next aim of this thesis is to contribute to an objective, runtime architecture evaluation process which links granularity adaptation to its added value predictions (which consider both potential benefit compared to the initial architecture and cost of the exercise) under uncertainty. The aim of this approach is to address a crucial question: Is it worth pursuing microservice granularity adaptation in the first place?*

Microservice architectures operate in highly dynamic, large-scale environments. In that context, enforcing even a simple architectural design decision can introduce significant benefit and/or cost. Therefore, answering the aforementioned question is constrained by the runtime scenario in which the architecture is operating (e.g., a holiday season for an online shopping application versus an out-of-season period of the year). The runtime scenario affects the predictions of cost and benefit associated with granularity adaptation. Therefore, we attempt to make our evaluation process track and update these predictions in response to changes in the runtime environment. Changes of concern are related to fluctuations of independent variables (e.g., workload, costs, energy consumption) rather than emergent changes in end user requirements.

In essence, these changes are sources of uncertainty that we account for in the added value predictions.

Given that pursuing granularity adaptation would introduce value to the microservice architecture in a particular runtime scenario, *the next aim of this thesis is to provide the architect with an insight regarding which granularity adaptation strategy (or strategies) are suitable in that scenario.Therefore, this thesis aims to orchestrate our modelling support along with the architecture evaluation process. This orchestration yields an iterative decision support system to aid microservice adopters in reasoning about microservice granularity adaptation dynamically at runtime. The decision support system should aid the architects to 1) determine whether it is worth pursuing granularity adaptation under uncertainty at runtime and then, 2) direct the software architect(s) to suitable granularity adaptation strategies or provide them with runtime evidence to re-visit unrealistic strategies they defined at design time.* The decision support system would take as input the bank of strategies defined at design time rather than develop the suitable strategy on-the-fly at runtime. In essence, granularity adaptation when supported by our decision support system would be planned rather than on-the-fly. Our approach conducts what-if analysis at runtime on an up-to-date model of the running architecture (captured using our modelling support) to output the suitable granularity adaptation strategies (if any). Consequently, enforcing the suitable granularity adaptation strategy at runtime on the running architecture involves code refactoring, configuration, securing communications, and implementing data translation layers among other activities.

Among the main adopters of microservices are highly competitive industries (e.g., Netflix [268], Amazon [353], and BBC [145]) which are characterised by scale. Therefore, any decision support proposed for reasoning about microservice granularity adaptation consider the scale of microservice architectures. It is therefore essential to systematically analysing extent to which a decision support system can render scalability-aware granularity adaptation decisions. *Therefore, the last aim of this thesis is to provide microservice-specific guidance for scalability analysis hoping to render scalability-aware decision support for microservice granularity adaptation.*

1.1 MOTIVATING CASE STUDY

We use a hypothetical online microservice application — Filmflix — as a case study to highlight the significance of the problems addressed in this thesis. The architecture and utilities driving the transition to microservices and architects' intuition in Filmflix are inspired by the Netflix's experience in adopting microservices [268, 336, 356]. Netflix is a large-scale, early microservice adopter therefore we consider it a neat representative for our case study in this thesis. In particular, we consider utilities driving microservice adoption here to be enhancing Filmflix performance and its compliance to social regulations. In reality, much more orthogonal utilities are behind the adoption of microservices in Filmflix. We consider Filmflix to be operating on a similar scale of Netflix.

Filmflix allows users to upload written movie reviews after they pass a regulation system. The regulation involves looking for a set of a predefined blacklist of "foul" terms in this review. The initial architecture of Filmflix contains three microservices: ReviewRegulation — implementing the regulation of movie reviews, ReviewUpload — managing the user input requirements when submitting a review, and MovieReview — capturing input from the user through an interface and uploading a review that passes the regulation. We consider that the transition to microservices in this application is driven by enhancing Filmflix performance where the expected runtime workload (i.e. number of reviews sent to Filmflix) is high compared to competing applications. The application is designed by two software architects which possibly have different opinions regarding how Filmflix should be designed. At design time, the initial architecture is visualised using Figure 1.1; the notation used is inspired by "context diagrams" from [233, 112]. Each bounded context represents an independent area of the firm's domain. Independence implies that this domain has consistent rules "in terms of team organisation, usage within specific parts of the application, and physical manifestations such as code bases and database schemas [112]" and these rules only apply within that bounded context. Each modular boundary represents a single microservice within a bounded context. The intuition here is that a modular boundary encapsulates concretely related functionalities. Associations between the respective

modular boundaries represent the coordination of the microservices together to provide the functionality of Filmflix. Even though the initial design of Filmflix in Figure 1.1

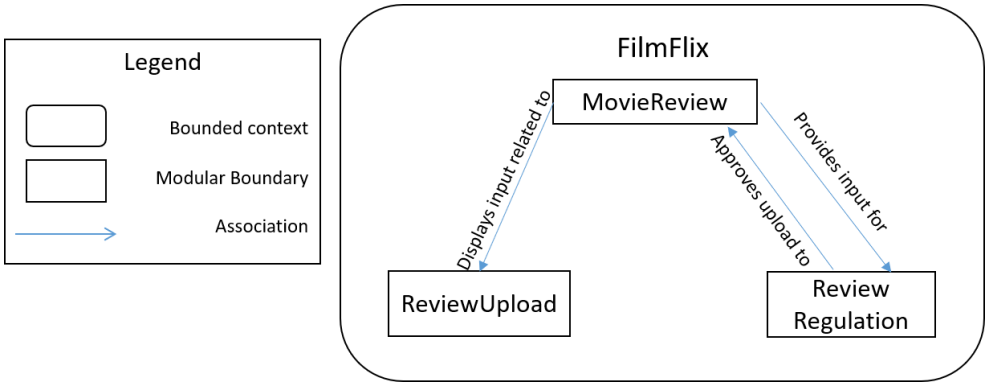


Fig. 1.1 Initial design time Filmflix architecture; a bounded context is an independent area in the firm's domain, a modular boundary is considered a single microservice and an association is the cooperation between modular boundaries to achieve Filmflix's functionality

can provide a neat logical separation (i.e. granularity level) of functionalities into stable microservice boundaries, the initial design might not continue to be suitable at runtime. To avoid aggressive decomposition of microservices, it is essential to reason systematically about whether and how microservices need to be decomposed or merged given the current runtime scenario of Filmflix.

Although the notation in Figure 1.1 can be intuitive to Filmflix architects, it does not benefit reasoning about the granularity of Filmflix. In particular, it does not treat microservice boundaries as adaptable first-class entities nor does it support formulating alternative granularity adaptation strategies in the form of triggers and steps for each strategy.

Even if there is the modelling support described above, reasoning about granularity can not be done in isolation from its benefit and cost, both of which depend on the runtime scenario in which Filmflix is operating. At runtime, the initial Filmflix architecture is monitored to understand the runtime scenario which might require adopting one of the candidate strategies. Subsequently, a model of the running architecture captured using a suitable architecture definition language (ADL) and we assume that this model is continuously synchronised with the running Filmflix

architecture. This model is used to conduct runtime analysis to answer the following two questions below given a runtime scenario:

1. Does adapting granularity of Filmflix add more value to its revenue streams (i.e., in terms of retaining end users, obtaining higher subscriptions, and/or entering more markets) compared to the value added if granularity level is kept unchanged? The added value in this case is the difference between predicted benefit and maintenance/re-structuring costs of adopting any particular granularity level.

2. If there is more added value in pursuing granularity adaptation, then which candidate granularity adaptation strategies (if any) are suitable for a given runtime scenario?

Answering this questions is not straightforward but rather a strategic exercise that can have economic impacts on Filmflix revenue streams. Consider that Filmflix has a diverse audience from various cultural background and ages. Netflix architects have come to the conclusion that *ReviewRegulation* can have economic benefits for some end users: implementing extensive regulation in *ReviewRegulation* can give Filmflix certification from compliance bodies; this can be displayed to end users. Consequently, such certification can provide assurance about the quality and compliance of the reviews to norm, age group, culture etc. On the other hand, *MovieReview* is the microservice which solicits and uploads movie reviews from end users; if the end user experience provided by MovieReview is attractive, it can act as "influencer" that can turn an end user from a viewer to a Filmflix subscriber. *MovieReview* provides movie reviews to *ReviewRegulation*, which in turn gives the final decision to MovieReview regarding the acceptance or rejection of uploading each review.

While implementing extensive regulation in *ReviewRegulation* have economic benefits, it can slow down *MovieReview* performance and hurt the end users' experience, specifically in situations when Filmflix would experience high demand - e.g., peaktime, holiday seasons, popular/new movies etc. This can have negative consequence on one revenue stream (related to turning viewers into Filmflix subscribers),

though it would be highly advantageous for the revenue streams related to providing assurance to end users.

Netflix architects come up with two candidate adaptation strategies that aim to manage the trade-off between *MovieReview* performance and *ReviewRegulation* compliance:

- Merging *MovieReview* and *ReviewRegulation* to reduce the latency across them thereby thriving for performance. Latency can be critical in peak times and/or for popular movies when Filmflix is expecting to experience a high volume of reviews; henceforth, it may likely slow down turning viewers into potential subscribers.

- Decomposing *ReviewRegulation* into multiple microservices where each one extensively implements rules related to a specific geographical region. This strategy offload can promote Filmflix for getting more region-specific certifications while balancing for the performance of *MovieReview*.

Each strategy can bring economic benefit to Filmflix and there is a cost of pursuing each strategy. Cost factors include but are not limited to: data migration costs, code refactoring costs, and network link costs. The difference between benefit and cost is the added value of pursuing granularity adaptation. However, this is only one side of the calculation; the added value of granularity adaptation needs to be compared to the added value of keeping the current granularity level unchanged. It could be the case that added value of keeping the current granularity level is higher than that of pursuing granularity adaptation. Therefore, it is essential to objectively answer the question of: When is it worth pursuing granularity adaptation in the first place?

Once this question is answered, then the Filmflix architects need to determine the suitable granularity adaptation strategy; this highly depends on the runtime scenario in which Filmflix is operating. Each aforementioned strategy is modelled at design time as a candidate granularity adaptation strategy. At runtime, the initial Filmflix architecture is monitored to understand the runtime scenario which might require adopting one of the candidate strategies. Subsequently, a model of the running architecture captured using a suitable architecture definition language (ADL) is used

to reason about the candidate strategies given the solicited runtime scenario. For example, when Filmflix experiences high demand at runtime, it is reasonable to retain as many end users as possible by ensuring their experience is smooth and focussing on Filmflix performance. In such a runtime scenario, the first candidate strategy above might be the more suitable one; this is output of reasoning about granularity at the modelling level at runtime. Subsequently, it is up to the architects how to enforce this strategy on the running architecture.

The suitable granularity adaptation strategy needs to consider the scalability of Filmflix. For example, the dimensions affecting Filmflix's scalability include volume of data shared across microservices and the geographical spread of end users. If only the volume of shared data is considered when adapting granularity, then a suitable strategy might be merging microservices to reduce the data exchange calls across them. If only the geographical distribution of the end users is considered, then decomposing microservices to align them with geographical spread of end users might be more suitable. Therefore, both dimensions need to be considered simultaneously in order to suggest a scalability-aware granularity adaptation strategy.

Reasoning about the suitable granularity level of Filmflix microservices in the context above highlights a number of challenges:

1. Filmflix architects lack the fundamental knowledge of the transition to microservices and hence the drivers for granularity adaptation.

2. The notation used to visualise Filmflix does not treat microservice boundaries as adaptable first class entities or facilitate reasoning about granularity adaptation at a suitable level of abstraction and systematicy.

3. The context above lacks an objective architectural evaluation process which objectively answers the following question considering the dynamic, uncertain environment of Filmflix: Is it worth pursuing microservice granularity adaptation in the first place?

4. The context above lacks a systematic, effective decision support system which suggests granularity adaptation strategies that can introduce added value compared to the initial Filmflix architecture.

5. The scale at which Filmflix operates necessitates scalability-awareness of the aforementioned decision support system; the suitable granularity adaptation strategy is one that can introduce added value even at the scale of Filmflix.

It is worth noting that the problems above assume that the functional requirements of Filmflix are met regardless of the granularity level.

Addressing the first problem involves probing for a better, more disciplined understanding of the transition and technical activities underlying it. This is a prerequisite to understand the core drivers behind granularity adaptation. In other words, such an understanding can determine how the added value of granularity adaptation could be manifested (e.g. in the form of better replaceability, traceability and/or scalability of Filmflix).

The second problem above motivates the first step to reasoning about the microservice granularity adaptation: modelling at design time the initial architecture and a candidate solution space of granularity adaptation strategies with varying granularity levels. This modelling should be a reasonable level of abstraction to account for the heterogeneity of development tools used to develop the concrete microservices in Filmflix. The heterogeneity of tools supporting microservice architectures calls for flexibility in modelling the granularity decision problem in a technology independent way. Once such modelling support is provided, it can be used to model a realistic view of the current microservice architecture at runtime.

The third problem above motivates a value-driven view [49] of granularity adaptation. Addressing the third problem requires an objective architectural evaluation process which links granularity adaptation to its added value which accounts for both the predicted benefit and cost of granularity adaptation. These predictions highly depend on the runtime environment in which the architecture is operating [351]. Therefore, they need to account for, track and update uncertainties that are caused by fluctuations in the runtime environment (e.g., in the volume of received reviews).

Addressing the fourth problem calls for an effective, systematic, objective decision support system that suggests suitable granularity adaptation strategies given a candidate bank of strategies and a runtime scenario of Filmflix. Each candidate strategy

can have a different rationale (e.g., designing for failure, aligning microservices with team structure, or mirroring the data structure of the architecture). The candidate bank is compiled at design time and fed as input to the decision support system. What-if analysis can be done at runtime on the model of the running architecture to determine the suitable candidate granularity adaptation strategies. The analysis is done given monitoring data the running Filmflix architecture. Monitoring is required to reflect the current runtime scenario of Filmflix. Reasoning on the model rather than the running architecture is cautious approach to avoid potential contentions of merging and/or decomposing microservices on-the-fly. Once the suitable granularity adaptation strategy (if any) is determined on the modelling level, it is up to the Filmflix architects how it is executed in the running architecture (i.e. whether or not the microservices of concern need to be taken off-line to merge or decompose them). Figure 1.2 summarises our briefly described vision regarding reasoning about microservice granularity adaptation at runtime. In this thesis we attempt to contribute to each green box in Figure 1.2.

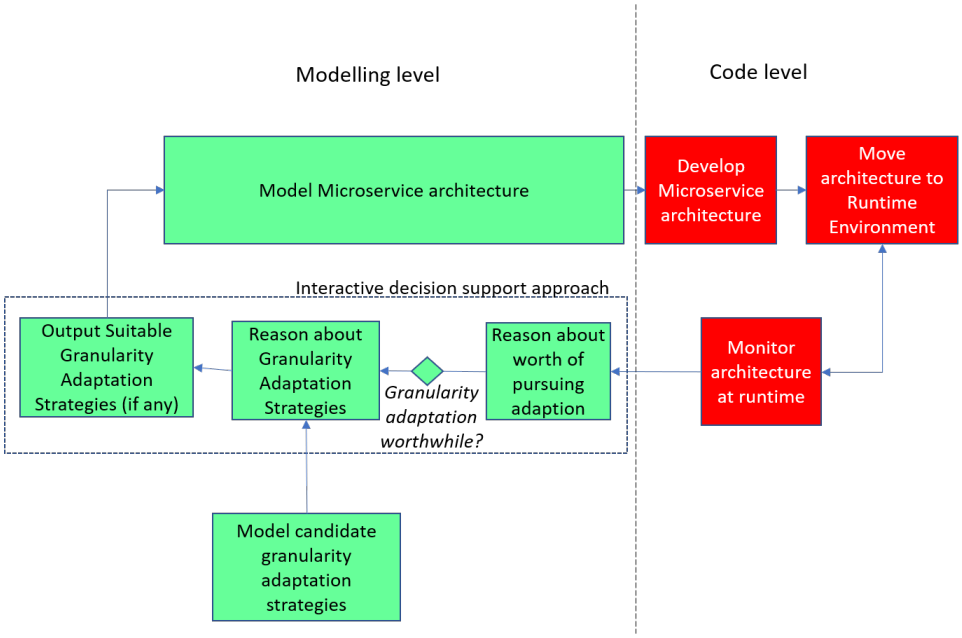


Fig. 1.2 Overview of our vision regarding roles of modelling support and interactive decision support system in the context of runtime microservice granularity adaptation; red boxes are outside the scope of our contributions; green boxes are addressed by contributions of the thesis

The fifth problem above motivates the need to consider the dimensions which can affect the scalability of Filmflix hoping to render a scalability-aware granularity

adaptation decision. Therefore, it is essential to identify and systematically analyse the dimensions and metrics important for the scalability of Filmflix. Moreover, it is essential to systematically analyse the extent to which decision support for granularity adaptation can scale to consider the necessary input dimensions. Therefore, the fifth problem calls for microservice-specific guidance for scalability analysis hoping to render scalability-aware decision support for microservice granularity adaptation.

We assume in this thesis is reasoned about on the modelling level rather than on the running architecture, this is suitable in cases where executing the adaptation without taking the microservices off-line can be risky. For example, in safety-critical microservice applications (e.g., in the health domain) it is still essential to reason about the suitable granularity adaptation strategy in the context of a given runtime scenario. However, any potential malfunctioning due to executing this strategy while the architecture is running can have critical ramifications (e.g., delay in detecting a life-threatening medical condition). Therefore, it is more reasonable to isolate the microservices to be adapted, test them off-line after executing the adaptation strategy (through both unit and regression tests), then re-integrate the adapted microservices into the running architecture.

1.2 PROBLEMS ADDRESSED BY THE THESIS

Motivated by the case study in Section 1.1, we formulate the problems addressed in this thesis as follows:

- *Problem 1 — inadequacy of a relatively disciplined understanding for the transition to microservices.*
- *Problem 2 — inadequacy of an architecture-centric modelling concept that captures microservice boundaries as adaptable first class entities and facilitate reasoning about the microservice granularity adaptation problem by allowing modelling different strategies for microservice granularity adaptation.*
- *Problem 3 — inadequacy of an objective architectural evaluation process that links granularity adaptation to predictions of the added value it can enable under uncertainty and allows updating these predictions at runtime.*

- *Problem 4 — inadequacy of a systematic, effective decision support system for reasoning about microservice granularity adaptation at runtime which suggests granularity adaptation strategies that can enable added value to the microservice architecture.*

- *Problem 5 — inadequacy of a systematic approach for analysing the extent to which a decision support system can render scalability-aware granularity adaptation decisions.*

1.3 RESEARCH QUESTIONS

Given the problems defined in Section 1.2, we aim to address them in this thesis by answering the following research questions:

- *Research Question 1: providing a better, more disciplined understanding of the transition to microservices* — how can the state-of-the-art and practice in microservices be probed to advance the understanding of microservices and whether the transition to them has an impact on development tools, deployment management and/or personnel organisation etc.?

- *Research Question 2: formalising the microservice granularity adaptation problem* — how can microservices be modelled using flexible, expressive support that aids defining different strategies for runtime microservice granularity adaptation?

- *Research Question 3: objective reasoning about the added value of architectural design decisions regarding microservice granularity* — can the trade-off between predicted benefit and cost of pursuing microservice granularity adaptation (or not) be objectively reasoned about and tracked under uncertainty?

- *Research Question 4: realising microservice granularity adaptation* — how can microservice granularity adaptation can be effectively engineered at runtime, combining systematic modelling of granularity adaptation strategies with objective reasoning about added value?

- *Research Question 5: analysing the extent to which it is possible to render scalability-aware granularity adaptation decisions* — how is it possible to systematically analyse the extent to which a decision support system can render scalability-aware granularity adaptation decisions?

1.4 RESEARCH METHODOLOGY

To address the questions in Section 1.3, this thesis adopts a classical research methodology inspired by the design science research methodology (DSRM) [255]. The following five steps that are carried out iteratively to guide our research in this thesis.

- *Identifying thesis problem:* The first step is to understand the problem domain (i.e. the transition to microservices). For this purpose, a systematic mapping study has been conducted. This study also helped identify rooms for improvement in the current state of the art and practice of defining the granularity of a microservice architecture. As the understanding of the problem domain is advanced, the target problem of this thesis — reasoning about microservice granularity — is identified and formulated in the form of the research questions.

- *Identifying thesis objectives:* Given the identified problem, the next step is to determine the objectives of its solution. We formulate these in the forms of Research Questions 1-5 in Section 1.3.

- *Design and developing thesis contributions:* To achieve the identified objectives, this thesis extends an aspect-oriented meta-modelling concept [258, 13] to contribute a modelling approach designed to suit reasoning about microservice granularity. Furthermore, this thesis combines binomial real options theory [330, 20, 248, 329] — which captures the value of design under uncertainty — with the theory of Bayesian surprises [44] to dynamically evaluate if it would be worth pursuing granularity adaptation. Finally, this thesis orchestrates our modelling and evaluation processes to develop the main objective of this thesis — decision support for reasoning about microservice granularity.

- *Demonstrating thesis contributions:* To motivate the designed and developed contributions, this thesis uses the hypothetical microservice application — Filmflix (described in Section 1.1) — as a case study. For each contribution, a motivating scenario derived from Filmflix is described which sets the contribution's context. After describing each contribution, the respective motivating scenario from Filmflix is used to demonstrate the contribution's usage.

- *Evaluating thesis contributions:* to evaluate each demonstrated contribution, a controlled experiment is set derived from the contribution's application to Filmflix. Qualitative and/or quantitative evaluation of the experiment results is used to highlight the unique benefits of the thesis contributions to reasoning about microservice granularity.

1.5 THESIS CONTRIBUTIONS

Throughout the steps described in Section 1.4, this thesis views reasoning about microservice granularity levels as a key challenge of the transition of microservices. This thesis targets the problem of reasoning about microservice granularity by aiming to provide systematic, effective decision support for reasoning about microservice granularity. To this end, the main contributions of this thesis are:

- *A systematic mapping study aiming for better, more disciplined understanding of the transition to microservices* and technical activities underlying it. We term the transition and technical activities leading to microservice architectures as microservitization.

- *A formalisation of the microservice granularity adaptation problem* that entails an extension of an aspect-oriented meta-modelling concept [258, 13] — ambients — by introducing microservice ambients and granularity adaptation aspects. Microservice ambients provide the primitives for modelling microservices and treat microservice boundaries as adaptable first-class entities. Granularity adaptation aspects provide the primitives for defining the runtime granularity adaptation strategy of an ambient. In particular, this aspect allows

the architect to define (at design time) objective triggers for granularity adaptation and distinct steps of merging or decomposing microservice ambients if the triggers are met at runtime. In other words, each granularity adaptation aspect systematically captures a strategy for adapting from one granularity level to another at runtime.

- *A novel dynamic value-driven architecture evaluation process* — It combines binomial real options theory [330, 20, 248, 329] — which captures the value of design under uncertainty — with the theory of Bayesian surprises [44]. Bayesian surprises enable updating added value predictions at runtime in response to runtime evidence variable values (e.g. workload variations). Therefore, our evaluation process allows dynamically determining if it would be worth adapting granularity at runtime from the perspective of introducing added value.

- *An interactive, iterative, value-driven planning engine for microservice granularity adaptation* that incorporates our evaluation process and microservice ambients to provide effective, systematic, objective decision support for microservice adopters at runtime. The planning engine uses our evaluation process to indicate whether is it worth pursuing granularity for a runtime iteration of the planning engine. It then provides software architect(s) with recommendations on suitable adaptation strategies or on revisiting unrealistic strategies defined at design time using microservice ambients.

- *An attempt for a provisional catalogue of the microservice-specific scalability dimensions and metrics* which need to be considered when designing any decision support for reasoning about microservice granularity adaptation. We compile this catalogue by digesting the state-of-the-practice in analysing and/or discussing the scalability of existing microservice granularity decision support.

- *An application of systematic scalability analysis to reasoning about microservice granularity adaptation* — the application utilises scalability goal-obstacle analysis [104, 91] which is inspired by Keep-All-Objectives-Satisfied (KAOS)

goal-oriented modelling. Given a refined KAOS goal-oriented model of a system, scalability goal-obstacle analysis aims to systematically identify, assess and resolve the potential scalability obstacles of the system. We apply this analysis in the context of microservices to show its potential in systematically analysing the scalability bottlenecks that can occur in contributions which support reasoning about microservice granularity adaptation. Henceforth, this application of the analysis can aid in optimising such contributions to withstand large-scale microservice applications.

1.6 PUBLICATIONS

Publications arising from this thesis are [148, 150, 151],which are respectively the following:

- Sara Hassan and Rami Bahsoon, "Microservices and Their Design Trade-offs: A Self-Adaptive Roadmap", *13th IEEE International Conference on Services Computing (SCC)*, San Francisco, USA, June, 2016, pp. 813-818, cited 30 times as of March 5th 2019.

- Sara Hassan, Nour Ali and Rami Bahsoon, "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity", *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, April, 2017, pp. 1-10, cited 15 times as of March 5th 2019.

- Sara Hassan, Rami Bahsoon and Rick Kazman, "Microservice Transition and its Granularity Problem: A Systematic Mapping Study", 2018, submitted to IEEE Transactions on Software Engineering (TSE).

- Sara Hassan, Rami Bahsoon, Leandro Minku and Nour Ali, "Dynamic Evaluation of Microservice Granularity Adaptation", 2019, in preparation for submission to IEEE Transactions on Software Engineering (TSE).

- Sara Hassan and Rami Bahsoon, "A Value-driven Planning Engine for Runtime Microservice Granularity Adaptation", 2019, in preparation for submission.

- Sara Hassan and Rami Bahsoon, "Microservice-specific Guidance for Scalability Analysis: Scalability-aware Decision Support for Microservice Granularity Adaptation", 2019, submitted to ACM Transactions on the Web (TWEB).
- Sara Hassan, Rami Bahsoon, and Rick Kazman, "Evaluating Granularity Adaptation of Microservice Architectures: How Economics Reasoning Can Help?", 2019, in preparation for submission.

Among the concepts used in this thesis is the notion of Bayesian surprise tolerance; we contribute to this concept in the following publication:

- Sara Hassan, Nelly Bencomo, and Rami Bahsoon, "Minimizing nasty surprises with better informed decision-making in self-adaptive systems", *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Florence, Italy, May, 2016, pp. 134-144.

1.7 THESIS ROADMAP

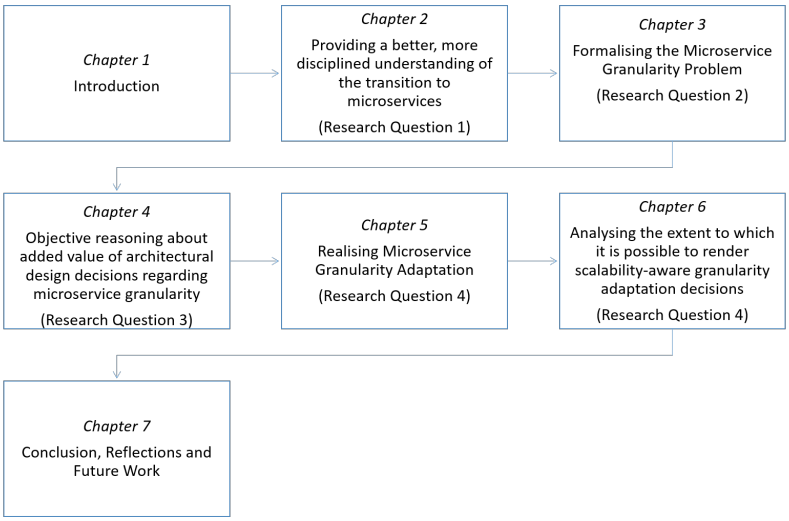


Fig. 1.3 Thesis roadmap;the research question relevant to a chapter is in brackets under that chapter's topic where applicable (e.g., Research Question 1 is relevant to Chapter 2)

To present the contributions from Section 1.5, the rest of the thesis follows the roadmap below (summarised in Figure 1.3):

- *Chapter 2 — Providing a better, more disciplined understanding of the transition to microservices*: In this chapter we present out contribution to address

Research Question 1: a systematic mapping study that consolidates various views, approaches and activities that commonly assist in the transition to microservices. We first present the objectives of the study, a brief overview of microservices and the process followed in the systematic mapping study which is inspired by guidelines from [261, 116, 181, 262]. We then report and analyse the results of the study and present a working definition to characterize the transition to microservices — which we call microservitization. We then infer the research gaps which we address in this thesis from the results of the study. Finally, we briefly discuss similar studies conducted in the field of microservices and contributions from other software engineering fields that can inspire our work. This chapter is derived from [151, 148].

- *Chapter 3 — Formalising the microservice granularity adaptation problem:* In this chapter we present our contribution to Research Question 2 — microservice ambients — an architecture-centric,aspect-oriented modelling concept for microservices. We first formulate the microservice modelling support requirements using a motivating scenario from Filmflix. Then we present a background on ambients [258, 13] and they can aid in meeting these requirements. Next we present our contribution which extends microservice ambients. We use Filmflix to illustrate the how of our contribution in modelling multiple granularity adaptation strategies. Finally, we use ADL classification frameworks to qualitatively evaluate our contribution for expressiveness, effectiveness and facilitating runtime analysis of the microservice granularity adaptation problem. This chapter is derived from [150].

- *Chapter 4 — Objective reasoning about the added value of architectural design decisions regarding microservice granularity:* In this chapter we present our contribution to Research Question 3 — a novel dynamic evaluation process customised for microservice granularity adaptation. We first use Filmflix to highlight the significance of linking microservice granularity adaptation to its predicted added value under uncertainty at runtime. We then explain our novel contribution and how it provides a novel combination of binomial real options

analysis [248, 330, 329] and Bayesian surprises [44] aimed at objectively reasoning about microservice granularity adaptation dynamically at runtime. We compare our process to representative state of the practice microservice runtime monitoring tools to show the extra advantage our contribution provides regarding reasoning about granularity adaptation. We then implement Filmflix using AWS Lambda to illustrate how our novel combination of concepts goes beyond traditional binomial real options analysis in revealing dynamic trends related to the potential added value of granularity adaptation in response to runtime changes in the microservice architecture's environment. This chapter is partially derived from [292] which is under review.

- *Chapter 5 — Realising microservice granularity adaptation:* In this chapter we present our contribution to Research Question 4 — an interactive, iterative, runtime planning engine for microservice granularity adaptation. We first use FIlmflix to motivate the need for a layer of runtime intelligence orchestrating our two other contributions in order to provide effective decision support for reasoning about granularity adaptation. We then present the input, process and possible output cases of our planning engine. We then we use Filmflix to illustrate the usage of the planning engine. Using this illustration we show how our planning engine justifies that the suggested granularity adaptation strategies can introduce more added value in the future. Finally, we discuss the planning engine's effectiveness and efficiency in providing decision support in a dynamic microservice environment.

- *Chapter 6 — Analysing the scalability of decision support for microservice granularity adaptation:* In this chapter we contribute to a provisional catalogue of the microservice-specific scalability dimensions and metrics which need to be considered when reasoning about microservice granularity adaptation at a large scale. Additionally, we apply scalability goal-obstacle analysis [104, 91] in the context of reasoning about microservice granularity adaptation. Given a particular decision support contribution, goal-obstacle analysis systematically analyses, justifies and perhaps extends dimensions in the catalogue. We first

use the planning engine's application to Filmflix from Chapter 4 to motivate the need for this chapter's contributions. Then we report on our digestion of the state-of-the-practice literature (from Chapter 2) which we use to compile our microservice-specific catalogue. Then we provide an overview of KOAS goal-oriented modelling and scalability goal obstacle analysis. We evaluate and discuss our contributions using ad-hoc and systematic scalability assessment of our planning engine and the initial Filmflix architecture in Section 1.1. This aims to highlight the potential comprehensiveness of our catalogue and how scalability goal-obstacle analysis can aid reasoning about microservice granularity adaptation at a large-scale. We further discuss how this analysis can be applied to other contributions which support reasoning about microservice granularity adaptation and other microservice applications Finally, we conclude by summarising the chapter contents.

- *Chapter 7 — Conclusion, reflections and future work:* In this chapter we reflect on our research questions and contributions. We summarise the contributions of the thesis mapping them to the research questions they address from Section 1.3. We reflect on each contribution and its evaluation using different qualitative criteria (e.g., flexibility, computational overhead, and practical deployment). We suggest research directions that can be addressed to enrich each contribution in the future.

Chapter 2

Microservice Transition and its Granularity Problem: A Systematic Mapping Study

In this chapter, we contribute a systematic mapping study that consolidates various views, approaches and activities that commonly assist in the transition to microservices. The study aims to provide a better understanding of the transition (thereby addressing Research Question 1 — how can the state-of-the-art and practice in microservices be probed to advance the understanding of microservices and whether the transition to them has an impact on development tools, deployment management and/or personnel organisation etc.). It also presents a working definition of the transition and technical activities underlying it. We term the transition and technical activities leading to microservice architectures as microservitization. We then review state-of-the-art and -practice related to reasoning about microservice granularity. This includes reviewing modelling approaches, aspects considered, guidelines and processes used to reason about microservice granularity. This study identifies rooms for improvement related to reasoning about microservice granularity; we aim to address these in the remainder of the thesis.

2.1 INTRODUCTION

The transition to microservices has not been purely driven by technical objectives; the transition requires careful alignment of the technical design decisions with the business ones. The ultimate objective of this alignment is to enhance utilities of the application's software architecture and to improve its value potentials. For example, among the technical design decisions is isolating business functionalities into microservices that interact through standardised interfaces. Isolation motivated only by technical objectives can lead to aggressive decomposition of functionalities favouring service autonomy without considering its impact on value potentials. However, isolation motivated by technical and business objectives can be more informed. It can enhance utilities such as autonomy, replaceability and decentralised governance (among other utilities) to improve the microservice architecture's ability in coping with operation, maintenance and evolution uncertainties. Ultimately, this can also relate to improved maintenance costs and cost-effective quality of service (QoS) provision to end users; these are examples of improved value potentials in the architecture.

Microservices are a more fine-grained and autonomic form of services [148]; the difference between service-oriented and microservice architectures is still an open research area. At a high level, the microservices architectural style has its core aligning an industry's business objectives with technical ones thereby introducing added value to the architecture. Underlying this alignment is promoting more service autonomy and decentralised governance in the architecture than that called for in service-oriented architectures. Due to the recency of microservices, they have a multitude of definitions; each definition captures different properties of microservices. Definitions mostly agree that the fundamental properties of microservices include enabling facilitated improvement of component characteristics — autonomy, replaceability, independent deployability — and of architectural characteristics — improved reliability, scalability, resilience to failure, availability, and evolvability [130, 190, 297, 148, 129, 34, 159, 201, 232, 231, 192, 298]. In essence, these definitions capture some drivers of the transition to microservices aimed at enhancing utility in the application's software architecture. The utility enhanced through the transition can render benefits that

can cross-cut architectural design, testing, maintenance and service management [90]. For example, microservice autonomy allows the architects to easily locate, implement and test necessary service amendments [148]. Microservice replaceability allows architects to confidently and independently add and/or manage new business functionalities over the system lifetime [148].

The transition to microservices can help the application in better meeting its quality of services (QoS) requirements; this may consequently result into improved compliance with service level agreements for QoS. Better QoS provision can lead to retaining end users for a longer time and/or entering new markets leading to potential economics gains. Since business objectives of microservice adopters ultimately translate into economic terms, obtaining economic gains through microservice adoption can mean this exercise helps in alignment of business objectives with architectural design decisions [148]. Because of their "micro" character, microservices can be mobilised to the benefit of several service-oriented applications that can require "lighter weight" processing (e.g., mobile services and Internet-of-Things (IoT)) [148].

Despite the industrial push towards microservices, there is no disciplined understanding of their transition nor consensus on the principles and activities underlying the transition [142]. A disciplined understanding of the transition is of paramount importance to inform and/or to justify its technical activities by aligning them with their added value and cost. Currently however, the state-of-the-practice in microservice adoption lacks appropriate methods and techniques that can justify value added of technical design decisions. For example, the software architect can be equipped with mechanisms and tools that can enhance replaceability by standardising the communication paradigms across microservices [239, 353]. Reasoning about the added value and possible cost becomes essential to justify this technical design decision regarding communication paradigms.

This chapter is an attempt for a better understanding of the transition to microservices. It contributes a systematic study to consolidate various views about microservices; it then uses the study results to present a working definition describing the transition and technical activities of the transition to microservice architectures. In

this chapter we introduce microservitization — a term for the transition and technical activities leading to microservice architectures.

This working definition has explicitly considered a fundamental problem of microservitization: reasoning about the granularity of a microservice (i.e. whether a microservice should be decomposed/merged further or not). A granularity level determines "the service size and the scope of functionality a service exposes [194, p.426]." Granularity adaptation entails merging or decomposing microservices thereby moving to a finer or more coarse grained granularity level.

A systematic mapping study allows the evidence in a domain to be plotted at an abstract level. "This allows for the identification of evidence clusters and evidence deserts to direct the focus of future systematic reviews and to identify areas for more primary studies to be conducted [181, p.5]." Directing the focus of future systematic reviews aligns with our aforementioned objectives. Ultimately, our attempt at defining the transition to microservices will pave the way for future development and research related to microservice transition. Furthermore, understanding the problem of reasoning about microservice granularity allows identifying areas for future primary studies. Since the examined literature regarding microservices spanned a broad variety of aspects, we found a systematic mapping study to be suitable given the amount of reviewed literature [181].

In Section 2.2 we describe the steps we followed in the systematic mapping study. In Section 2.3 we report and briefly analyse our mapping study results. In Section 2.4 we use this analysis to 1) present our working definition for the transition to microservices (Section 2.4.1) and, 2) identify gaps in the state-of-the-art and -practice related to reasoning about microservice granularity (Section 2.4.2). Determining the granularity level too early in the software architecture's lifetime can lead to losing rather than introducing added value to the microservice architecture [174, 17]; this can in turn materialise into economic losses for the microservice adopter. This problem is of significance both in brownfield and greenfield development [93]. In both fields, a suitable granularity level is paramount to inform choosing concrete services from a plethora of off-the-shelf microservices. Overall, the identified gaps motivate the need for:

- Microservice-specific modelling support potentially using an architecture definition language (ADL) that "treats microservice boundaries as adaptable first-class entities [150, p.2]" thereby facilitating runtime analysis of microservice granularity in a systematic architecture-oriented manner [150].
- A dynamic architectural evaluation process that captures two dimensions under uncertainty at runtime: added value to be introduced and cost to be incurred if the granularity of microservice architectures is adapted.
- Effective, scalability-aware decision support that should systematically guide the software architects towards suitable granularity adaptation strategies at runtime or suggest re-visiting their expectations of the architecture's runtime environment.

In Section 2.6 we compare and contrast related literature reviews, studies and surveys in the field of microservices against our systematic mapping study. In Section 2.5 we reflect on threats to the validity of our study. In Section 2.7 we summarise contributions that are not directly linked to microservices but can be relevant to their emergence and development. In Section 2.8 we conclude by summarising the results of our systematic mapping study.

2.2 SYSTEMATIC MAPPING STUDY PROCESS

The process we follow in our mapping study is inspired by guidelines from [181]. In Section 2.2.1, we reify the objectives of our study into more concrete study research questions. Subsequently, this section describes how we use them in each stage of the mapping study we follow in this chapter.

2.2.1 Research Questions

Overall, this chapter conducts a systematic mapping study to address the following objectives:

- *Objective 1*: providing a better understanding of the transition to microservices — we consolidate various views (industrial, research/academic) of the principles,

methods and techniques that are commonly adopted to assist the transition to microservices. This consolidation allows us to reach a working definition for the transition to microservices; we term this transition *microservitization*.

- *Objective 2*: understanding a fundamental problem of the transition to microservices related to reasoning about their granularity — we review state-of-the-art and -practice related to reasoning about microservice granularity. This review allows us to understand the state-of-the-art and -practice in the modelling approaches, aspects considered, guidelines and processes used to reason about microservice granularity.

Given these objectives, we reify them into the following study research questions. Along with each question we outline the rationale behind it.

Objective 1: providing a better understanding of the transition to microservices

- **SRQ1**: What are the activities undertaken to adopt microservices? This question helps understand the principles, methods and techniques of the transition to microservices by digesting experiences of microservice adopters in industry and in academic research.

Objective 2: understanding a fundamental problem of the transition to microservices related to reasoning about their granularity

- **SRQ2**: What are the modelling approaches used to define the granularity of a microservice? The aim of this question is to identify the support provided by microservice-specific models for reasoning about granularity and to investigate how systematic (i.e. standardised and methodological) these models are.

- **SRQ3**: Which quality attributes are considered when reasoning about microservice granularity and how are they captured? The aim of this question is to elicit the possible trade-offs which software architects need to balance when reasoning about microservice granularity and/or how these trade-offs can be captured objectively.

- **SRQ4**: How is reasoning about microservice granularity described? The aim of this question is to explore the state of the art regarding triggers and steps

of microservice granularity adaptation and their suitability to the dynamic microservice environment.

2.2.2 Search Strategy

The terms used when searching for English publications were "microservice" and "micro-service"; Google Scholar, ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink and Wiley InterScience were used during this search. Our scope of publications includes journals, theses, books, conferences, workshops, blog articles, presentations, and videos. For academic publications which surveyed further relevant literature, snowballing was applied [263] to further extract relevant literature. Non-academic publications were included since most industrial experiences regarding microservices were published in these forms. We made our best effort however to only include articles that either transcribe the view of adopters or ones where the author is the opinion holder. We believe answering the research questions above comprehensively requires examining both academic and non-academic experiences with microservices. This broad scope also aligns with a property of systematic mapping studies — aiming for broad coverage rather than narrow focus [181]. Our search was restricted to publications between 2013 and 2018, since the microservice trend had not emerged prior to that; non-academic literature started to appear in 2013 [337] while peer-reviewed publications started to appear in 2014 [134]. Meta-data of the search results was maintained using a tool called "Publish or Perish" [259].

2.2.3 Selection of Primary Studies

Initially, the search results were examined for relevance according to inclusion and exclusion criteria below. Each study research question elicited in Section 2.2.1 has corresponding inclusion/exclusion criteria (their structure is inspired by [262]). Along with each criterion is the rationale justifying it italicised.

What activities are undertaken to adopt microservices?

Inclusion Criteria:

- Publications generically presenting the challenges of adopting microservices *since they can be used to infer the activities comprising the transition to microservices*.
- Case studies of adopting microservices *are used to complement and verify the activities in generic publications.*
- Publications comparing specific development tools in the microservice industry *because they can be used to infer activities comprising the transition.*

Exclusion Criteria:

- Publications without any reference to microservices. *Including such publications would confuse rather than clarify understanding the transition to microservices. This is against Objective 1 of our study.*
- Publications that refer to servitization in the business not the software context *since we are only concerned about the activities of shifting a software system from another architectural style to microservices.*

What modelling approaches are used to define the granularity of a microservice?

Inclusion Criteria:

- Publications defining formal notations/diagrams for modelling microservices. *Such publications can be used to assess how systematic the state-of-the-art is for modelling microservice granularity.*
- Proposals of modelling concepts for microservices. *Even when unverified, a proposed modelling concept can provide an insight for the building units of reasoning about microservice granularity.*
- Publications presenting industrial case studies for modelling microservices. *Such publications would verify and illustrate the expressiveness of proposed modelling concepts to microservice granularity.*

Exclusion Criteria:

- chapters which provide binding and re-configuration solutions for SOAs/web-services/mobile services only *are excluded since they do not capture the properties specific to microservices, hence they are not suited for reasoning about microservice-specific decision problems (in this case microservice granularity).*
- chapters which provide modelling approaches for SOAs/web-services/mobile services are *excluded since they do not capture properties specific to microservices, hence they are not suited for reasoning about microservice-specific decision problems (in this case microservice granularity).*

Which quality attributes are considered when reasoning about microservice granularity and how are they captured?

Inclusion Criteria:

- Publications presenting metrics used when reasoning about microservice granularity. *Such publications would help assess how objectively the trade-offs affecting microservice granularity are captured in academia and/or industry.*
- Case studies involving the quality drivers considered when reasoning about granularity. *Such publications would realistically capture the significance of specific trade-offs when reasoning about granularity adaptation.*
- Publications focused on vendor-specific comparisons between platforms supporting reasoning about microservice granularity. *This can help derive the quality attributes and metrics considered when reasoning about granularity.*

Exclusion Criteria:

- Case studies that do not explicitly relate a challenge to its impact on microservice granularity. *Since case studies report concrete challenges and trade-offs impacting them, it is unreasonable to claim an impact of a reported trade-off on granularity if that is not reported explicitly in a case study.*

How is reasoning about microservice granularity described?

Inclusion Criteria:

- Publications including guidelines for reasoning about microservice granularity. *This helps to identify the state-of-the-art regarding triggers and/or steps for granularity adaptation.*
- Publications showing a sequence of when and how granularity is reasoned about in the application's lifecycle. *These publications help assess how much the state-of-the-practice considers dynamicity in microservice environments when reasoning about granularity adaptation.*

Exclusion Criteria:

- Publications that provide generic best practices for the granularity of applications with no reference to microservices (e.g. related to web services, SOA, mobile services). *Such best practices are not targeted specifically at microservices, so it would not be reasonable to use them in the context of microservice granularity.*

2.2.4 Keywords and Classification

"The purpose of this stage is to classify chapters with sufficient detail to answer the broad research questions and identify chapters for later reviews without being a time consuming task [181, p.44]." In Appendix A we classify the included publications according to two classification frameworks. The first framework classifies our publications according to their research approach, where the categories are elicited from [360]. The second framework classifies our publications according to whether or not they include keywords related to our mapping study objectives; we elicit and refine these keywords from a microservice-specific mapping study [17].

2.2.5 Data Synthesis

RSS feeds and manual search were used to obtain publications complying with the strategy defined in Section 2.2.2. The results were then manually examined for

inclusion and categorised according to the criteria and frameworks described above (Sections 2.2.3 and 2.2.4 respectively).

2.3 RESULT REPORTING AND ANALYSIS

In this section we present graphs summarising distributions of the included publications along the categories described in Table A.1. For each graph, we discuss how it helps serve the objectives outlined in Section 2.1.

2.3.1 Publication Distribution Overview

A total of 608 publications met the inclusion criteria in Section 2.2.3. Table 2.1 lists representative examples of the included publications categorised according to Table A.1. Figure 2.1 shows the overall distribution of the publications according to the publication type.

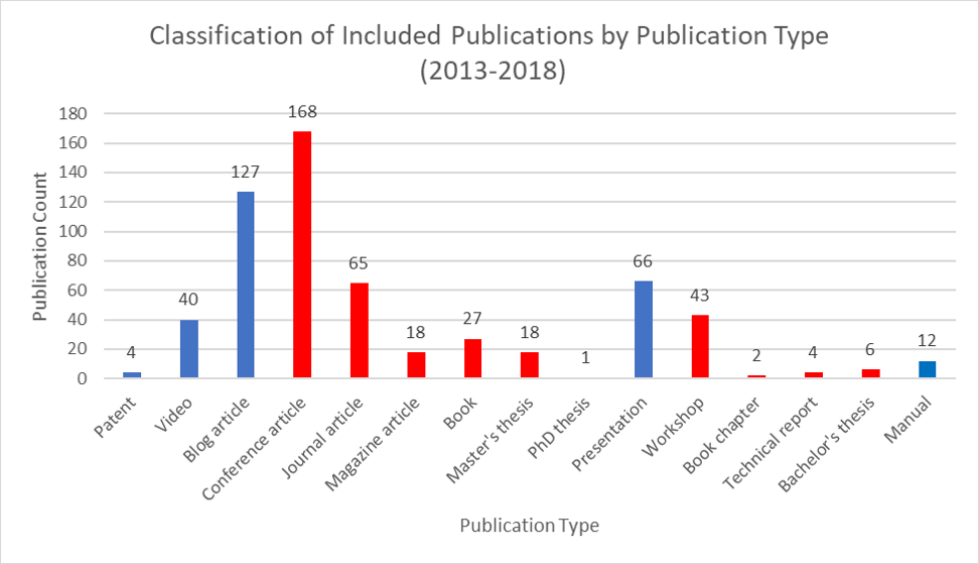


Fig. 2.1 Publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 included as per criteria from Section 2.2.3 classified according to their publication type; the red bars are the publications we consider as academic and the blue bars are those we consider non-academic

As justified in Section 2.2, we widened the scope of our study to include both academic and industrial publications. Broadly, we consider a publication type to be academic if it has gone through editing or peer-revision (the red bars in Figure 2.1); about 62% of the included publications. On the other hand, non-academic publications account for about 38% of the total. Although the majority of publications

are academic, non-academic literature still comprises a significant percentage. Had we excluded these publications, attempting a definition for microservitization (Objective 1 in Section 2.1) would have been biased. The exclusion of non-academic sources would lead to missing relevant keywords related to each category and the coverage of industrial opinions and experiences related to microservice adoption would be narrower.

Study Research Question	Category	Representative Examples
What activities are undertaken to adopt microservices?	Architectural design	[229, 191, 153, 154, 216, 363, 249, 238, 291, 285, 114, 3]
	Organisation	[372, 180, 254, 35, 277, 240, 364, 101, 361, 25]
	Operation	[47, 118, 113, 59, 293, 86, 336, 245, 74, 121, 347, 35]
	Deployment	[179, 220, 370, 53, 115, 72, 362, 279, 335, 368]
	Development	[264, 244, 311, 272, 222, 332, 111, 144, 322, 21, 24]
	Monitoring	[160, 307, 141, 122, 57]
	Logging	[299]
What modelling approaches are used to define the granularity of a microservice?	Structural	[112, 168, 319, 339, 270, 234, 128, 173, 125]
	Behavioural	[276, 131, 142, 38]
Which quality attributes are considered when reasoning about microservice granularity and how are they captured?	Performance	[235, 164, 193, 56, 315]
	Reliability	[95, 271, 54]
	Scalability	[176, 202, 152, 184, 14]
	Maintainability	[243, 4]
	Complexity	[223, 340, 83, 183]
How is reasoning about microservice granularity described?	Guidelines	[58, 348, 273, 317, 275, 302, 200, 46, 327]
	Processes	[126, 125, 278, 256, 265, 266, 316, 162]

Table 2.1 Representative examples of publications included in the systematic mapping study

We further classify peer-reviewed publications according to a highly-cited chapter classification framework [360] which targets IEEE publications. This framework classifies publications according to their research approach; a brief description of each approach is presented in Section 2.2. This framework has been applied before in the

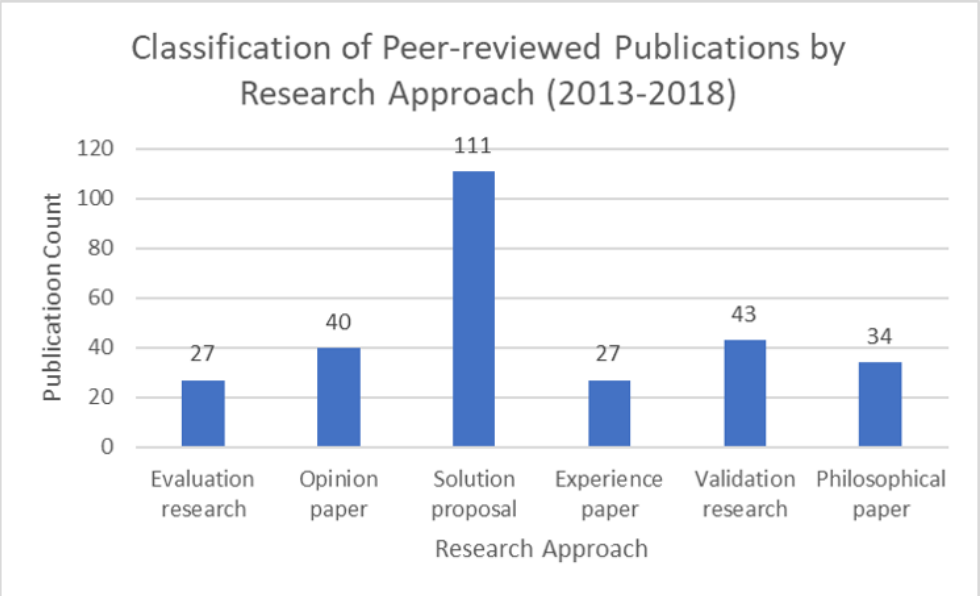


Fig. 2.2 Academic (i.e. peer-reviewed) included publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 classified according to their research approach; the classification criteria are derived from [360];

context of microservices [17], so we consider it a neat fit for our study. To match the target context of the framework, we only apply it to peer-reviewed publications (Figure 2.2).

Solution proposals by far comprise the largest number of peer-reviewed publications. Solution proposals present novel, significant techniques without a full-blown validation. A proof-of-concept may be offered in solution proposals by means of a small example, a sound argument, or by some other means [360]. Therefore, the microservices trend is a thriving field for novelty but it is still lacking maturity. Validation research publications — which thoroughly investigate solution proposals [360] — only amounted to 43 publications, which further proves the lack of maturity in the field. The large difference between the solutions proposals and validation research publications proves the need for disciplining the transition to microservices. The following subsections further discuss this need then focus on one of the fundamental problems of the transition — reasoning about microservice granularity.

2.3.2 Objective 1: Providing a Better Understanding of the Transition to Microservices

Figure 2.3 classifies the publications which include keywords related to: *What are the activities undertaken to adopt microservices?* Architectural design and managing deployment are the most popular activities undertaken when adopting microservices. Therefore, we infer they are crucial activities in the transition to microservices. Nevertheless, there is a significant number of publications in the other categories of Figure 2.3. The variation in number of publications across categories of Figure 2.3 implies there is no consensus in describing the transition to microservices. This leaves room for us to introduce the microservitization term which attempts to provide a better understanding of the transition to microservices. Even though it is common and acceptable in academic research for publication to focus only on activities of interest related to microservice adoption, we argue that such publications should still show appreciation of all the activities relevant to the exercise. However, we did not find this appreciation and awareness in the publication summarised in Figure 2.3.

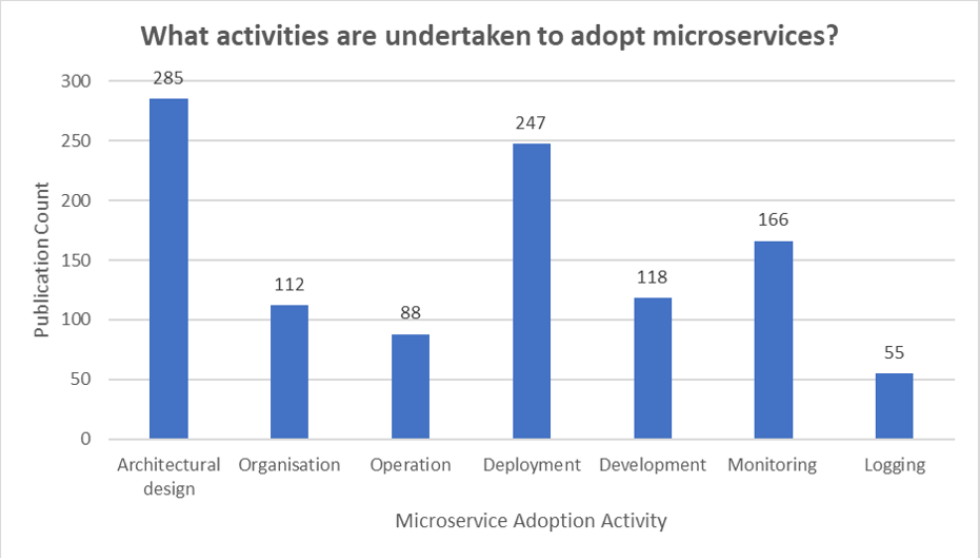


Fig. 2.3 Included publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 that have keywords related to SRQ1

2.3.3 Objective 2: Understanding the Microservice Granularity Problem

One of the fundamental problems of the transition to microservices is finalising their level of granularity [334, 123, 154, 216, 238, 93]. Architecture definition language (ADL) classification frameworks [213] indicate that structural (or "topological [213, p.26]") as well behavioural aspects of an architecture need to be modelled. Figure 2.4 helps to clearly identify the state-of-the-practice in modelling microservices; to answer *what are the modelling approaches used to define the granularity of a microservice?* The structural modelling approaches proposed for microservices are almost double the behavioural approaches. Structural approaches capture the topology and/or dependencies across building units of the microservice architecture. On the other hand, behavioural approaches capture the actions of these units in several runtime scenarios in which the modelled microservice operates.

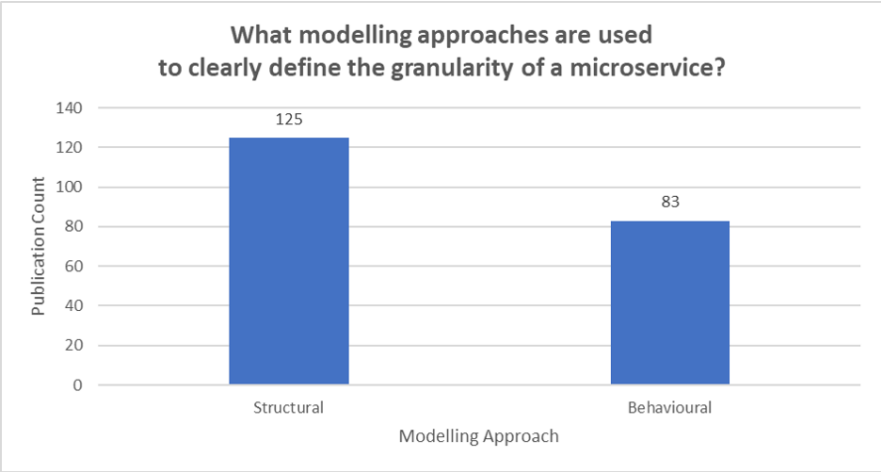


Fig. 2.4 Included publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 which have keywords related to SQR2

Therefore, a systematic, architecture-oriented modelling approach for microservice architectures which facilitates reasoning about granularity needs to capture the architecture's structural and behavioural aspects. The difference in numbers between structural- and behavioural-oriented publications in Figure 2.4 indicates that there is a lack in modelling approaches that capture both aspects of a microservice architecture.

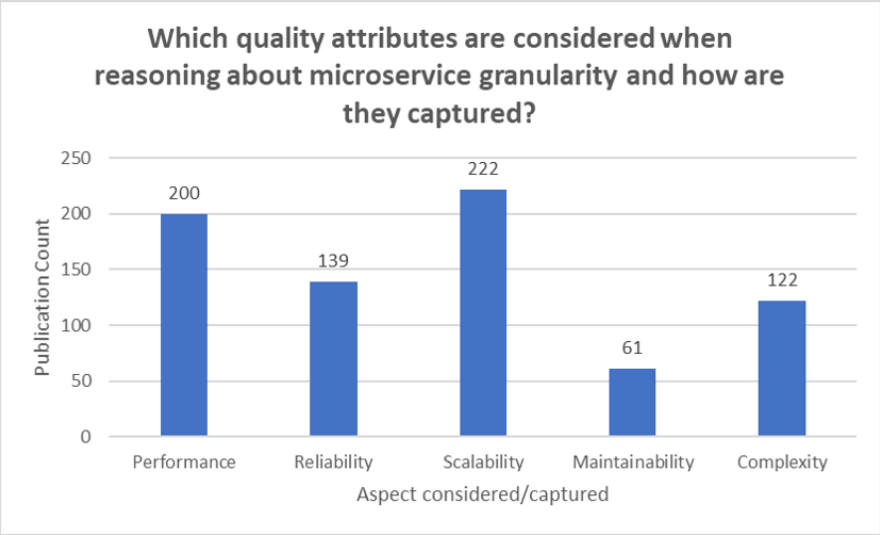


Fig. 2.5 Included publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 that have keywords related to SQR3

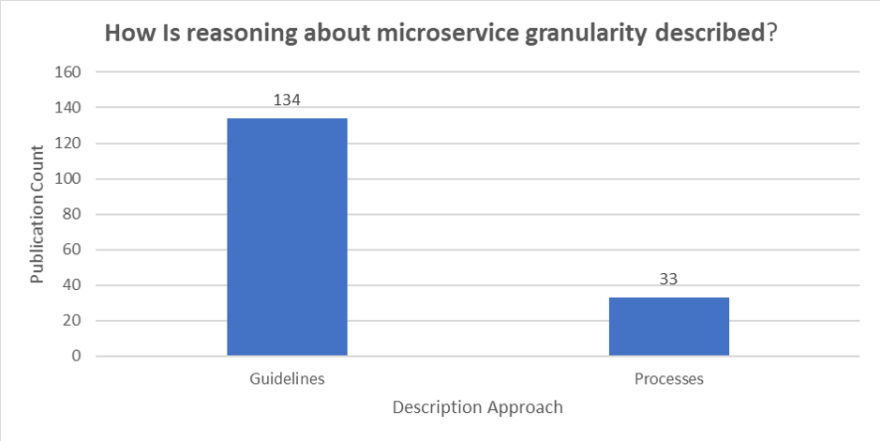


Fig. 2.6 Included publications between 2013 and 2018 as per the search strategy defined in Section 2.2.2 that have keywords related to SQR4

Figure 2.5 classifies publications according to the quality attributes they aim to optimise when reasoning about microservice granularity; to answer *which quality attributes are considered when reasoning about microservice granularity and how are they captured?* In other words, they can be the most common means to introduce utility through cost-effective microservitization. Scalability is the most common quality considered in the examined literature. This is reasonable given the dynamic, large-scale environment in which microservices operate [145, 353, 159, 63, 273, 142, 55, 306, 82, 324]. Therefore, we infer that scalability can be among the attributes of highest concern when it comes to introducing added value to microservice architec-

tures. Relatively few publications have considered complexity/cost when reasoning about microservice granularity. Therefore, there is room for contributing to dynamic decision support which objectively considers both the potential value and cost of decisions related to microservice granularity (i.e. adapting granularity by decomposing or merging microservices).

Figure 2.6 classifies proposed approaches for reasoning about microservice granularity according to how they are described; this answers *how is reasoning about microservice granularity described?* Most of the proposed approaches are ad-hoc guidelines that could be applied differently at various points of the microservice application's lifecycle. However, effective decision support for microservice granularity should comprise a clear sequence of distinct steps to be triggered under clear conditions. We have not found such effective support even in the 30 publications proposing a process to reason about microservice granularity. Therefore, we infer that there is a lack in effective decision support for reasoning about microservice granularity in terms of clear adaptation steps and triggers.

2.4 ADDRESSING SYSTEMATIC MAPPING STUDY OBJECTIVES

In Section 2.4.1 we present a working definition for the transition to microservices; we call this transition microservitization. In Section 2.4.2, we identify the gaps in state-of-the-art and -practice related to reasoning about microservice granularity (inferred from Section 2.3) and discuss how they can be addressed.

2.4.1 Objective 1: Providing a Better Understanding of the Transition to Microservices

We have not found in the surveyed publications an empirically grounded definition which characterises the transition to microservices, but rather several conflicting, informal attempts coming from industry and academia. Table 2.2 analyses these attempts in terms of whether or not they explicitly include the activities derived from the microservice-specific systematic mapping study [17] and described in Section 2.2.4. It is worth noting that the attempts analysed in this table are just a fraction of the publications summarised in Figure 2.3. In Table 2.2 we focus on the explicit attempts to define the transition to microservices, while Figure 2.3 includes both

explicit attempts to define the transition and case studies of microservice adoption which do not explicitly attempt to define the transition.

Publication	Microservice Adoption Activity						
	Architectural design	Organisation	Operation	Deployment	Development	Monitoring	Logging
[334]	✓	✓		✓			
[201]				✓			
[269]	✓	✓	✓				
[366]	✓					✓	✓
[123]	✓	✓		✓		✓	✓
[187]				✓			
[102]					✓		
[298]	✓	✓				✓	
[228]	✓	✓			✓		
[34]	✓		✓	✓	✓	✓	✓
[156]	✓						
[134]	✓				✓	✓	
[174]		✓		✓			
[171]	✓		✓		✓		
[16]				✓	✓		
[327]	✓						

Table 2.2 Analysing publications that attempt to define the transition to microservices included in the systematic mapping study; a check means the publication includes the activity in the corresponding column

Observing Table 2.2, we introduce a definition for the transition to microservices (adapted from the activities included in [17]) which we call microservitization to cover all the relevant activities of the transition to microservices. This definition is adapted from activities included in [17] and inspired by a trending concept in business and manufacturing domains [219] — servitization.

In the manufacturing and business domains, servitization is seen as a paradigm shift entailing “manufacturers growing their revenues and profits through services [309]" rather than tangible functional products. A service in this context is any feature that helps the business to 1) “really make money" and, 2) deliver new outcomes to customers. Examples of services in servitization include software applications, customer support, and self-service capabilities [346].

Servitization “embraces business model innovation, organisational change, and new technology adoption. Services exist in various forms, and represent differing values to both the customer and provider [309]". To benefit from these values, servitization involves developing new relationships with customers, innovating customer value propositions, forming new value chain relationships, and adapting business models [309, 318, 33, 89]. The key to successful servitization is “choosing the right technology, picking the appropriate moment to invest, and ensuring successful implementation [32]" which aligns with the business objectives [346].

We liken the transition to microservices to servitization because of the following resemblances:

- Servitization is driven by delivering new outcomes to customers which can translate into revenues and profits. Similarly, the transition to microservices is driven by improving QoS provision to end users and translating that into an economic gain.

- Servitization entails embracing innovation in the manufacturing activities (e.g., building business models and defining customer value propositions) are carried out. Similarly, the transition to microservices entails a dramatic change (motivated by value creation) to the way technical activities manifested in the software architecture are carried out.

Therefore, we define *microservitization as a form of servitization where "services/components are transformed into microservices — a more fine-grained and autonomic form of services [150, p.1]" — to introduce added value to the architecture [148]. "Microservitization is also an example of a paradigm shift [150, p.1]" since it involves dramatically changing how the following technical activities are carried out to align them with a microservice adopter’s business objectives:*

- *Architectural design*: Microservitization introduces the following critical architectural design activities:

  - Choosing light-weight communication mechanisms: Microservitization can increase the distribution of functionalities across the architecture

thereby introducing extra communication calls between microservices [142]. Therefore, rather than relying on enterprise service buses (which are the state-of-the-practice in SOA architectures), more light-weight mechanisms such as pipes and filters, event-based queues, and correlation identifiers are critical to avoid very high communication costs in microservice architectures. The large number of microservices can lead to a large volume of message exchange and hence high communication costs.

  - Reasoning about microservice granularity levels: A suitable granularity level is paramount to inform buying commercial-off-the-shelf (COTS) concrete services or developing them in-house [148]. Choosing these services correctly is critical to introducing added value to the microservice architecture.
  - Adopting fault tolerance design patterns: Although striving for fault tolerance is a best practice in any architecture, investing in fault tolerance design patterns is all the more critical to microservitization. The criticality is due to the scale of industries adopting microservices (e.g., retail [333], entertainment [336, 145]) where microservices span different continents with a wide variety of end users. Microservice-specific fault tolerance design patterns include circuit breakers and bulkheads.
  - Incorporating microservice registration and discovery mechanisms: Microservices are typically developed, deployed and replaced at a very quick rate [142]. Therefore, it is critical to incorporate robust registry and discovery mechanisms in microservice architectures to ensure an up to date record of the currently “alive" microservices.

- *Managing the organisational hierarchy*: Microservitization has a direct impact on the organisational hierarchy [230]. In particular, the autonomy and independent deployability enhanced in the architecture through microservitization facilitate decentralised governance by breaking “silos" (based around strict separation of job roles) in the organisation.

- *Operation*: Microservitization aligns operation management with breaking organisational “silos". DevOps and NoOps are among the state-of-the-practice operation management approaches in microservitization; they involve "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality [40]". Decentralised operation management can reduce the risk of bottlenecks that can materialise into economic losses to the microservice adopter. Reducing this risk is conditional upon development operation teams adhering to service level agreements between them.

- *Deployment*: Microservitization introduces a critical challenge of determining the hosts on which a deployment pipeline is implemented [230]. This is critical to balancing between the added value of microservitization and cost that can be introduced by a deployment pipeline implementation choice (e.g., physical link installation, server rental and maintenance costs). Virtualisation and containerisation are among the common deployment pipeline implementation choices. They can enable swift auto-scaling the microservice architecture in response to changes in its runtime environment; this can materialise into economic gains for competitive microservice adopters.

- *Development*: Unlike other software development paradigms, microservitization enables freedom in choosing development tools which can in turn introduce more added value to the architecture [17]. This freedom is a bi-product of the decentralised governance enabled by microservitization. It is worth noting that communication and knowledge sharing across teams using different development tools needs to be carefully managed to ensure this freedom actually introduces added value.

- *Monitoring*: microservitization requires much more robust, decentralised and customisable monitoring than that of classical SOAs due to the heterogeneity of tools used to develop microservices and scale at which they typically operate [142]. These requirements are critical to cater for the heterogeneity, scale and dynamism of microservice architectures.

- *Logging*: Maintaining logs of the monitoring data needs to be more customisable and distributable than logging classical SOAs due to the heterogeneity of tools used to develop microservices and scale at which they typically operate [142]. This requirements are aligned with the aforementioned monitoring challenges introduced by microservitization.

2.4.2 Objective 2: Understanding the Microservice Granularity Problem

Based on our definition above, microservitization introduces the challenge of reasoning about the suitable granularity level of a microservice.

To formalise the microservice granularity problem, the gap we identified from Figure 2.4 is the lack of an architecture-oriented modelling approach that captures a microservice's granularity behaviour, thereby supporting runtime analysis of this behaviour. The approach should treat microservice boundaries as the primitives for formulating the microservice granularity decision problem and actuators of microservice granularity adaptation decisions. These decisions include merging multiple microservices into a single boundary and decomposing a microservice into multiple ones encapsulated by multiple boundaries. In other words, this approach should treat microservice boundaries as adaptable first-class entities to ensure that both the structural and behavioural aspects of the microservice architecture are captured; we contribute to this approach in Chapter 3. While conducting our study, we have seen architecture-oriented modelling approaches that treat the notion of boundaries statically (e.g., [325]), or provide support for adaptability but without explicitly capturing the notion of adaptable boundaries (e.g., [185]). Because the role of microservices is to encapsulate functionality, it is intuitive to use boundaries as adaptable first-class entities in the modelling approach. By contrast, if the decision problem was to determine the optimal physical infrastructure to host the microservice for example, the adaptable first-class entity would be different (e.g., microservice configuration variables).

To reason about microservice granularity objectively and dynamically, the gap we identified is the lack for an architectural evaluation process that captures two aspects explicitly: benefit to be introduced and cost to be incurred by pursuing granularity

adaptation (or not). Explicit objective reasoning about benefit and cost about this exercise is essential since we view pursuing granularity adaptation as a strategic architectural design decision which can have significant economic gains and/or losses on the architecture. In essence, such objective reasoning can be seen as a means of assessing the risks related to pursuing granularity adaptation (or not).

To effectively support reasoning about granularity adaptation, the gap we identified is the need for a decision support tool for reasoning about this problem at runtime. It is seen as a runtime problem since the suitable granularity level highly depends on the current scenario in which the microservice architecture is operating [148, 351]. For example, if a certain functionality in a microservice-based application is continuously receiving a large volume of requests at runtime, it makes sense to decompose this functionality in a separate microservice to manage its load separately. On the other hand, if two microservices are continuously communicating across a network at runtime causing latency, then merging these microservices is sensible to help reduce such latency. Overall, uncertainties related to the expected environment and behaviour [301, 75] of the microservice architecture can not be fully captured at design time. Therefore, a runtime decision support tool is necessary to track and analyse this uncertainty. The tool should systematically guide the software architects towards suitable granularity adaptation strategies at runtime or suggest re-visiting their expectations of the microservice runtime environment. Each candidate granularity adaptation strategy must be systematically described as a sequence of merging/decomposition steps accompanied by triggers on them. Moreover, the tool's suggestions need to be justified objectively while leaving the final decision to the architects for adopting the suggested strategies; approaches such as [147] can inspire the design of this tool.

2.5 THREATS TO VALIDITY

In this subsection we acknowledge the threats to validity in the process we use in our systematic mapping study as well as the application of each stage. Threats to validity

are "influences that may limit our ability to interpret or draw conclusions from the study's data [261, p.351]".

When defining our search strategy in Section 2.2.2, we considered blog articles, presentations, and videos as the means of reporting first-hand industrial experiences with microservice adoption. We acknowledge that an alternative means could be interviews with microservice adopters in the industry. However, published blog articles, presentations and videos are arguably more trusted since they present a more responsible and objective view than interviews. Though they can enrich the study with diverse opinions, interviews tend to suffer from bias, subjectivity and irresponsible answers [182]. Moreover, our search strategy yields publications that explicitly mention microservices. Nevertheless, we acknowledge that there are publications prior without direct mention of microservices that could be relevant to their emergence and thereby to our research (e.g., web service composition and agile development). We attempt to address this threat briefly in Section 2.7.

We acknowledge that the inclusion and exclusion criteria might have led to missing contributions that can inspire the microservices field. However, since our study was motivated by studying the state-of-the-art and -practice in the microservices trend we based the criteria on publications which already have a link to microservices. Nevertheless, we briefly outline the research areas that can inspire the microservice trend in Section 2.7. On a more technical level, we excluded publications that could not be translated to English which might have affected the study results.

We iteratively built Table A.1 to include all the relevant keywords and made our best effort to justify them (Section 2.2). However, we acknowledge that some keywords might have been missed related to each study research question.

When extracting videos and presentation for inclusion in the study results, we made our best effort to include videos whose content contained keywords from each category. The keywords are either mentioned by the speaker in the video or in the slides presented in the presentation. We acknowledge however that this might have biased the study results and that in the future transcription of the videos/presentations would be a more accurate means of determining their relevance.

When categorising publications according to Table A.1, we made our best effort to considers synonyms of the keywords in each category. However, since we categorised the publications by manual examination, we acknowledge that their distributions might have been skewed. In particular, we acknowledge that subjective interpretation of keywords might need to be complemented with more systematic approaches of categorising the yielded publications. For example, the context in which a keyword or a fragment is mentioned needs to be considered before putting each publication under a certain category. A more systematic categorisation of the publications can ensure the reproducibility of our results.

We acknowledge that skewed distributions might lead to biassed inferences regarding gaps in the literature related to each objective of our study. For example, the variation in numbers of publications across categories in Figure 2.3 might be due to the inadequacies in keywords under each category. It might also indicate interest in architectural design across practitioners (i.e. in non-academic publications); this might not be as accurate for academic publications. Nevertheless, we argue that our mapping study results give a strong insight into the microservice trend and opens directions for more detailed research down each direction. For example, a systematic mapping study can be conducted which focusses specifically on microservice architectural design and/or development — the most dense categories in Figure 2.3.

2.6 RELATED STUDIES

Motivated by disciplining the understanding of microservices, several studies have been conducted to examine the existing literature in this young yet trending field. They analyse existing literature with a variety of focuses. In this section, we compare and contrast the examined studies against the systematic study we conducted.

The closest to our study are [17, 249] since they both adopt a systematic mapping study process when examining the literature. However, the motivations in these studies are different from ours. In [17] for example, the study research questions motivating the study are related to challenges, modelling approaches and quality attributes considered when adopting microservices. These questions do overlap

partially with both objectives of our study but they do not focus on reasoning about microservice granularity as we do in our study. In [140], the activities comprising the transition to microservices are inferred through a literature review (aligned to Objective 1 of our study). However, it does not focus on microservice granularity so our study is significant to understanding this microservitization challenge.

Several systematic literature reviews and surveys focus on defining the fundamental properties of microservices and the challenges of adopting them. They range in their rigour: Some follow a rigorous search protocol [72, 71, 372, 156, 352, 174, 134, 320] while others are less formal [100]. These studies overlap partially with Objective 1 of our study; they present activities related to microservices thereby contributing to a better understanding of the transition. However, they do not define on the transition to microservices nor can they be used to understand the microservice granularity problem. In [228], the transition to microservices is partially described in terms of design patterns that can be applied to microservices. However, the scope of activities comprising the transition is not clearly defined.

Some studies mainly focus on modelling microservices [18, 70]. Their focus partially overlaps with our following research question: *What are the modelling approaches used to define the granularity of a microservice?*

On the other hand, [198] aims to "construct knowledge of quality attributes in architecture through a Systematic Literature Review (SLR), (an) exploratory case study and (an) explanatory survey [198, p.1]." In essence, this study can be used to partially address our question: *What are the quality attributes considered when reasoning about microservice granularity and how are they captured?* Nevertheless, the other study research questions were not answered in this study.

Overall, the examined studies can complement this chapter to discipline the understanding of the transition to microservices. In this chapter, our study research questions are formulated with focus on a specific problem of this transition — reasoning about microservice granularity.

2.7 RELEVANT RESEARCH FIELDS

Although we focus on publications that have direct links to microservices in our study, we acknowledge that there are publications prior to the time period considered in this mapping study that could be relevant to the emergence of microservices and thereby to our research. Such publications do not fit our search strategy because they do not make direct reference to microservices. Nevertheless, in this section we briefly summarise the examined literature in these areas indicating how they can be relevant to our systematic mapping study objectives.

2.7.1 Research Fields Relevant to Understanding the Microservice Transition

Even among microservice adopters, there are still debates over distinguishing service-oriented architectures (SOA) from microservice architectures [228, 16, 67, 328, 130, 291, 326, 238]. Therefore, we infer that contributions defining the properties and challenges of adopting SOA architectures can be relevant to understanding the transition to microservices.

Seminal work defines SOA as an architectural style which can guide business process definition [110, 109, 206, 288, 260, 374] and support "rapid, low-cost composition of distributed applications [252, 251, 253, p.1]." The SOA style was introduced to address architectural complexity, redundant programming and inconsistent interfaces [73, 48]. In SOAs a service is a self-describing unit that "consists of a contract, one or more interfaces and an implementation [189, p.57]." SOAs in turn comprise an application frontend, services, a service repository and enterprise service bus.

Despite the resemblance between microservices and SOAs, there is a subtle distinction we infer from our microservitization definition. The distinction comes from [148]: 1) the potential of microservices as autonomous fine-grained computational units with lightweight communication mechanisms rather than service buses and, 2) the operational and organisational flexibility enhanced by microservitization. Further elaboration of these points is presented in [274]. Microservices are a more fine-grained and autonomic form of services [148]; the difference between service-oriented

and microservice architectures is still an open research area. At a high level, the microservices architectural style has its core aligning an industry's business objectives with technical ones thereby introducing added value to the architecture. Underlying this alignment is promoting more service autonomy and decentralised governance in the architecture than that called for in service-oriented architectures.

2.7.2 Research Fields Relevant to Understanding the Microservice Granularity Problem

Modelling microservice granularity can be inspired by architectural modelling approaches — a wide research field, where contributions capture different notions of the architecture at varying levels of abstraction [31]. Domain-driven modelling is the most relevant to the modelling approach we describe in Section 2.4.2 because it strives for logical isolation of business functionalities [112, 233]. However, domain-driven modelling is more concerned about the relationships between functional boundaries rather than their scope. In essence, domain-driven modelling can inform the structural rather than behavioural aspect of modelling microservices.

The Zachman framework [367] provides a comprehensive guide to the different dimensions and perspectives for architectural modelling. Two dimensions in this framework are aligned with the modelling approach we call in Section 2.4.2: *What* the model units are and *where* these units are located relative to each other. We call for a modelling approach that explicitly captures *what* each microservice is concerned about and helps define *where* the business functionalities encapsulated by the microservice are located relative to each other.

A more dynamic architectural modelling approach is feature modelling [359], where an architecture is defined as a set of variability points, the candidates for each variability point and the rules constricting the dependencies across variability points. We appreciate this modelling technique is useful for formulating architectural decision-making problems. However, we would need to leverage such concept to focus on microservice boundaries being the variability point. While dependency rules in a feature model can give an insight about microservice granularity, they do not explicitly model it. EUREMA [349] provides a yet more powerful dynamic modelling

technique. Here an adaptation engine contains a runtime model which represents the evolution of the system as well as adaptation activities to be executed on that model. The link between the adaptation activities and the runtime model is expressed using runtime mega-models. Similar to feature modelling, EUREMA needs to capture the notion of microservice boundaries more explicitly.

Boundaries are modelled more explicitly in design structure matrices (DSMs) [87]. Modularity metrics [217] can be used to assess the degree of interdependence across these boundaries. We have not seen a dynamic application of DSMs that captures the changes in these dependencies across a time unit (e.g., release cycles).

Since we call for objective reasoning about microservice granularity adaptation, design metrics can inspire this requirement since they provide an objective way to capturing attributes of a design decision. Effort-based metrics [314] evaluate software development and maintenance efforts when a transition is made from centralised to distributed system architectures [287]. By analogy, an objective way is needed to evaluate development and maintenance costs when microservice granularity is adapted. Metrics related to cohesion, coupling and visibility of system components are presented and visualised in [287, 52], which can be used to assess the impact of granularity adaptation on the microservice architecture's modularity.

Since reasoning about microservice granularity is in essence a dynamic architectural design decision problem, several software engineering fields can be relevant to it. Architectural analysis methods, architectural design patterns, service composition and orchestration approaches, runtime architectural adaptation, architectural refactoring and feedback control loops are only some of the relevant fields. In the following subsections we categorise contributions in these fields according to their level of autonomy:

- *Manual* contributions with full reliance on the software architect and/or stakeholders (e.g., architectural design patterns and/or anti-patterns)
- *Partially autonomous* contributions where there is an autonomous agent but the software architect still makes the final decision regarding the optimal architecture (e.g., interactive service orchestration and/or composition)

- *Fully autonomous* contributions where the decision-making process and executing the decision are fully handled by an autonomous agent (e.g., feedback control loops, online architectural refactoring)

2.7.2.1 Manual Contributions

Out of the approaches in this subsection, the most cost- and value-aware approaches we examined are [248, 329, 330]. Refactoring the architecture according to patterns [248] or to introduce modularity [330] are regarded as value-bearing investments [329]. However, these approaches are only applied statically at design time. In this chapter, we call for a similar view for reasoning about microservice granularity.

In [28], a cost benefit analysis method (CBAM) is proposed as a generic architecture evaluation method which utilises techniques in decision analysis, optimisation, and statistics to evaluate architectural design decisions. However, CBAM does not dynamically track and update the added value of architectural decisions. These dynamic updates are critical to the nature of the microservice granularity problem.

In [280], the net benefit of a software is calculated by deducting its total costs from the total benefit. These are manually elicited and monetised from software architects through a series of questions (e.g., "what is the status of the environment without the system?"). To our knowledge, this approach however does not consider the uncertainty in the answers to these questions nor does it update them at runtime. In [61] the predictive analysis of design captures the value-driven impact of architectural decisions as multi-dimensional normalised, weighted cost and value vectors. Therefore, it can only provide static objective decision support for reasoning about microservice granularity.

The techniques in [98] present different architectural evaluation methods and their motivations. Software Architecture Analysis for Evolution and Reusability (SAAMER), Scenario-Based Architecture Re-engineering (SBAR) and Architecture Level Prediction of Software Maintenance (ALPSM) in particular take an objective approach to architectural evaluation.

SBAR captures the runtime nature of decision-making by providing different quality attribute evaluation techniques depending on whether the quality attribute is

concerned with the "development" of the system (i.e. design time, such as reusability, which is handled by scenario-based evaluation) or the "operation" of the system (i.e. runtime, such as performance, which is handled by simulation-based evaluation). SAAMER on the other hand partially addresses granularity problem by analysing the level of interaction between different scenarios of the system as a means of assessing the level of functionality isolation in the system. Nevertheless, both methods have not been explicitly applied in a dynamic environment to our knowledge.

ALPSM takes a more value-driven approach to the evaluation, similar to the Cost Benefit Analysis Method (CBAM) [28], which makes them more systematic architectural evaluation approaches than SAAMER and SBAR above. Furthermore, ALPSM uses probabilities to capture the likelihood of the impact of scenarios and CBAM captures the uncertainty the architectural analysis. ALPSM and CBAM therefore partially capture uncertainty, although they do not operate at runtime and thereby they suffer from the limitation of design-time analysis.

Classical design patterns presented in [137] extensively study creational (concerned with object instantiation), structural (concerned with relationships between objects) and behavioural patterns (concerned with coordination between objects). These patterns are further categorised according to their static or runtime nature. In that context, reasoning about microservice granularity can benefit from runtime creational design patterns. However, the design patterns of that category in [137] do not capture the scope — boundary — where a pattern can be enforced. Service workflow patterns have been presented in a seminal work [341] which implicitly discussed the issue of granularity is SOAs. However, we envision that the distinction between microservices and SOAs calls for explicitly addressing granularity adaptation decisions in the context of microservice constraints.

2.7.2.2 Partially Autonomous Contributions

In [68, 300], pattern-based engines are proposed to synthesize a composition of atomic and composite services. However, we envision that reasoning about microservice granularity needs to be grounded on objective rather than pattern-based evaluation. In [224, 225], a microservice-specific approach for addressing the microservice granular-

ity problem is proposed which relies on microservice web application log mining to extract the usage pattern and then making adaptive decisions regarding microservice granularity to ensure an economically sustainable architecture. We acknowledge this work is very closely related to the decision support called for in this chapter. Nevertheless, it does not explicitly analyse the value-driven implications of such adaptive decisions as we call for in Section 2.4.2.

2.7.2.3 Fully Autonomous Contributions

The closest fully autonomous contribution to the effective decision support we call for in Section 2.4.2 is the ASTRO-CAptEvo orchestration framework [178]. It is a runtime framework that allows partial definition of business processes for service-based systems at design-time. Subsequently, the framework orchestrates "automatically composing the currently available services, provided by other actors and systems, according to the execution context and the goal of the process to be refined" using state transition systems. This framework takes a runtime approach to decision-making. However, the decision problem targeted by ASTRO-CAptEvo is service composition rather than microservice granularity. Moreover, ASTRO-CAptEvo does not consider objectively reason about composing the available services.

The Self-Serv framework presented in [42] facilitates composite web service execution through peer-to-peer message exchange between coordination agents, which manage the service composition according to a static state chart. This framework can be utilised to address microservice granularity adaptation, but the knowledge that drives the service composition is static, meaning the runtime nature of the granularity problem is not captured in this framework.

Reputation-based dynamic service configuration techniques such as [209] use a policy language to capture service consumers' and providers' profiles and then utilise these profiles to dynamically configure an optimal concrete service architecture. The work in [354] leverages on the concept of reputation by capturing trust in the feedback given regarding the services. In particular, a model is proposed to aggregate feedback from several consumers of a service to reduce the effect of biased feedback. In both

cases, a subjective user profile is used to drive the composition rather than an objective value- and cost-driven approach.

Case-based reasoning about concrete service composition is presented in [199] where a solution space of composite services is formalised using recursive tuples of services. An agent then synthesises the optimal service decomposition given a request for services from the user which are then bound at runtime to concrete services fetched from a registry. A distinction is therefore made here between concrete service selection and the higher level composition of services; this can be utilised to address the granularity problem. However, this solution does not capture the dynamic nature of the microservice granularity problem.

Service composition techniques based on model checking [117, 65, 64] dynamically adapt probabilistic models of the system according to runtime changes in the scenarios surrounding the system or runtime changes in the requirements of the system. In [64] Bayesian learning improves service composition synthesis process through runtime knowledge updates about the system's behaviour over its lifetime. In [65] an abstract service composition is mapped to a concrete service composition at runtime. The field of model-checking therefore is an attractive one for runtime decision-making. Such contributions however need to be leveraged for the specific problem we are concerned with (i.e. the granularity of microservices).

Similar to the field of model checking is runtime architecture modelling. Several contributions in this field manifest runtime changes to the architecture [45, 50, 369, 26, 246, 39, 172, 349, 119, 221]. Other contributions are catered for systems which exhibit a similar level of dynamism to microservices [120, 218, 155]. However, these contributions would need to be leveraged with objective reasoning that considers both the added value and cost of granularity adaptation.

Another approach of an autonomous solution is dynamic service formation rather than dynamic service composition. Frameworks such as [342, 195] provide means to dynamically produce web service specifications conforming to a service composition. These approaches can be utilised to complement the decision support we call for in this chapter.

The runtime, uncertain context of microservice granularity adaptation calls for support similar to that provided by engineering self-adaptivity [148, 75] into an architecture. The role of a self-adaptive solution is to refine and update at runtime the architects' design-time expectations about the architecture's behaviour. There are several mechanisms which can be adopted in this solution [43, 76, 138, 161, 92]. Underlying most of them is the concept of feedback control loops [99, 60] which can be used "to monitor, analyse, plan and execute adaptations [81, p.16]" in a system regarding trade-offs of concern; the knowledge learnt about the system needs to be maintained in a knowledge base (MAPE-K loop [138]).

Control loops can be composed in a centralized, hierarchical, master-slave, or fully decentralized pattern [92]. Each pattern varies in which components of the architecture carry out which phase(s) of the control loop. The centralized pattern is more suitable for monolithic architectures. In a hierarchical pattern, the full MAPE loop is effected at individual services, with higher level services having a more general view of the architecture. The individual service MAPE loops pass information to the higher level loops at short time intervals. Although this pattern is well-suited to addressing the trade-offs of concern here, its only shortcoming is deciding what the higher level component with the global view of the architecture should comprise and the possibility of this service creating a bottleneck. A variant of the hierarchical pattern is the master-slave pattern where the individual services only monitor and execute while a higher level service comprises the analysis and planning phases. Although more lightweight than the hierarchical pattern, the master/slave pattern suffers from the same shortcomings as the hierarchical pattern. The decentralized pattern [143, 208, 350, 358] on the other hand takes away the need for a service with a global view of the control loop. The MAPE loop is implemented in each service and information is passed across the services for decentralized management. The major challenge of this pattern is guaranteeing a consistent view of the system and its environment across all the control loops [92]. However, this pattern is the most aligned with the autonomy of microservice architectures and the scale at which microservices operate.

Runtime architectural adaptation approaches have been proposed before in the Rainbow framework [138] which is the most aligned with the concept of feedback loops. The Rainbow framework provides a reusable solution to induce self-adaptivity into a system in a cost-aware manner. However, it is debatable whether dynamically adapting the level of granularity of a microservice can be captured using the Rainbow framework. This is because the solution space here varies regarding the number of services used and the interaction patterns between them. To our knowledge, the Rainbow framework has not been applied to such a setting before. Another prominent approach is the architectural refactoring approach [96] where a set of anti-patterns is proposed which can be detected dynamically and used to trigger refactoring an architecture. This approach has as its main motive enhancing the modularity of the architecture rather than reasoning about modularity in a objective manner.

Realising microservice granularity adaptation involves changes to a deployed architecture. These activities are similar to those underlying the field of online architectural refactoring. Several contributions in this field pave the way to automated online architectural refactoring [373, 203, 323, 308, 167, 371, 247] and modelling transformations [87, 166]. These contributions are driven by meeting a specific architectural design pattern, but they do not objectively reason about granularity adaptation.

In the field of AI planning, an ontology-based approach to architectural adaptation is proposed [211]. A shared ontology of generic "procedures" (or templates) is produced which the stakeholder can choose from at run-time. An agent then executes this procedure customizing it depending on the scenario in which the system will operate. It is appreciated that the use of ontologies promotes sharing knowledge across across architects. However, we envision that the decision support we are calling for in this chapter can promote knowledge sharing and profiling to inform reasoning about microservice granularity.

2.8 CONCLUSION

In this chapter we contribute a systematic mapping study to consolidate various views, principles, methods and techniques that are commonly adopted to assist the transition to microservices. We systematically describe the study's process and report its results. We use this contribution to introduce a working definition capturing the fundamentals of the transition; we term it as microservitization. Microservitization is a form of servitization [309, 32] where services/components are transformed into microservices — a more fine-grained and autonomic form of services — to introduce added value to the architecture [148]. Microservitization is also an example of a paradigm shift since it involves a dramatic change to the way technical activities are carried out and aligned with a microservice adopter's business objectives. We then shed light on a fundamental problem of microservitization: microservice granularity and reasoning about its adaptation as first-class entities. This study has reviewed and identified gaps in the state-of-the-art and -practice that relate to the modelling approaches, aspects considered, guidelines and processes used to reason about microservice granularity. The identified gaps pave the way to opportunities for future research and development related to reasoning about microservice granularity. In particular, we identify there is room for: 1) systematic architecture-oriented modelling support for microservices that treats their boundaries as adaptable first-class entities, 2) a dynamic architectural evaluation process to reason about the cost and added value of granularity adaptation and, 3) effective, scalability-aware decision support to inform reasoning about microservice granularity at runtime.

Chapter 3

Microservice Ambients: An Architectural Meta-modelling Approach for Microservice Granularity

In Chapter 2, we define microservitization as the paradigm shift towards microservices and determining the optimal microservice boundaries (i.e. microservice granularity) as one of the key microservitization design decision problems. In this chapter, we provide an architecture-centric approach to model this problem, thereby addressing Research Question 2. We build on ambients — a modelling approach that can explicitly capture functional boundaries and their adaptation. We extend the aspect-oriented architectural meta-modelling approach of ambients — AMBIENT-PRISMA — with *microservice ambients*. A microservice ambient is a modelling concept that treats microservice boundaries as an adaptable first-class entity.We use Filmflix to capture a granularity adaptation strategy using our aspect-oriented modelling approach. This shows the ability of microservice ambients to express the functional boundary of a microservice, the concerns of each boundary, the relationships across boundaries and the adaptations of these boundaries. Additionally, we evaluate the expressiveness and effectiveness of microservice ambients using criteria from Architecture Description

Language (ADL) classification frameworks since microservice ambients essentially support architecture description for microservices. The evaluation focuses on the fundamental modelling constructs of microservice ambients and how they support microservitization properties such as utility-driven design, tool heterogeneity and decentralised governance. The evaluation highlights how microservice ambients support analysis, evolution and mobility/location awareness which are significant to quality-driven microservice granularity adaptation. The evaluation is general and irrespective of the particular application domain and the business competencies in that domain. In other words, this evaluation shows how microservice ambients provides the support required to flexibly and expressively model different runtime granularity adaptation strategies (thereby addressing Research Question 2).

3.1 INTRODUCTION

In this chapter, we address the inadequacy of an architecture-centric modelling concept that captures microservice boundaries as adaptable first class entities and facilitate reasoning about the microservice granularity adaptation problem by allowing modelling different strategies for microservice granularity adaptation (related to Research Question 2 — how can microservices be modelled using flexible, expressive support that aids defining different strategies for runtime microservice granularity adaptation?). We address it by contributing to a systematic, flexible and expressive architectural modelling and analysis support for representing microservices and managing their granularities to model this decision problem. A systematic architecture-centric approach for modelling microservice granularity provides the appropriate level of abstraction for managing this runtime decision problem. In particular, this approach has the promise to scale the analysis of this problem. Reasoning about microservice granularity at the architectural rather than code level can facilitate analysis of systems that exhibit heterogeneity and decentralised governance — as is the case with microservice applications. The heterogeneity of tools supporting microservice architectures calls for flexibility in modelling the granularity decision problem in a technology independent way. Architectural modelling of microservices in particular

is a pressing issue due to the inadequacy of standardisation for this young research field [17]. Although intuitively the main trigger for microservitization is "people finding they have a monolith that's too big to modify and deploy [133]", we have not encountered any architectural modelling support in the literature to manage the adaptation from the "too big" or "too small" services to the "good enough" level of granularity.

The novel contribution of this chapter is an architecture-centric modelling concept for microservices which extends ambients [66, 13] — microservice ambients. Microservice ambients provide the primitives for modelling microservices and treat microservice boundaries as adaptable first-class entities; henceforth microservice ambients address Research Question 2. Microservice ambients use "aspects" to define the adaptation behaviour needed to support changes in granularity at runtime. Aspects flexibly separate cross-cutting concerns (and thereby scope) of each boundary. We introduce the *granularity adaptation aspect* to help define the runtime triggers for granularity adaptation to be used as the mciroservice(s) are monitored at runtime. When any of these triggers is invoked, the graphical notation of microservice ambients provides architectural modelling support to express the candidate solution for granularity adaptation. The transition (by merging or decomposing a microservice ambient(s)) from an initial architecture to a chosen candidate is captured in the granularity adaptation aspect of the microservice ambient.

We use Filmflix to demonstrate the flexibility and expressiveness of the modelling approach in capturing granularity adaptation strategies. We evaluate our modelling approach using properties from ADL classification frameworks [213, 12, 286] since microservice ambients essentially support architecture description for microservices. The evaluation focuses on the fundamental modelling constructs of microservice ambients and how they support microservitization properties such as utility-driven design, tool heterogeneity and decentralised governance. The evaluation also highlights how microservice ambients support runtime analysis, mobility and location awareness; all of which are significant to quality-driven microservice granularity adaptation. The evaluation is general and irrespective of the particular application domain and the business competencies in that domain.

In Section 3.3 we motivate the need for our contribution using a motivation scenario from Filmflix. We then use this scenario to formulate the modelling requirements suited to our problem in Section 3.4. In Section 3.2 we give a brief background to ambients. In Section 3.5 we introduce our novel *microservice ambients*. In Section 3.6 we present our evaluation of microservice ambients. We summarise the chapter contents in Section 3.7.

3.2 BACKGROUND: AMBIENTS

Ambient-PRISMA [13] extends traditional architectural elements with a new kind of element called an ambient inspired from Ambient Calculus [66]. Ambients are architectural elements that "coordinate a boundary, model the notion of location and provide mobility support to other architectural elements [13]." Ambients locate other architectural elements in their boundary and manage them. Therefore, ambients can be utilised as runtime modelling analysis tools, where the parent ambient manages the interaction between its children and exterior architectural elements. For example, in Figure 3.7.a the *MovieReview* ambient is the parent of *RevPolM* and *RevInfoReqM* microservice ambients, so any access to these children ambients from outside the parent ambient is managed by it.

AMBIENT-PRISMA separates the behaviour of cross-cutting concerns through a set of aspects. Aspects give a white-box view of each ambient's boundary. Ambient-PRISMA is supported by a specification language that allows an aspect-oriented description of the software architecture. Crucially, this specification language provides just enough insight into the behaviour (i.e. concerns) of each ambient to model an architecture expressively — capturing the behaviour within ambient and the relationships across ambients. The aspects communicate through weaving relationships. Each aspect fulfils its concern by utilising interface services. Many aspects can utilise the same interface service if need be. Ambients publish interface services through ports. Attachments represent the communication channel between a port of an ambient and any architectural element located inside or outside that ambient. The interface services that we utilise for the granularity adaptation problem are [13, p8,p9]:

- *newAmbient* creates a new ambient by providing the name of the ambient as a parameter.
- *addChild* adds a new architectural element in the boundary of an ambient.
- *removeChild* removes an architectural element that is located in the boundary of an ambient.

3.3 MOTIVATING SCENARIO

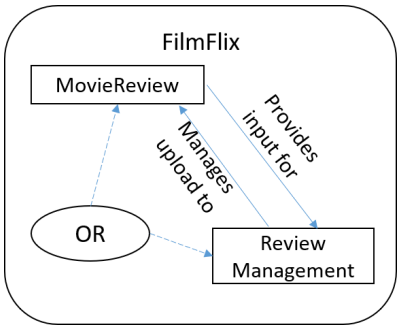
Consider that the architects utilise the same notation of Figure 1.1 for modelling the initial architecture of Filmflix and a solution space of architectures with varying granularity levels. We assume the intuition of the architects when constructing these candidates is to utilise domain-driven design concepts, such that each bounded context represents an independent area of the domain (inspired by [233, 112]). Independence here implies that this domain has consistent rules " in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas [112]" and these rules only apply within that bounded context. Each modular boundary represents a single microservice within a bounded context. The intuition here is that a modular boundary encapsulates more concretely the related functionalities that belong to the same bounded context.

Furthermore, consider that the architects define decomposition rules such as XOR and OR to represent the interaction between respective modular boundaries if the granularity of the initial architecture is adapted by adopting the respective candidate. For example, consider that the granularity of Figure 1.1 is adapted by merging *ReviewRegulation* and *ReviewUpload* in the initial architecture into *ReviewManagement*, thereby adopting the candidate presented in Figure 3.1a. The architects' intuition behind the OR rule in Figure 3.1a is that computing power can be alternated between *ReviewManagement* and *MovieReview*. Adapting granularity by merging *ReviewRegulation* and *ReviewUpload* aims to avoid unnecessary frequent inter-microservice communication between them.

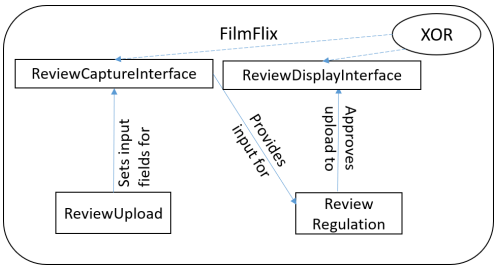
On the other hand, Figure 3.1b calls for adapting granularity by decomposing the *MovieReview* user interface to separate functionalities related to capturing input and

displaying reviews. This aims to give the architects flexibility in the development tool choices for each set of functionalities. The XOR rule between *ReviewCaptureInterface* and *ReviewDisplayInterface* in Figure 3.1b indicates that they are related so they can not run independently if the initial architecture's granularity is adapted by adopting candidate Figure 3.1b.

This setting raises the following challenges (among others): 1) what is concise the difference between a "modular boundary" and a "bounded context" and, 2) how does the interaction between modular boundaries differ for XOR and OR rules? The first challenge shows the significance of providing *systematic architectural modelling to express functional granularity in microservices.* The second challenge shows the significance of *flexible and expressive analysis support to clearly capture how granularity adaptation should be performed* (e.g., does the OR rule mean newly formed modular boundaries can be invoked in parallel and have simultaneous access to end user information?).



(a) Candidate 1 — calls for merging *ReviewUpload* and *ReviewRegulation* into *ReviewManagement*; the OR rule allows running *ReviewManagement* and *MovieReview* in parallel



(b) Candidate 2 — calls for decomposing *MovieReview* into *ReviewCaptureInterface* and *ReviewDisplayInterface*; the XOR rule indicates these 2 microservices can not be invoked in parallel

Fig. 3.1 Impromptu candidate solution space with different microservice granularity levels

3.4 PROBLEM FORMULATION

Reflecting on Section 3.3, we describe the requirements of a systematic architecture-centric approach to model microservice granularity:

- **Requirement 1:** The approach shall explicitly capture the primitives for granularity adaptation — microservice boundaries — as first-class entities. Boundaries are primitives for granularity adaptation because they are the effectors/actuators of granularity adaptation design decisions.

- **Requirement 2:** The approach shall promote flexibility and expressiveness in the modelling microservice granularity behaviour, thereby supporting runtime analysis of this behaviour. Crucially this analysis needs to optimise for autonomy of computation and independent deployability of the microservices. The objective of the analysis is to avoid aggressive decomposition of functionality.

The concept of ambients is particularly attractive for the requirements above. *We extend ambients by introducing a microservice ambient type. A microservice ambient as an architectural element allows modelling the boundary of computation of a microservice (Requirement 1).* Aspects, weaving relationships, ports and attachments model the impacts on the concerns and relationships when these boundaries are adapted. *We enrich the microservice ambient with a granularity adaptation aspect to express granularity adaptation strategies as a set of distinct conditions and decomposition/merging steps. This in turn facilitates runtime analysis for different quality trade-offs for different granularity levels of a microservice ambient (Requirement 2)).* Capturing granularity adaptation behaviour as a transactional set of decomposition/merging steps to facilitate runtime analysis is a novel contribution of this work. Other ADLs we have examined in the literature either assume static modelling of the architectural elements or provide little support for their evolution (e.g. by inheritance or component replication); we discuss this further in Section 3.6.

3.5 MICROSERVICE AMBIENT DESCRIPTION

A microservice ambient uses meta-services (e.g., related to updating to updating the ambient hierarchy when need be) to reflect on its behaviour and invoke meta-services at runtime. These meta-properties and meta-services are defined in the meta-model [13]. A microservice ambient can encapsulate further microservice ambients in a hierarchical manner. As long as the autonomy of computation (and thereafter

independent deployability) are enforced by the boundary, it is fair to assume each microservice ambient in a hierarchy will be instantiated as a concrete microservice. Nevertheless, each microservice will be of a different granularity level, depending on the hierarchy position of the microservice ambient instantiated.

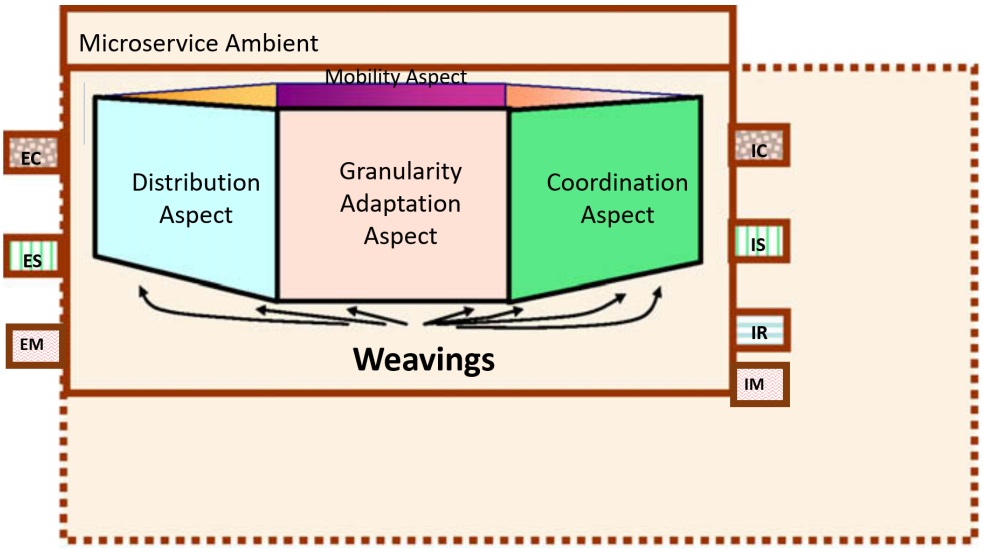


Fig. 3.2 White-box view of a microservice ambient; IC and EC are ports related to the mobility aspect, IS and ES are ports related to the coordination aspect, IR is the port related to the distribution aspect, InM and ExM are ports related to granularity adaptation aspect

A microservice ambient can coordinate with its parent ambient (or microservice ambient) to fulfil the behaviour defined by its aspects. Additionally, the microservice ambient can act as an autonomic element that employs the phases of the MAPE-K loop to model the runtime granularity adaptation [11]. The MAPE-K loop defines the stages for an autonomous element to plan the adaptation of a running system: monitoring the running system, analysing monitored data, planning the adaptation actions, executing the plan on the running system, and finally updating the knowledge base of the autonomous element for future planning optimisation. However, in this chapter, we focus more on providing architectural modelling support for the decision problem rather than stating a decision-making method for choosing a granularity adaptation candidate.

A microservice ambient must have at least four aspects to support architectural modelling for granularity: a granularity adaptation aspect (our novel contribution), a mobility aspect (inherited from ambients [13]), a coordination aspect (inherited from

ambients [13]) and a distribution aspect (inherited from ambients [13]). We elaborate on the role of each aspect below. The overall white-box view of a microservice ambient is shown in Figure 3.2.

- **Granularity adaptation aspect:** This aspect is responsible for describing the behaviour of how a microservice ambient adapts its granularity. The aspect utilises the *newAmbient,addChild,removeChild,addAttachment and removeAttachment* interface services inherited from the definition of ambients (Section 3.2). At runtime, the granularity adaptation aspect monitors parameters (e.g., change rate, failure rate) indicating QoS. It then utilises the inherited interface services to trigger granularity adaptations in response to changes in the runtime parameters.

- **Mobility aspect:** In the context of supporting granularity adaptation, the mobility aspect allows microservice ambients to enter or exit the boundaries of a parent microservice ambient. There are two ports related to this aspect: *ExCapabilities* port (EC) and *InCapabilites* (IC) port.

- **Coordination aspect:** At a high level, the coordination aspect redirects calls from external architectural elements to internal architectural elements inside an ambient. It also manages the redirection of calls from internal architectural elements to external ones. There are two ports related to the coordination aspect: *InServices* (IS) port and *ExServices* (ES) port.

- **Distribution aspect:** The role of the distribution aspect is to make an ambient aware of its hierarchical position by storing the name of its parent ambient. The predefined weaving relationship between the mobility and distribution aspect ensures the distribution aspect manages this awareness when adaptations in the hierarchy are triggered by the mobility aspect. There is a single port related to this aspect: *InRouting* (IR) port.

Microservice ambients inherit all the invariants imposed on ambients [13]. The introduction of the granularity adaptation aspect however introduces an extra invariant specific to microservice ambients. All microservice ambients must implement a

granularity adaptation aspect whose concern is granularity (***invGranAdapt***). In addition, the following weaving and port constraints are introduced for the microservice ambient:

- The *GranAdapt* aspect definition implement an *IMonitor* interface to define the runtime monitoring requirements of the granularity adaptation aspect. Publishing and requesting the services of this interface indicates the need for two additional ports in a microservice ambient:
  - An *InMonitorPort* (IM) is required to publish/request the runtime monitoring interface service to architectural elements inside the microservice ambient.
  - An *ExMonitorPort* (EM) is required to publish/request the monitoring service to architectural elements outside the microservice ambient. The *ExMonitorPort* of children microservice ambients is connected to the *InMonitorPort* of its parent microservice ambients.
- The granularity adaptation aspect introduces a weaving relationships between itself and the distribution aspect.

3.5.1 GranAdapt Aspect and Microservice Ambient Templates

We have defined different templates that provide guidance for architects when defining a microservice ambient and its granularity adaptation behaviour. Architects can then instantiate these templates to capture specific granularity adaptation strategies. A template of the granularity adaptation aspect is outlined in Figure 3.3, using the Ambient-PRISMA Textual Language that is based on a variant of dynamic logic [258]. The list of architectural elements which a microservice ambient is allowed to monitor is stored in set called *Sources* and the size of the set is stored in *SourceNum*. Each *Source* stores the architectural element located in an ambient, its name and the corresponding *Parameter*s monitored for it. A *Parameter* is characterised by its name, *ParameterKind* and sequence of recorded values. The *recordReading IMonitor* interface service receives parameters readings from an architectural element inside

an ambient implementing this service. The *getReading IMonitor* interface service publishes the latest reading recorded for a parameter of an architectural element. External microservice ambients can request readings for architectural elements that they can not directly communicate with. The italicised terms are variability points in the template, which can be instantiated depending on the adopted granularity adaptation strategy.

3.5.2 Architectural Configuration Specification

A template of the microservice ambient importing the granularity adaptation aspect is outlined in Figure 3.4, also using the Ambient-PRISMA Textual Language. This template includes all the invariants a microservice ambient needs to fulfil. The highlighted lines refer to *invGranAdapt* invariant. Because it is a template, Figure 3.4 does not specify the architectural elements within the microservice ambient or the attachments between them [13]. Moreover, it is assumed that the attachments between a microservice ambient and its children architectural elements is automatically managed by the runtime environment of the microservice ambient [13].

Figure 3.5 illustrates a possible aspect type that uses the template in Figure 3.3. Intuitively, it is motivated by clustering simultaneously changing architectural elements within the same microservice ambient. Here the number of attachments (*AttNumber*) across children architectural elements of a microservice ambient (*mem1* and *mem2*) is the indicator of simultaneous change. The parent microservice ambient (*self*) utilises the *getReading* interface service for this monitoring. Where simultaneous change occurs in the number of attachments within *mem1* and *mem2*, this triggers merging *mem1* and *mem2* into a single microservice ambient.

Figure 3.3 defines merging as a transaction, allowing the rollback of merging and decomposition in case of error. These transactions utilise the interface services from *IMonitor* and establish the changes in the ambient hierarchy required when a child microservice ambient enters or leaves its parent ambient. Crucially, these changes assume that only sibling microservice ambients are merged or decomposed. If a transaction is invoked on non-sibling microservice ambients, the services utilised in the transactions and the triggers in Figure 3.5 would have been different. The *Valuations*

Granularity Aspect GranAdapt **using** IMonitor

Attributes
Sources: **Set** (Source);
SourceNum:**Integer**
....
Transactions in DECOMPOSE (**input** AESet: **Set(**ArchitecturalElement)):
New1 = newAmbient(AmbName: string);
New2 = newAmbient(AmbName: string);
New1.addChild(AESet.n^{th} *element.getChildren().subset1*);
New2.addChild(AESet.n^{th} *element.getChildren().subset2*);
newSource ("new1",new1,*ParameterList*);
sources.add(new1);
newSource ("new2",new1,*ParameterList*);
sources.add(new2);
sources.delete(AESet. n^{th} *element*);
recordReading(*"new1",ParamName,ParameterKind)*
recordReading(*"new2",ParamName,ParameterKind)*

Transactions in MERGE (**input** AESet: **Set(**ArchitecturalElement)):
ChildrenA = AESet.n^{th}*element.getChildren();*
ChildrenB = AESet.m^{th}*element.getChildren();*
newAmbient(AmbName : string);
AmbName.addChild(*ChildrenA*);
AmbName.addChild(*ChildrenB);*
newSource ("ambName",ambName,*ParameterList*);
sources.add(AmbName);
sources.delete(AESet. n^{th} *element*);
sources.delete(AESet. m^{th} *element*);
recordReading("AmbName",*ParamName,ParameterKind)*

Valuations
[*trigger*] *Attribute=NewValue*;
Triggers
MERGE(AESet)/DECOMPOSE(AESet)
when
Sources.ParamIter().includes(*ParamName*) &
Parameter reading comparisons among N microservice ambients
End_Aspect

Fig. 3.3 GranAdapt aspect template specification

```
Ambient_Microservice type ReviewFR

Import Granularity Adaptation Aspect GranAdapt
Import Mobility Aspect MobilityAspect
Import Coordination Aspect ACoordination
Import Distribution Aspect ADist
Weavings
        ADist.getLocation(Location) instead
MobilityAspect.getParent(Parent);
    GranAdapt.removeChild(AE) after
ADistAspect.getLocation(Location);
End_Weavings

Ports
    InMonitorPort: IMonitor;
    ExMonitorPort: IMonitor;
    InCapabilitiesPort: ICapability;
    ECapabilitiesPort: ICapability;
    InServicesPort: ICall;
    EServicesPort: ICall;
    InRoutePort: IGetRoute;
End_Ports
End Ambient_Microservice type ReviewFR
```

Fig. 3.4 Our microservice ambient template specification; leveraging upon ambients in [13]

part of Figure 3.5 updates the list of microservice architectural elements that *self* would have to monitor after merging. This involves removing *mem1* and *mem2* from the *Sources* of *self* and replacing them with the newly formed microservice ambient *cluster*. Once we have the types of our architecture, we create different configurations by instantiating the elements including ambients and defining the ambient hierarchies. Figure 3.7.a illustrates a possible instantiation — architectural configuration — of the microservice ambient template. A textual reflection of this graphical architectural configuration is shown in Figure 3.6. *MovieReviews* is an instance of the microservice ambient *ReviewFR*.

Referring to Figure 3.7.a, *MovieReview* is the highest level microservice ambient therefore its constructor does not take any parameters. In reality however, this

```
Granularity Aspect GranAdapt using IMonitor
Attributes
Sources: Set (Source);
SourceNum:Integer
....
Transactions
      DECOMPOSE (input AESet: Set(ArchitecturalElement))
      MERGE (input AESet: Set(ArchitecturalElement))
Valuations
      [self.addChild(cluster)] Sources.add(new Source("cluster",cluster,new
Parameter("AttNumber",Integer,[])));
      [self.addChild(cluster)] SourcesNum=SourceNum+1;

      [self.removeChild(mem1)] Sources.delete (mem1);
      [self.removeChild(mem1)] SourcesNum=SourceNum-1;

      [self.removeChild(mem2)] Sources.delete(mem2);
      [self.removeChild(mem2)] SourcesNum=SourceNum-1;
Triggers
      MERGE({mem1,mem2})
when
      Sources.ParamIter().includes("ChangeEvent") &
      getReading(self,"mem1","AttNumber")=getReading(self,"mem2","AttNumber")
End_Aspect
```

Fig. 3.5 Granularity adaptation aspect for clustering changes

microservice ambient might reside within a larger ambient but for the purpose of illustration we focus on this scope of the architectural configuration. *InfoReqDef1* and *RevPolDef1* are instances of the functional elements residing in *RevInfoReqM* and *RevPolM* respectively. Because the runtime environment maintains the attachments between the microservice ambient and its children architectural elements, only attachments between the children architectural elements need to be explicitly configured. Therefore, only the attachment between *RevInfoReqM* and *RevPolM* is explicitly defined. Moreover, this automatic runtime management of attachment ensures that granularity adaptation is invoked reliably —- the attachments are maintained correctly after the adaptation is invoked.

Both the template and configuration specifications are defined at design time according to a granularity adaptation strategy intuition (in this case clustering simultaneous changes together). At runtime, the transactions and valuations of the instantiated

aspects and the invariants of the instantiated ambients are executed and managed. The power of the triggers in the granularity adaptation aspect can be extended such that it controls runtime switching across different instances of the granularity adaptation aspect, thereby changing the theme of microservitization at runtime.

```
Architectural_Model_Configuration MovieReviewsConf

New MovieReviews
{
MovieReview = new ReviewFR ();
RevInfoReqM=new ReviewFR (MovieReview);
RevPolM=new ReviewFR (MovieReview);

InfoReqDef1 = new InforReqDef  (``RevInfoReqM'');
RevPolDef1 = new RevPolDef (“RevPolM”);

AttRevInfoRevPol1 = new AttRevInfoRevPol (RevInfoReqM,
ESInfoPort,RevPolM, ESPolPort);
```

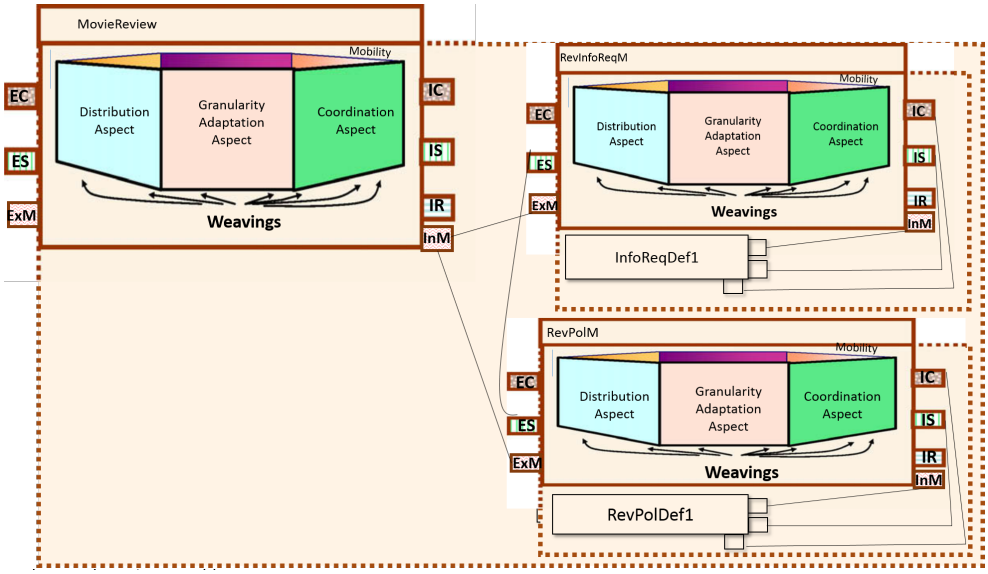
Fig. 3.6 Possible architectural configuration using microservice ambient instances

3.6 EVALUATION

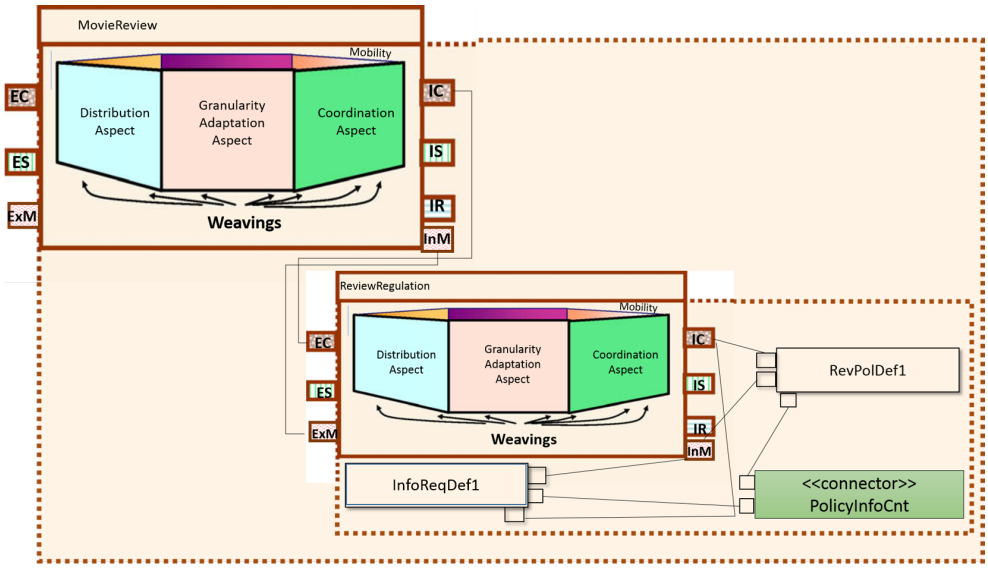
In this section we apply our microservice ambients to model a granularity adaptation strategy using Filmflix as an example. In Appendix B we present a number of sample strategies that can be formulated using granularity adaptation aspects. We then evaluate our modelling approach for flexibility, expressiveness and support for runtime analysis using properties from [213, 12, 286]. Evaluation along these dimensions help show the extent to which our contribution provides the support required to flexibly and expressively model different runtime granularity adaptation strategies (as aimed for in Research Question 2).

3.6.1 Clustering for Localising Change

Rationale: Change clustering has been applied in industry in the context of microservices; it was however motivated merely by enhancing modularity [130]. In other



(a) Problem Context



(b) Solution

Fig. 3.7 Merging for Change Clustering; *InfoReqDef1* and *RevPolDef1* are instances of the functional elements residing in *RevInfoReqM* and *RevPolM* respectively

words, different rates of code churn implied a different structure of the microservice architecture rather than a different level of granularity. We leverage on change clustering but in the context of adapting granularity. *Problem Context:* There are cases where sets of functional elements change simultaneously over time. Figure 3.7a exemplifies such a scenario. The change events in the *RevInfoReqM* (managing the user input requirements when uploading a review) and *RevPolM* (managing the regulations on the contents of the uploaded reviews) are monitored by the parent *MovieReview*

ambient. The attachment between the ExM (ExMonitor) ports of the sub-ambients and the InM (InMonitor) port of the parent ambient enforces this monitoring.

Problem: Localising change and reducing its ripple effects through granularity adaptation.

Main driving forces:

- Reducing logical dependencies across microservice boundaries, thereby enhancing the autonomy of each individual microservice and the maintainability/evolvability of the overall architecture.
- Enhancing the productivity and independence of the team responsible for a microservice.

Invoking granularity adaptation: In Figure 3.7b, the related functional elements are now encapsulated by a single microservice ambient after the granularity adaptation is applied. The *GranAdapt* aspect definition of this strategy is shown in Figure 3.5. Reflecting on the number of attachments, there is a total of 9 attachments in Figure 3.7a (represented by lines between the ports). After invoking the granularity adaptation aspect, there is a total of 8 attachments within the *MovieReview* microservice ambient in Figure 3.7b. This reduction in the number of attachments is an indicator of a slight reduction in the logical dependencies when this adaptation is invoked. Although reducing one attachment can seem insignificant at the modelling level, it can translate to a significant reduction in deployment costs at the underlying code level and subsequent reduction in maintenance and testing efforts over time.

Further analysis can help elaborate the merging decision further. Such analysis can optimise for balancing between the reduction in logical dependencies and the accompanied increase in overheads. Examples of overheads include the cost of merging/decomposing the underlying code, additional network link costs (in the case of decomposition), and additional data format translation (in the case of merging).

Logical dependency reduction can enhance the productivity of the team managing the microservice. It particular, the reduction easier for identifying the right expertise that can best deal with the design, development and utility-driven evolution of the microservice, as these concerns to big extent are realised in the “glues” between the

microservices — the logical dependencies. Allocating the right expertise potentially reduces long-term social and technical debt in large-scale, distributed systems. This is the case for many microservice applications.

3.6.2 Qualitative Evaluation

3.6.2.1 Effectiveness and Expressiveness of Modelling

We use properties from the ADL classification framework in [213] to evaluate the expressiveness and effectiveness of microservice ambients in providing systematic architectural support for modelling microservice boundaries (Requirement 1). ADLs are deemed crucial to architecture-centric modelling. We evaluate the expressiveness and effectiveness of microservice ambients using properties from an ADL classification framework as microservice ambients can essentially support an ADL that realises the concerns of microservices. We evaluate microservice ambients irrespective of the particular application domain and the business competencies in that domain, therefore we use this framework as opposed to the more recent framework presented in [212] which comprises the domain and business aspects of modelling architectures. We elicited the classification framework properties which are relevant to microservitization to make the evaluation more focussed.

According to [213], the essential elements that an ADL must explicitly model are components, connections, and their architectural configurations. Components (microservice ambients in our case) are the unit of computation in an ADL. Connections in turn model the relationships across these components. Architectural configurations represent the overview of the architecture comprising both components and connections [213]. Architectural configurations therefore depict the impact of design decisions regarding granularity on the overall microservice architecture.

3.6.2.1.1 Component modelling

- **Component Interface**: The interface of a component defines computational commitments a component can make and constraints on its usage [213]. Microservice ambients have commitments regarding the scope of runtime monitoring for each

granularity adaptation strategy. A microservice ambient also has commitments on which architectural elements it can control when triggering granularity adaptation. The monitoring commitments are captured by the *Sources* parameter of the granularity adaptation aspect. In Figure 3.5, the *Sources* parameters only include the children of the microservice ambient importing those aspect instances. For example, the instantiation of the *Sources* parameter in the granularity adaptation aspect of the *MovieReview* microservice ambient in Figure 3.7.a would include only *RevInfoReqM* and *RevPolM*. On the other hand, the commitments of control are captured by the input to the *MERGE* and *DECOMPOSE* transactions of the granularity adaptation aspect. For example, the input to the *MERGE* transaction in Figure 3.5 would be *RevInfoReqM* and *RevPolM* referring to Figure 3.7.a. Most existing ADLs (e.g., Aesop [139], SADL [282], and Wright [15]) support modelling component interfaces despite under different names (e.g., component ports, players, or provides/requires interfaces). Our specialised MERGE/DECOMPOSE transactions make microservice ambients more suitable for the target problem of this thesis that counterpart ADLs which do not explicitly capture the commitments related to granularity adaptation.

- **Component Evolution**: We introduce the granularity adaptation aspect to support boundary evolution of a microservice ambient. Figure 3.7 demonstrates how the granularity adaptation aspect can enforce evolution of the architecture to optimise for multiple quality drivers. This is a novelty compared to absent or limited support for component evolution in other examined ADLs. MetaH [7] and UniCon [313] for example do not support component evolution thereby they can't be used to reason about dynamic problems such as microservice granularity. Other ADLs such as ACME [69], Rapide [204] and C2 [214] support component evolution by sub-typing, which is a less suitable counterpart to decomposition/merging which we support using transactions in the granularity adaptation aspect.

- **Component non-functional properties**: Localising change addresses the changeability and replaceability non-functional properties of the architecture. Since both properties can be encoded in the granularity adaptation aspect, our contribution sup-

ports modelling distinct quality trade-offs related to different levels of microservice granularity. Since microservitization is ultimately a utility-driven exercise [148], modelling non-functional properties is particularly significant to microservitization since it allows reasoning about the impact of granularity adaptation strategies on utility. Other ADLs do provide support for modelling non-functional properties as well (e.g., MetaH [6] and UniCon [312]) despite specifically related to schedulability analysis.

3.6.2.1.2 Connection modelling

- **Connection types**: Microservice ambients make explicit use of two types of connection: attachments and weavings. Attachments model the relationship across microservice ambients (and other architectural elements). Therefore, attachments are central to analysing the impact of adapting a microservice boundary. Weavings on the other hand enforce reliable granularity adaptation. The weaving relationship we introduce between the granularity adaptation aspect and the distribution aspect ensures that a granularity adaptation triggered by the granularity adaptation aspect is reflected in the hierarchy of ambients managed by the distribution aspect. ADLs such as ACME [69], AESOP [139], and C2 [214] support modelling different connection types based on protocols. We argue however that due to the heterogeneity of microservice architectures limiting the connection types to a certain set of protocols would reduce the practicality of these ADLs in the context of microservices.

3.6.2.1.3 Architectural Configuration Modelling

- **Configuration Understandability**: The graphical support for microservice ambients enables visualising the overall configuration. In particular, the graphical support is detailed enough to express granularity adaptation strategies in action. However, it is still abstract enough to hide implementation details of this adaptation. On the other hand, the textual support for modelling the configuration (Figure 3.6) complements the graphical support and sharpens the overall understandability of the modelling approach. ADLs such as ACME [69], MetaH [6], and C2 [214] all provide textual explicit support for configuration understandability. However, ambi-

ents and microservice ambients complement this with graphical support yielding them a more user-friendly modelling approach.

- **Configuration compositionality and scalability**: A compositional ADL supports describing architectural configurations at different levels of detail [213]. The ambient approach in general allows hierarchical composition of architectural elements; it supports compositionality. Support for compositionality facilitates iterative analysis of granularity adaptation strategies. For example, the internals of the *ReviewRegulation* ambient in Figure 3.7.b can be further analysed to invoke granularity adaptation within this microservice ambient. ADLs such as Rapide and SADL support configuration compositionability through mappings from the architecture to the interface. Although we do acknowledge this is suitable for our context, Rapide [204] and SADL [282] do lack required support along other criteria in our framework. Therefore, we argue they are not comprehensive solutions for our problem in this chapter.

- **Configuration refinement and traceability**: An ADL must enable consistent mapping between its graphical notation and an executable system [213]. An aspect specification can be automatically mapped to a runnable modelling construct using AMBIENT-PRISMANET middleware [13] rendering an implementable runtime model of the solution space of this decision problem. ADLs such as Rapide [204] and SADL [282] use refinement maps enable comparative simulations of architectures at different levels. We argue however that these maps would not be scalable enough to accommodate large-scale microservice architectures.

- **Configuration heterogeneity**: It is essential for ADLs to facilitate modelling architectures that employ varying technological choices (e.g., different programming languages) [213]. The aspect-oriented nature of microservice ambients supports such heterogeneous modelling. Crucially, the granularity adaptation aspect captures granularity adaptation behaviour independent of the technologies used within the architectural elements involved in Sthis behaviour. Support for modelling heterogeneous configurations aligns with the decentralised governance enforced by microservitization [148]. Existing ADLs in the literature vary in the scope of languages they support (e.g., Aesop supports C [139], and C2 supports C++,

Java and Ada [214]). However, ambients abstract away from support for specific programming languages.

- **Configuration evolvability and dynamism**: ADLs need to support incremental addition, removal, replacement, and reconnection of components and connections in a configuration [213]. The *MERGE* and *DECOMPOSE* transactions of the granularity adaptation aspect provide reliable support for this dynamism — a novelty in our work. ADLs such as Darwin [207] and [204] support reconnection, addition and removal of components and connectors. However, the reliability of these changes are not ensured.

- **Configuration constraints**: An ADL needs to model configuration-wide dependencies in addition to component constraints [213]. Configuration constraints are supported by attachments, since they depict the dependencies across microservice ambients. Support for defining configuration constraints is essential to capturing the impact of different microservitization drivers (e.g., enhancing team productivity and architecture changeability) on the overall architecture. In ADLs such Aesop [139] and Wright [15], constraints are captured by attaching ports to specific roles. We acknowledge that such support is suitable for our context as well. However, these ADLs lack support for other essential features related to modelling microservice architectures.

- **Configuration non-functional properties**: Analogous to component non-functional properties, an ADL needs to support representing configuration-wide non-functional properties [213]. We support this by providing a template of the microservice ambient. Each architectural configuration in turn instantiates this template according to the required non-functional properties.

3.6.2.2 Facilitating Design Time and Runtime Analysis

We use properties from [12, 286] to evaluate the support that microservice ambients provide for runtime analysis of granularity (Requirement 2). In this section we illustrate the significance of each feature in relation to runtime reasoning about microservice granularity and how it is supported in the microservice ambient approach.

- **Location:** Location refers to space where an architectural element is allowed to move. This is significant to defining the scope of the solution space in the granularity adaptation problem. Location is enforced by the *Sources* parameter of the granularity adaptation aspect. The architectural elements that a microservice ambient monitors determines the ambient's scope of knowledge about its runtime environment. This knowledge in turn confines the space in which the children of a microservice ambient can be decomposed or merged. This support for location is novel to microservice ambients. Although most ADLs examined (e.g. [69], [214], and [207]) support logical location by allowing hierarchical compositions of components and connectors, they do not capture explicitly space in the hierarchy where these components are allowed to move, which is significant to the microservice granularity adaptation problem.

- **Location-awareness:** Awareness of location is supported by the distribution aspect imported by the microservice ambient. In addition, the weaving relation between the granularity adaptation aspect and the distribution aspect ensures that location awareness is maintained after granularity adaptation decisions are executed. Location-awareness is a co-requisite for supporting location; only if a microservice ambient is aware of its position can it reason about the candidate solution space for granularity adaptation. Upon examining ADLs such as [282] and [204], we did not find support for location awareness within components of the architecture despite existing support for location modelling.

- **Unit of Mobility:** This is the most central feature to the granularity adaptation problem. Microservice ambients by definition facilitate runtime reasoning about the unit of mobility in a microservice architecture. The unit of mobility refers to the "smallest entity of a model that is allowed to move [12, p.2]." Support for component evolution and configuration dynamism in the previous subsection are both pre-requisites for reasoning about the unit of mobility. In turn, it is the granularity adaptation aspect which defines the unit of mobility when decomposing or merging a microservice ambient. This explicit support for unit of mobility is a novelty in this work compared to ADLs such as ACME [69]. In these ADLs,

invoking a modification to an architectural configuration involves re-defining both components and connectors. In microservice ambients, it is the runtime environment which implicitly handles the implications of granularity adaptation, allowing the unit of mobility to be captured clearly as inputs to the decompose and merge transactions (Figure 3.5).

3.7 CONCLUSION

In this chapter we provide an architecture-centric approach to model microservice granularity. In particular, we extend the aspect-oriented meta-modelling approach of ambients with ***microservice ambients*** — a modelling concept that is unique in treating boundaries as an adaptable first-class entity of microservices. We demonstrate the use and significance of our approach by using it to capture a possible granularity adaptation strategy in Filmflix. This presentation shows that microservice ambients can expressively capture granularity adaptation strategies enhancing distinct utilities of microservitization. We further outline other examples of granularity adaptation strategies inspired by microservice adopter practices that can be captured using granularity adaptation aspects. Additionally, we evaluate the microservice ambients using properties from ADL classification frameworks. The evaluation shows the potential of microservice ambients as an ADL for microservices. The evaluation highlights how microservice ambients support analysis, evolution and mobility/location awareness. Given that microservice ambients can be used to model candidate granularity adaptation strategies, in the next chapter we present our contribution to answer a critical question regarding granularity adaptation: Is it worth pursuing microservice granularity adaptation in the first place?

Chapter 4

Dynamic Evaluation of Microservice Granularity Adaptation

In Chapter 3, we contribute to the primitives for modelling microservices and defining granularity adaptation strategies to be adopted if it is worth adapting microservice granularity. However, there is no existing work investigating how to reason about the added value of granularity adaptation under uncertainty at runtime [27, 334]. This gives rise to Research Question 3: Can the trade-off between predicted benefit and cost of pursuing microservice granularity adaptation (or not) be objectively reasoned about and tracked under uncertainty? To address this question, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem. Architects have the right without a symmetric obligation to reap benefits of these options. We propose a novel evaluation process for dynamically evaluating the granularity adaptation design decisions under uncertainty. Our process is based on a novel combination of real options and the concept of Bayesian surprises. Our evaluation process dynamically determines at runtime if it would be worth adapting granularity from the perspective of introducing added value. We show the benefits of our evaluation process by comparing it to four representative industrial microservice runtime monitoring tools — Amazon Web Service (AWS) Cloudwatch, X-Pack, IBM

Bluemix and Riemann. In the state-of-the-practice, these tools can be used to reason retrospectively about granularity adaptation decisions. Therefore, our comparison shows how using our process can be used to dynamically evaluate the added value of granularity adaptation at runtime. We implement a microservice application — Filmflix — using Amazon Web Service (AWS) Lambda and use this implementation as a case study to show how our process can go beyond traditional application of real options analysis in revealing dynamic added value trends of granularity adaptation in response to runtime evidence. We further show the feasibility of our process to assist microservice adopters in industry by presenting a Java implementation of it.

4.1 INTRODUCTION

Microservitization is highly motivated by introducing added value to the software architecture. This added value can be introduced as a result of the microservice architecture's ability to cope with operation, maintenance and evolution uncertainties. Ultimately, this can also relate to improved maintenance costs and cost-effective quality of service (QoS) provision to end users.

One of the critical challenges related to the transition from a microservice architecture design to another is reasoning about the suitable granularity level of a microservice. A granularity level determines "the service size and the scope of functionality a service exposes [194, p.426]." Granularity adaptation entails merging or decomposing microservices thereby moving to a finer or more coarse grained granularity level. This problem is critical because splitting microservices too soon can lead to negative consequences on the architecture. It will likely happen that the software architect will learn about the suitable granularity levels along the architecture's lifetime rather than at the beginning of transiting to microservices [93]. Although the distinction between service-oriented architectures and microservices is subject to active research, there is a critical distinction between both styles which makes the granularity problem even more so critical for microservices: "SOA was primarily a vendor led phenomenon, microservices is by contrast largely being driven from the bottom up

[242]." Therefore, microservice granularity decisions have a very significant impact on higher level decisions related to the microservice architecture.

In this chapter, we address the inadequacy of existing evaluation processes in dynamically evaluating microservices granularity adaptation decisions under uncertainty. The common practice in industry is to reason about microservice granularity adaptation through retrospective analysis, using metrics and operations logs recorded at runtime by microservice monitoring tools (e.g., Amazon Web Service (AWS) Cloudwatch [353], X-Pack [107], IBM Bluemix [161] and Riemann). For example, the architects retrospectively analyse the network latency and fault rate obtained over a period of time by fault monitoring and diagnostic tools. Microservice architects then analyse these logs retrospectively for problems that can be resolved by adapting microservice granularity levels. As an example, if the logs indicate latency across the microservices, the architects might reason that this latency can limit the architecture's ability to scale and/or to meet its performance requirements. They can then decide to adapt the granularity (i.e. an exercise that may lead to re-structuring the functionalities of considered microservices by decomposing or merging them) to enable better scaling of the system.

On the academic front, there are approaches which aid reasoning about microservice granularity adaptation. However, these approaches either do not quantify the added value of granularity adaptation decisions (e.g., [126, 197]) and/or they do not consider the uncertainty related to the suitability of these decisions (e.g., [185]). Both shortcomings of the existing approaches can lead to cases where by the time the decision to adapt granularity is made, the architecture has potentially already suffered the negative consequences from an unsuitable granularity level. This can lead to reducing rather than introducing the added value of the microservice architecture. *There is a general lack of evaluation processes which quantify the added value of granularity adaptation and the fluctuation of that added value over time, so that adaptation decisions can be made as soon as the added value of the current granularity level becomes inadequate.*

In particular, we identify the following problem: *inadequacy of processes which evaluate the added value of granularity adaptation decisions under uncertainty.*

Although microservice monitoring tools have been used for retrospective analysis, logs from these tools can be used as useful input for evaluating the added value of granularity adaptation decisions. Such data can help as it provides a realistic account of the microservice architecture's behaviour under different runtime scenarios. Consider a fictional running microservice-based application with a functionality and scale similar to Netflix — called NetWatch. It receives around one billion streaming requests everyday [159]. Further consider that NetWatch's performance is being monitored at runtime using AWS Cloudwatch; it indicates a deterioration in NetWatch performance. One possible route in this scenario is to pursue granularity adaptation. The architect would assume that this improves NetWatch performance. However, it might be the case that keeping the current granularity level would be of more added value to NetWatch than the value of immediate performance improvement resulting from pursuing granularity adaptation. Nevertheless, this added value is uncertain because architects can be uncertain at design time about the likelihood of needing to integrate off-the-shelf products with NetWatch in the future for example. This uncertainty needs to be considered when calculating predictions of the added value related to pursuing granularity adaptation (or not). Other sources of include but are not limited to: fluctuations in the number of end users, in the message exchange across microservices, and/or in the availability of off-the-shelf products to integrate them.

Granularity adaptation by merging or decomposing microservices can be seen an exercise that can introduce an added value under uncertainty. Examples of added value include better balancing the load and preventing bottlenecks on microservices that may experience high loads. A counter argument may make a case for keeping the current granularity level unchanged if it proves to have more added value compared to the value which can be introduced by granularity adaptation. Central to answering these questions is to evaluate the added value of granularity adaptation under uncertainty.

The added value which can be enabled by granularity adaptation is dynamic; it varies depending on usage scenarios and context, fluctuations in workload, and level of QoS provision. Henceforth, the added value of granularity adaptation is a moving target and can be best tracked and analysed at runtime. Microservices are essentially designed for environments that are dynamic, scalable and exhibit changes in QoS

demands. As a result, it is difficult for software architects to predict such value at design time. Therefore, it is essential for the prediction to be updated at real time based on the observed runtime dynamic behaviour of the architecture.

This chapter aims at answering the following question: *"how can the added value of granularity adaptation be evaluated dynamically at runtime to determine when is it worth pursuing adaptation?"* The central theme of this chapter is that *granularity adaptation is only worthwhile if it will introduce added value to the architecture. The argument is that it is only worth adapting granularity of a microservice architecture if it may introduce added value to the architecture which enables it to better cope with the dynamism and uncertainties in microservice environments.*

The novel contribution of this chapter is two-fold. First, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem. Second, we contribute to *a novel evaluation process for dynamically evaluating the granularity adaptation design decisions under uncertainty at runtime*. Our process is way to evaluate added value through a novel hybrid combination of binomial real options theory [338] and the concept of Bayesian surprises [44]; it is customised for the context of microservice granularity adaptation. Our process dynamically determines at runtime if it would be worth adapting granularity from the perspective of introducing added value [51].

Unlike classical design time usage of binomial options analysis in software architectures [330, 248, 329], we apply the analysis at runtime and we extend the formulation to cater for runtime changes in the added value. Bayesian surprises quantify the divergence between uncertain beliefs (claims [44]) about a system's runtime behaviour prior and posterior to observing runtime evidence [44]. Our evaluation process uses Bayesian surprises [44] to enable binomial real options analysis to do real time updates of added value in response to runtime evidence variable values (e.g. workload variations).

We compare the usage of our novel evaluation process to the usage of AWS Cloudwatch to reason about granularity adaptation decisions retrospectively. This shows the unique benefit of our process: linking microservice granularity adaptation to its added value under uncertainty. On the methodological level, we demonstrate using

our AWS Lambda implementation of an online movie review microservice application — Filmflix — how our evaluation process goes beyond traditional application of real options by revealing dynamic trends related to the added value of granularity adaptation at runtime in response to variations in workload. Finally, we discuss the ability of our process to accommodate increases in the input provided to it.

In Section 4.2 we use a microservice application — Filmflix — to set the context for this chapter's contribution using the modelling primitives we contributed in Chapter 3. Section 4.3 presents our novel formulation of the granularity adaptation decision problem as a real options problem. In Section 4.4, we explain our usage of traditional binomial real options analysis [338] and Bayesian surprises [44]. We then describe our proposed evaluation process, highlighting how our novel combination of these concepts helps determine, at runtime, if it would be worth adapting the current microservice granularity level from an added value perspective. In Section 4.5 we present our evaluation of the contributed process using Filmflix. We further show the feasibility of our process to assist microservice adopters in industry by presenting a Java implementation of it in Section 4.5.4. In Section 4.6 we conclude by summarising the content of this chapter.

4.2 MOTIVATING SCENARIO

Consider that the architect of Filmflix defined alternative granularity adaptation strategies at design time using microservice ambients and granularity adaptation aspects from Chapter 3. We use this context to motivate the contribution of this chapter.

The initial architecture of Filmflix contains three microservices: ReviewRegulation — implementing the regulation of movie review, ReviewUpload — managing the user input requirements when uploading a review, and MovieReview — capturing input from the user through an interface. We consider that the microservitization in this application is driven by enhancing the performance of Filmflix where the expected runtime workload (i.e. number of reviews sent to Filmflix) is high.

The utility of concern is Filmflix performance in response to workload fluctuation. Therefore, the invocation duration of a microservice (i.e. time elapsed from when the microservice code starts executing as a result of an invocation to when it stops executing [353]) and the number of reviews received by Filmflix are useful variables to observe at runtime and use them to inform the evaluation of granularity adaptation decisions. The expected high workload calls for a strict threshold on invocation duration to ensure performance provision and avoid losing users to competing applications. Furthermore, the potential cost of adapting the granularity level of Filmflix microservices is estimated using a microservice-specific cost model called CostHat [196] (which we explain in Section 4.4 and use in our evaluation).

At runtime, each microservice's invocation duration is monitored using AWS Cloudwatch. Consider a situation where MovieReview's invocation duration deteriorates beyond the pre-defined threshold. Filmflix architects then are faced with a choice between adapting the granularity level of MovieReview to get the invocation duration within the pre-defined threshold or not adapting to avoid the adaptation cost.

To decide between pursuing granularity adaptation or not, the potential added value of adapting granularity needs to be compared against the adaptation cost. However, the current state-of-the-practice (i.e. using tools such as AWS Cloudwatch) do not provide ready means and built-in mechanisms to calculate the potential added value of adapting granularity.

Due to dynamism of the runtime environment in which Filmflix is operating, potential added value calculations need to account for this dynamism. For example, the added value of adapting granularity depends on the likelihood of Filmflix experiencing a large volume of reviews. Intuitively, Filmflix architects strive for maintaining acceptable end user experience during peak times. Therefore, if there is a high likelihood that Filmflix will be used by a large number of end users then there is high potential added value for adapting granularity to improve MovieReview performance.

Moreover, this likelihood itself can fluctuate at runtime when the number of end users is observed rather than predicted. Consequently, the potential added value calculations needed to updated at runtime to reflect these likelihood fluctuations. For

example, if the observed number of end users is as high as expected, then there is higher likelihood that there is a high potential added value for granularity adaptation.

4.3 PROBLEM FORMULATION

We can infer the following characteristics of the problem context which the contribution in this chapter is addressing. These characteristics justify formulating the decision problem when it is worth pursuing granularity adaptation in a value-driven manner:

- Granularity adaptation can introduce an opportunity to enhance a microservitization utility (e.g., MovieReview performance). Architects have the right without a symmetric obligation to reap benefits of opportunities enabled by granularity adaptation.
- The value of the architectural potentials enabled by granularity adaptation is uncertain due to fluctuations in the microservice runtime environment (e.g., in workload volumes).
- Uncertainty regarding the value of architectural potentials can be updated given runtime observations of the microservice architecture and its runtime environment.

Given these characteristics, we infer the problem formulation of the core decision problem in this chapter. In particular, answering the question of when it is worth pursuing granularity adaptation entails answering the following questions:

- What is the potential added value of pursuing granularity adaptation (related to the first point above)?
- How can uncertainty regarding this potential added value be captured (related to the second point above)?
- How can this uncertainty be updated in the face of runtime observation of the microservice architecture and its running environment (related to the third point above)?

We formulate the problem of evaluating the worth of pursuing granularity adaptation as a real options problem. A real option is the right but not the obligation to take an investment decision under uncertainty [227]. In the context of software architecture different types of real options have been utilised including the option to switch, defer, expand or stage investments [248, 329].

Among the examples of real options is that studied by Baldwin and Clark's theory [36]; it studies the real options embedded with a modular software architecture design. The theory defines a model for reasoning about the value added to a base system by modularising it. The theory originated from studying the impact of modularised design on structure of a computing industry. In particular, the study revealed that "modularity in computer designs caused the industry to evolve from its initial concentrated structure to a highly dispersed structure. Modularity allows design tasks to be divided among groups that can work independently, and do not have to be parts of the same firm." Consequently, this theory aims to study how modularity can be attained in software architectures, how this can enable modular design evolution (i.e. create real options) and how economic incentives in each module can be exercised autonomously (i.e. exercising the real options). A particularly relevant type of real options to our case is a call option — it is related to buying a stock, bond, commodity or other instrument at an exercise price at any point within a specific time period before the option expires (according to the American view of options). In particular, Baldwin and Clark's theory aims to map and quantify the option value of modular software architecture designs. This theory has been applied in multiple complex system contexts (e.g., [305, 365, 321, 303, 210])

In the context of Baldwin and Clark's theory, modularising a design entails using operators (described in Table 4.1) to change the design of an architecture into modules according to a formal plan and embed real options in the architecture through the exercise. Granularity adaptation has been specifically used in the context of microservices to mean merging or splitting functionalities into microservices according to a systematic granularity adaptation strategy [156]. Granularity adaptation can therefore be seen as a generalisation of the modularisation exercise. In other

words, a module can be seen as equivalent to a fine-grained microservice resulting from a decomposition exercise.

	Baldwin and Clark theory regarding modularisation	Pursuing granularity adaptation
Concept Name	Concept Comparison	
Exercise Operations	Modularisation operations include: • Splitting: serves to modularize proto-modular designs. • Augmentation: adds a module to a system, • Substitution: replaces a module, • Exclusion: removes a module, • Inversion: standardizes a common design element, • Porting: transports a module for use in another system.	The initial execution of merging microservices is analogous to augmentation, substitution and exclusion of modules. In particular, merging microservices can be seen as an aggregate sequence of the aforementioned operators. Decomposing microservices on the other hand is analogous to splitting microservices.
Exercise Problem Context	Some of the modules are "hidden", meaning that design decisions in those modules do not affect decisions in other modules; some of the modules are "visible," meaning that they embody "design rules" that hidden-module designers must obey if the modules are to work together.	Some microservices do not have any logical dependencies (i.e. they do not depend on each other in an execution flow) on the remaining architecture while others have strong dependencies on other microservices in the architecture.
Real Options Benefits	Modular designs offer alternatives (i.e. real options) that non-modular ("interdependent") designs do not provide. Specifically, in the hidden modules, designers may replace early, inferior solutions with later, superior solutions. Such alternatives can be modelled as "real options".	A suitable granularity level enables opportunities to enhance a microservitization utility. These opportunities can be reaped later in the architectures' life. We envision that these opportunities can be modelled as real options.

Types of Real Options	Types of real options that can be introduced by modularisation include [338, 97]: • Defer decisions to invest until optimal to do so • Default on a project that is structured in phases or abandon a project for its salvage value • Expand or contract production if favorable or unfavorable conditions emerge • Switch materials used in a product, or switch to producing another product as supply and demand conditions change • Invest in a market should it become lucrative to do so (growth options) • Options on options (used to model phased investments)	Types of real options that can be introduced by granularity adaptation include: • Defer decisions to invest in a new microservice product until it is optimal to do so • Discard a microservice(s) when they are deemed of no value to the architecture • Scale the architecture when necessary at runtime • Replace a currently used off-the-shelf product with a superior one when it becomes available • Expand the architecture to enter a newly emerging market or enhance the QoS provision for a particularly attractive market • Exercising an option can create further options along any of the types above.
Calculating Real Options	The value of a real option is calculated using 1) its "technical potential" (which in itself is uncertain), 2) the cost of mounting independent design experiments and, 3) the "visibility" of the module in question.	The value of a real option can be calculated using a dynamic evaluation process which is based on a novel hybrid combination of binomial real options analysis and Bayesian surprises. This process tracks and updates fluctuations in the predicted added value given runtime observations of the dynamic microservice environments. Our process also accounts for the cost of pursuing granularity adaptation. The same procedure of our process is followed to dynamically evaluate the potential added value of keeping the currently granularity level unchanged.

Exercise Justification	A positive real option value in the calculation above justifies investment in a new modular architecture.	If the real option value of pursuing granularity adaptation is higher than that of keeping the granularity level unchanged even after tracking and updating runtime fluctuations in these values then the investment in pursuing granularity adaptation is justified.

Table 4.1 Mapping between Baldwin and Clark's view of modularisation and our view of microservice granularity adaptation

Though both exercises exhibit resemblance, in our work we view the fundamental differences between them to be: 1) the relatively more dynamic environment in which granularity adaptation takes place compared to modularisation and 2) modularisation is driven solely by enhancing modularity while granularity adaptation can be driven by enhancing other microservitization utilities (e.g., autonomy, replaceability and/or QoS provision). However, they both agree on the notion of encapsulating functionalities within boundaries according to a particular driver.

Henceforth, we use inspiration from the theory of Baldwin and Clark in our context. The inspiration is along two dimensions: 1) we formulate the granularity adaptation decision problem as a real option problem and, 2) we evaluate the added value of real options related to pursuing granularity adaptation. However, our approach to evaluation is different from Baldwin and Clark. Firstly, our work is the first to transit binomial real options analysis to runtime (addressing question 3 above) as opposed to static design-time traditional usage of this theory. Bayesian surprises [44] enable binomial real options analysis to update added value predictions at runtime in response to runtime evidence variable values (e.g. workload variations). Secondly, granularity adaptation can be evaluated relative to the drivers of the exercise and to the added value of the architectural utility; the traditional usage of Baldwin and Clark's theory focusses on modularity as the architectural utility of concern. Table 4.1 maps the principles and describes the analogy of Baldwin and Clark's theory to our granularity adaptation problem.

4.4 DYNAMIC EVALUATION OF MICROSERVICE GRANULARITY ADAPTATION

Sections 4.4.1 and 4.4.2 explain our utilisation of traditional binomial real options analysis [338] and Bayesian surprises [44]. Section 4.4.3 then describes our proposed evaluation process which is a novel combination of the aforementioned theories customised for the context of microservice granularity adaptation. Our process dynamically evaluates at runtime the added value of pursuing granularity adaptation versus keeping the current granularity level unchanged. Our process is iterative; it evaluates the added values then suggests pursuing granularity adaptation or not at the end of each iteration.

4.4.1 Utilising Traditional Binomial Real Options Analysis

There are several models for evaluating real options (e.g., Black and Scholes [77], Monte-Carlo methods [136], and finite difference methods [310]). In our work, we use binomial real option analysis evaluation. Our choice of binomial real options analysis is due to the flexibility it provides in using the architects' estimates for potential rises or falls in the value of the architectural utility over time given a certain granularity level. Additionally, this analysis accounts for the probabilities of these rises/falls. Binomial trees are effective in visualising the value in scenarios where architects' prediction can serve as a sensible input to the evaluation. However, the estimates can suffer from under- or over-estimation due to the dynamic behaviour of architecture and/or the architects' imperfect knowledge of the highly dynamic microservice environment. Traditional use of binomial real options in software engineering (e.g. [248]) relies on the architects' expectations or publicly available cross-company data to inform real option evaluation. These sources can provide insights regarding potential improvement of architectural utilities due to an investment in the architecture. We capture these insights in the form of utility trees.

Suppose that the architect is interested in valuing the real options which can be embedded by adapting or retained by not adapting granularity. In our work, this is done using two binomial trees. One binomial tree is used to evaluate the potential

added value which can be embedded by adapting granularity in terms of real options (e.g., to scale the architecture). Similarly, another binomial tree is used to evaluate the potential added value retained by not adapting granularity in terms of real options. In both cases, added value quantifies the difference between potential benefit (regarding microservitization utilities of concern) and cost of pursuing granularity adaptation (or not).

For example, the bottom parts of the cells (*ROVs*) in Figure 4.2 show the real option values at different timestamps. They represent the value of embedded opportunity alone while the top parts of the cells represent the overall architectural value if the granularity is adapted or not in the corresponding timestamp. The top parts of each cell in the binomial tree (Figure 4.2) show the overall architectural value (*AV*) at different points in time. Each *AV* quantifies the potential value of the overall architecture including the utility in the corresponding utility tree cell. Figure 4.1 shows the likely improvement to the overall architecture's utility upon investing in a design decision (either adapting or not adapting granularity) that delivers on performance improvement by minimising invocation duration.

The output of a binomial real options analysis tree is the bottom part of cell A in Figure 4.2. ROV_0 represents the present real option value if it were to be exercised once at timestamp F_t captured in the tree. This output is produced at the end of two stages in Figure 4.2: 1) the forward pass, which involves calculating the *AVs* starting from cell A then, 2) the backward pass, which involves calculating *ROVs* starting from cells B and C then moving backward.

We interchangeably refer to the timestamp in which an option is exercised as the last timestamp of a binomial tree (F_t) or the option exercising timestamp (*et*).

In the following paragraphs, we explain each component of the binomial trees in more detail.

Architectural value (AV): Traditionally architectural value is likened to a financial stock's value. Architectural value quantifies the potential value of the overall architecture when an improvement in architectural utility (e.g., QoS provision) [248] would be obtained by granularity adaptation or retained by keeping the current level of granularity. Each architectural value (AV_t) in each timestamp t is calculated using

Eq. (4.1) in the first stage — the forward pass in Figure 4.2 — of constructing the tree.

$$AV_t = Value_0 + V_{Tree} \tag{4.1}$$

where $Value_0$ is the initial architectural value once the initial microservice architecture is deployed. V_{Tree} is the value V of the cell in utility tree *Tree* corresponding to binomial tree cell for which AV_t is calculated. In our work, we use the same definition and formula for architectural value as that used traditionally in software engineering.

Real option value rise rate (R_{ROV}) or fall rate (F_{ROV}): This refers to the expected rate of real option value rise or fall at timestamp t with respect to timestamp *t-1* [20, 248]. It corresponds to the improvement/deterioration in the architectural value at timestamp t with respect to timestamp *t-1*, calculated as follows:

$$R_{ROV} = AV_{raised_t}/AV_{t-1} \tag{4.2}$$

$$F_{ROV} = AV_{reduced_t}/AV_{t-1} \tag{4.3}$$

where AV_{raised_t} is the raised architectural value AV for current timestamp t. AV_{t-1} is the architectural value at the previous timestamp *t-1*. $AV_{reduced_t}$ is the reduced architectural value AV for current timestamp t. In our work, we use the same definition and formulas for real option value rise/fall rates.

Probability of real option value rise $P(Rise_{ROV})$ / $P(Fall_{ROV})$: This is a risk-neutral probability of real option value rise/fall, calculated based on Eq. (4.4) and Eq. (4.5) respectively. The calculation includes a pre-defined risk-free interest rate *IR* representing the expected fluctuations in demand for the system [248].

$$P(Rise_{ROV}) = \frac{(1+IR)}{(R_{ROV} - F_{ROV}) * (100 - F_{ROV})} \tag{4.4}$$

$$P(Fall_{ROV}) = 1 - P(Rise_{ROV}) \tag{4.5}$$

Real option value (ROV): The real option value quantifies the potential added value if the option is exercised. The real option values in a single binomial analysis tree are calculated using the equations below in the second stage — backward pass

in Figure 4.2 — of constructing the tree. The real option value (ROV_{et}) in the final timestamp/option exercising timestamp *et* of a tree is calculated using Eq. 4.6. All ROV_t where $t \neq et$ are calculated using Eq. (4.7).

$$ROV_{et} = MAX(0, AV_{F_t} - MC) \tag{4.6}$$

$$ROV_t = \frac{ROV_{raised_{t+1}} * P(Rise_{ROV_t})}{100} + \frac{ROV_{reduced_{t+1}} * P(Fall_{ROV_t})}{100} * \frac{(1+IR)}{100} \tag{4.7}$$

where AV_{F_t} is the architectural value at F_t and MC is the estimated maintenance cost of the architecture. $ROV_{raised_{t+1}}$ is the raised and $ROV_{reduced_{t+1}}$ is the reduced real option value at the following option exercising timestamp *t+1* of the backward pass.

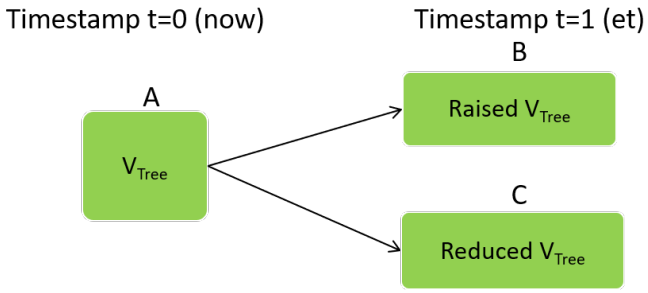


Fig. 4.1 Example utility tree related to an architecture; cells A,B and C have the utility value (V_{Tree}) (e.g., the utility resulting from improving the invocation duration of an architecture by merging microservices together) used to calculate AVs of the respective cells A,B and C in Figure 4.2

It is worth noting that we regard maintenance cost to include the activities required for the health of a running architecture (be that the adapted or un-adapted one) rather than those required for adapting its granularity [61, 248]. These include but are not limited to development, configuration, securing communication links and implementing data translation layers. Our process is flexible, making it possible to integrate coarse-grained cost models (e.g. CostHat [196]) or finer-grained ones if historical data is available (e.g., expert judgements,cross-company data, what-if analysis using a simulated ambient environment).

The maintenance cost estimation is assumed to be fixed unless whenever further data becomes available which can refine the accuracy of the estimation. In this case

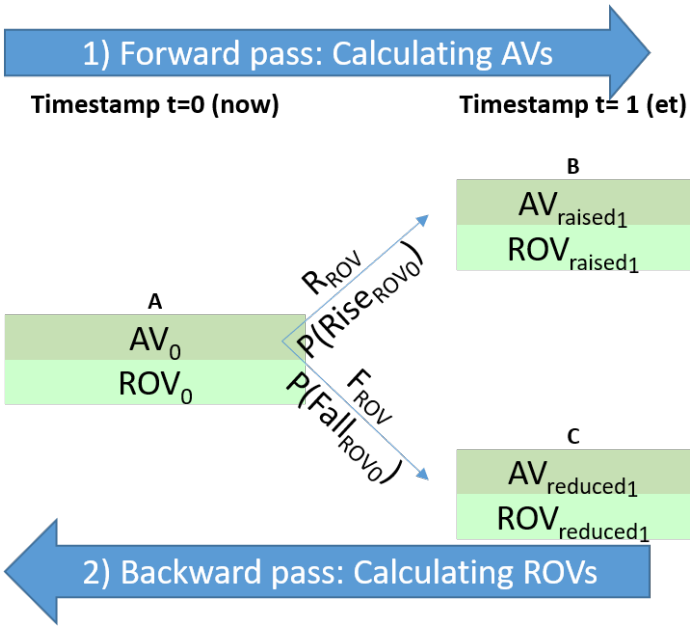


Fig. 4.2 Example traditional binomial real options analysis tree calculating the architectural values (AVs) (e.g., the overall value of a microservice architecture where the granularity is adapted by merging microservices together to improve the application's invocation duration) and real option values (ROVs) for a single real option embedded in the architecture (e.g., a real option to scale the merged architecture not available) it is exercised at timestamp 1. The ROV rises at rate R_{ROV} with probability $P(Rise_{ROV_0})$ or falls at rate F_{ROV} with probability $P(Fall_{ROV_0})$

we assume manual updates of maintenance cost; the dynamism of the microservice operating environment has much higher impact on real option values than on the maintenance costs [196].

At the end of the backward pass, the bottom part of the base cell in the tree (cell A in Figure 4.2) shows the present real option value if the option were to be exercised only at the last timestamp F_t captured in the tree. However, as the potential value of exercising an option is time sensitive, we have assumed the American view of options in our process. This means the option can be exercised at any point before it expires [20]. We assume that exercising an option can enable further options given the dynamism of microservice environments. So, it is desirable to consider the real options to be compound (i.e. exercising one option can further enable a package of distinct options [163]). For that, the total real option value ROV_{total} is calculated based on Eq. (4.8):

$$ROV_{total} = (\sum_{i=0}^{i=F_t} ROV_{t=0_i}) - RC \tag{4.8}$$

where i is a counter for the intermediate binomial trees, ranging from 0 to the final timestamp F_t and RC is the re-structuring cost — the cost of pursuing granularity adaptation. This cost includes but is not limited to taking the microservices of concern offline, then encoding and enforcing a chosen granularity adaptation strategy and finally re-deploying the adapted architecture. The value of RC is zero in the case of not adapting. Eq. (4.8) adds the real option value in the base cells ($ROV_{t=0_i}$) of multiple binomial trees (each capturing a different range of timestamps) then deducts the re-structuring cost. It is worth noting that our process is flexible allowing using any cost model to estimate the re-structuring cost (e.g., expert judgement and/or cross-company-data).

The upper bound of the maximum range covered by intermediate trees (i.e. F_t in Eq. 4.8) corresponds to the real option's expiry date, after which the real option (e.g. scaling the architecture by replication across rented application servers) becomes unavailable to exercise (due to the application server's lease end for example). $ROV_{t=0_i}$ is the real option value in the base cell t=0 of the i^{th} intermediate binomial tree.

In order to make a decision on whether or not it is worth adapting the granularity (i.e. analogous to buying a call option), the architect can compare ROV_{total} of the real options in the adapted and un-adapted architectures at runtime. If adapting has larger ROV_{total}, it is worth adapting the granularity. Section 4.4.3 shows how to combine real options analysis with Bayesian surprises, so that ROV_{total} can be adapted at real time in response to runtime evidence.

4.4.2 Utilising Bayesian Surprises

Fluctuations in QoS provision at runtime tend to affect the utility of the system which can impact added value. Therefore, estimates of rise/fall probabilities in real option values based on expert judgements or cross-company data for example can suffer from under- or over-estimation due to the dynamic behaviour of architecture and/or the architects' imperfect knowledge of the highly dynamic microservice environment. Therefore, instead of taking a fixed estimation for the probability of rise/fall in real option values, we continuously update these probabilities at runtime making use of runtime observations of QoS provision and fluctuations of workload.

Bayesian surprises quantify the divergence between beliefs (claims [44]) about the system's behaviour prior and posterior to observing runtime evidence [44]. No divergence means the belief is verified leading to zero Bayesian surprise. The larger the magnitude of the surprise the more evidence there is regarding the belief violation. We use Bayesian surprises to quantify the significance of runtime evidence in updating beliefs about the system's likely runtime behaviour. We then use Bayesian surprises to update the probabilities of rise/fall in real option values. Our work is the first to combine Bayesian surprises with real options analysis and the first to apply Bayesian surprises in the context of microservice granularity adaptation.

In this work, we consider a claim that articulates uncertain beliefs in the context of end user workloads. Assume the scenario where Netflix receives around one billion streaming requests everyday [159]. In this context the end users are Netflix subscribers, the workload is the amount of streaming requests and the claim articulates the architects' belief about how much the QoS provision of the overall architecture is affected by fluctuations in the streaming workload. A claim capturing this can be articulated as follows:

- If end user workloads are greater than a pre-defined workload threshold then the QoS provision always becomes worse than a QoS threshold and,
- If QoS provision is worse than a QoS threshold, this is always caused by the end user workload exceeding the pre-defined workload threshold.

Other claims can also be articulated assessing the impact of other contexts (e.g. organisational changes) on QoS provision levels.

When the binomial trees are initially constructed in our process, they are constructed assuming a claim holds true. However, this claim can be assessed at runtime; if it is violated this calls for updating the trees at runtime since behaviour of the architecture is not as initially expected. Thereby the values which can be enabled by adapting granularity or not (which are captured in the binomial trees) are not as initially believed. We acknowledge that this route could be supplemented in future research with updating the claim thresholds to reflect the runtime evidence on the

architects' beliefs. In the following paragraphs, we explain several concepts necessary to understand how Bayesian surprises work.

Runtime evidence variables (REV): These are variables monitored at runtime; their values indicate violation or verification of claims. Considering the example claim above, the runtime evidence variables are pairs of each end user workload and corresponding QoS attribute value recorded at runtime. These pairs are chosen to observe whether the architecture meets the pre-defined thresholds articulated in the claim or not at runtime.

Claim prior probability P(Claim): This is the probability of a claim to hold true before observing runtime evidence variables [44]. For example, there is no means of ensuring at design time whether QoS provision will indeed be within the pre-defined threshold and the workload will be as articulated in the claim. Therefore before a microservice architecture is deployed, the claim prior probability holding true is assumed to be 0.5. Since there is no runtime evidence observed, there is an equal likelihood of the claim holding true or being violated at runtime. Thereafter, the prior probability of the claim is carried over at runtime before observing new runtime evidence variable values.

Claim posterior probability P(Claim | REV): This is the probability of the claim to hold true assessed in the light of observing relevant evidence *(REV)*. In our process, the claim posterior probability is only different from the prior probability if the runtime evidence violates the claim. Our process is flexible regarding the method used to estimate the posterior probability. Crucially though, this method has to capture the dependence between runtime end user workload and QoS recorded values (e.g. by using Bayesian networks [290]) rather than assuming independence between them (e.g. Naives Bayes [290]). In context of the example claim above, QoS provision depends on the workload, and the claim holding true in turn depends on both.

Bayesian surprise (S_{claim}): This is the divergence between the claim prior and posterior probabilities, calculated using Eq. 4.9 [44]:

$$S_{claim} = P(Claim|REV) * \log \frac{P(Claim|REV)}{P(Claim)} \tag{4.9}$$

Bayesian surprises only measure the magnitude of the divergence given runtime evidence variable values (*REV* in Eq. 4.9) regardless of the direction (good or nasty surprise). The larger the magnitude of S_{claim}, the larger the update regarding beliefs (articulated in the form of claims) about the system behaviour. S_{claim} can take continuous values starting from zero.

In this chapter, we elicit this direction by comparing the trends of QoS attribute values against end user workload values. A good surprise occurs if the QoS attribute values improve over time even though the end user workload is above the pre-defined workload threshold. A good surprise promotes keeping the current microservice granularity level. For example, if the QoS provision is within the QoS threshold articulated in the claim when the workload is beyond the respective threshold in the same claim, this is a good surprise. In other words, the microservice(s) is behaving better than expected, calling for not adapting. Therefore if S_{claim} is good, it is used to update the binomial tree of not adapting by increasing the probability of ROV rise in that tree by a certain fraction explained later in this section. All the probabilities of ROV rise across a binomial tree are updated by the same fraction.

A nasty surprise occurs if the QoS attribute values deteriorate over time even though the end user workload is below the pre-defined workload threshold. Intuitively, a nasty surprise promotes granularity adaptation. For example, if the QoS provision is beyond the QoS threshold articulated in the claim when the workload is within the respective threshold of the same claim, this is a nasty surprise. In other words, the microservice(s) is behaving worse than expected, calling for granularity adaptation. Therefore if S_{claim} is nasty, it is used to update the binomial tree of adapting by increasing the probability of ROV rise in that tree by a certain fraction explained later in this section. All the probabilities of ROV rise across a binomial tree are updated by the same fraction.

Bayesian surprise tolerance threshold — in percentage (T_S), in corresponding exact value (T_s^{exact}): In[149], a surprise tolerance threshold is pre-defined for every claim to indicate the criticality of the QoS attributes captured in it to the software architects. Only if the Bayesian surprise triggered on the claim exceeds this threshold will action be taken. In our context, this action is updating the probability of option

rise for granularity adaptation or for keeping the current architecture at runtime. For every claim, this tolerance threshold is defined at design-time as a percentage by the software architect (T_S in Eq. (4.10)). The higher this percentage the less critical the QoS attribute is to the software architects. The exact value (T_s^{exact} in Eq. (4.10)) to which this percentage corresponds is calculated during every runtime iteration of our process given a record of the Bayesian surprises triggered in the previous runtime iterations using Eq. 4.10.

$$T_S^{exact} = T_S * (S_{claim_{max}} - S_{claim_{min}}) \tag{4.10}$$

where $S_{claim_{max}}$ is the maximum triggered Bayesian surprise on the claim so far and $S_{claim_{min}}$ is the minimum triggered Bayesian surprise on the claim so far. $T_{S_{exact}}$ always gets tighter as more runtime evidence is acquired. The larger the range of triggered surprises, the larger the difference between one T_s^{exact} and the next.

4.4.3 Proposed Dynamic Evaluation Process

Our novel process determines whether or not it is worth pursuing adaptation at each runtime iteration of our process. We contribute to a novel evaluation process for dynamically evaluating the worth of pursuing granularity adaptation. It uses the runtime evidence to update at real time the potential added value of pursuing granularity adaptation. It extends traditional real options analysis, thereby being the first contribution which enables using real options analysis at runtime. Bayesian surprises are used to update the probabilities of real option rise or fall in traditional binomial trees. This enriches traditional binomial real option analysis with the impact of runtime evidence on the real option values. In our process, the added value calculation continuously calibrates evidence variables observed over runtime iterations.

4.4.3.1 Inputs

We formulate the inputs of our process benefiting from the concepts described in Sections 4.4.1 and 4.4.2. Our inputs capture the potential likely cost and benefit as

well as runtime dynamics related to adapting and not adapting granularity. In addition to the inputs below, the running microservices to be reasoned about are also inputs to our process. Our process is flexible regarding the scope of microservices input to it; the input could be a single microservice or multiple interdependent microservices. It is worth noting that this scope determines the targets of the inputs below. For example, maintenance and re-structuring costs are estimated for the overall pre-defined scope of concern. The same reasoning applies to the inputs related to calculating Bayesian surprises. The pre-defined scope is monitored at runtime for QoSs of concern. Where the scope entails multiple microservices the overall QoS is monitored at runtime rather than the individual microservices.

Inputs for binomial tree construction (see Section 4.4.1):

1. *Re-structuring cost.*
2. *Maintenance costs for the adapted and un-adapted architectures.*
3. *Initial architectural value of the adapted and un-adapted architecture.*
4. *Utility trees for adapting and not adapting.*
5. *Option expiry date:* the predicted number of binomial tree timestamps (in unit days, months or years) corresponding to the option expiry date. The same numeric value is used for the un-adapted and adapted architectures for fair comparison of the total real option values.

Inputs for claim articulation (see Section 4.4.2):

1. *Runtime evidence variables*: The QoS attribute indicated by the architects to be monitored at runtime to assess the validity of the claim. Without loss of generality, we assume that larger values indicate worse QoS. We consider the end user workload to be the other runtime evidence variable to be recorded at runtime along with the indicated QoS attribute. End user workload will measure the runtime end user interest in the QoS attribute indicated by the software architect.
2. *QoS attribute threshold (this could be any attribute of concern to a given microservice architecture),*

3. *End user workload threshold.*

Inputs for Bayesian surprise calculation (see Section 4.4.2):

1. *Claim prior probability,*

2. *Bayesian surprise tolerance threshold.*

The architects provide the inputs required to carry out only the first iteration of our process. As long as granularity adaptation is not pursued across iterations, the input of the previous iteration is carried over as input to the next one. Otherwise, new input is needed to conduct our process on the newly deployed microservice(s).

4.4.3.2 Process

We contribute to a novel evaluation process for dynamically evaluating the worth of pursuing granularity adaptation. In every runtime iteration of our process, it solicits and monitors runtime evidence then feeds it into calculating and updating added value. Pseudocode — AddedValueUpdate — presents our proposed evaluation process and Table 4.2 defines the acronyms used in the pseudocode. Initially, AddedValueUpdate

Acronym	Term
RC	Re-structuring cost of granularity adaptation
MCs	Maintenance costs for the adapted and un-adapted architectures
$Value_0$	Initial architectural value
Trees	Utility trees for adapting and not adapting
F_t	Final timestamp of a single tree (or overall option expiry date)
$Q_{ri\ name}$	QoS attribute name at runtime iteration ri
T_w	Workload threshold
T_q	QoS attribute threshold
P_{claim}	Claim prior probability
T_s	Bayesian surprise tolerance threshold
t	Binomial tree timestamp counter
ri	Runtime iteration counter
Q_{ri}	The value of $Q_{ri\ name}$
W_{ri}	End user workload value at runtime iteration ri
$ROV_{total}^{adapting}$	The total real option value for adapting
$ROV_{total}^{not\ adapting}$	The total real option value for not adapting
$Suggestion_{ri}$	Final verdict of the pseudocode for pursuing granularity adaptation in ri

Table 4.2 AddedValueUpdate acronym definitions

assumes the software architects formulate their belief about the system's runtime behaviour in the form of a claim articulated as Line 3. A claim is articulated using

Algorithm 1 The pseudocode presenting our proposed process leveraging upon [248] and [44]; Table 4.2 defines the acronyms used in the pseudocode

1: **procedure** ADDEDVALUEUPDATE
2: **Input:** *RC, MCs Value$_0$,Trees,F_t, $Q_{ri\ name}$, T_w, T_q, P_{claim}, T_s*
3: Claim: [$W_{ri} > T_w \leftrightarrow Q_{ri} > T_q$]
4: **for** each ri **do**
5: **if** ri=0 **then**
6: Construct Binomial Real Options Trees for Adapting and Not Adapting using Value$_0$, Trees, F_t, MCs
7: **else**
8: Carry Binomial Real Options Trees for Adapting and Not Adapting from ri-1
9: **end if**
10: Solicit and Monitor Q_{ri} and W_{ri}
11: Calculate S_{claim} and $T_{s\ exact}$
12: **if** S_{claim}=0 OR ($S_{claim} < T_{s\ exact}$) **then**
13: Go to next iteration
14: **else if** ($Q_{ri} > Q_{ri-1}$) AND $W < T_w$) **then** ▷ *nasty surprise*
15: P($Rise_{ROV}^{adapting}$) * S_{claim}/T_s^{exact}
16: **else if** ($Q_{ri} < Q_{ri-1}$) AND $W > T_w$) **then** ▷ *good surprise*
17: P($Rise_{ROV}^{not\ adapting}$) * S_{claim}/T_s^{exact}
18: **end if**
19: Re-calculate $ROV_{total}^{adapting}$ and $ROV_{total}^{not\ adapting}$
20: **if** $ROV_{total}^{adapting} > ROV_{total}^{not\ adapting}$ **then**
21: $Suggestion_{ri}$=Yes
22: **else**
23: $Suggestion_{ri}$=No
24: **end if**
25: **Output:** $ROV_{total}^{adapting}$, $ROV_{total}^{notadapting}$, $Suggestion_{ri}$
26: **end for**
27: **end procedure**

an input end user workload threshold (T_q) and a corresponding input QoS attribute threshold (T_q). For example, consider the following claim: "workload $>$ 100 requests $\leftrightarrow$ Invocation duration $>$ 1 millisecond". This claim is read in two ways:

- If a microservice(s) receives more than 100 parallel requests, then the invocation duration will always be more than 1 millisecond and,
- If the invocation duration of the microservice(s) is more than 1 millisecond, this is always due to the number of parallel requests exceeding 100.

The claim is assessed and subsequent analysis is carried out in each runtime iteration (Line 4). The length of time between runtime iterations determines the rate of soliciting runtime evidence, the rate of updating added values, when adaptation is pursued. This length depends on the criticality of the application and/or runtime scenario. For

example, when Filmflix is operating during holiday seasons under high workloads then even a short time with poor QoS could lead to serious consequences.

Therefore, the rate at which this QoS needs to be monitored is high and thereby the length of time between runtime iterations needs to be in the order of milliseconds. It is worth noting that the length between runtime iterations is different from the option exercising timestamp. The option exercising timestamp is a point in time (not duration) when the option is exercised. For example, consider a real option to add new functionality in Filmflix. The point in time when this new functionality is added (i.e. the real option is exercised) is the option exercising timestamp.

In each runtime iteration, binomial real options analysis trees are first constructed to calculate the total potential real option value which can be enabled adapting and retained by not adapting granularity using the inputs and equations explained in Section 4.4.1. For the first iteration, the calculation is done using traditional binomial real options analysis (Line 6). For each following iterations, no manual construction is required. Instead, the most-up-to-date binomial trees are carried over from the end of previous iteration (Line 8). Then the quality Q_{ri} and workload W_{ri} values are recorded (Line 10) and analysed to update probabilities of option rise in the binomial trees if need be (Lines 14 to 18).

The analysis is done by calculating Bayesian surprise S_{claim} regarding the claim and the exact Bayesian surprise tolerance threshold T_s^{exact} (Line 11). No probability update is required if the Bayesian surprise is zero (the claim is verified) or below T_s^{exact} (Line 12). Otherwise, option value rise probability updates will be performed. Using the same claim example above, if the invocation duration is 1.05 milliseconds when there are 101 parallel requests, the claim is violated. Therefore, S_{claim} is greater than zero in this iteration. However, if T_s^{exact} is greater than S_{claim}, then this violation of the claim is not significant enough to have an impact on the adaptation decision.

Intuitively, a nasty surprise promotes granularity adaptation. For example, if the invocation duration is 2.5 milliseconds when there are only 75 parallel requests, this is a nasty surprise. In other words, the microservice(s) is behaving worse than expected, calling for granularity adaptation. Therefore, the probability of option rise in adapting granularity is increased if a nasty surprise is encountered (Line 15).

The factor by which the probability is increased depends on the significance of the Bayesian surprise. A more significant Bayesian surprise is further away from the surprise tolerance threshold than a less significant surprise. For example, if the invocation duration is 10 milliseconds for 50 parallel requests, this is a more major claim violation than if the invocation duration is 1.75 milliseconds for 95 parallel requests. We reflect this significance as a fraction of the calculated Bayesian surprise over the exact surprise tolerance threshold (S_{claim} / T_s^{exact}).

Conversely, good surprise promotes keeping the current microservice granularity level. For example, if the invocation duration is 0.8 milliseconds when there are 110 requests, this is a good surprise. In other words, the microservice(s) is behaving better than expected, calling for not adapting. Therefore, the probability of option rise in not adapting granularity is increased by the (S_{claim} / T_s^{exact}) factor if a good surprise is encountered. Once the probabilities of option rise and fall are updated according to the direction of the Bayesian surprise, the updated total real option values are calculated which correspond to adapting and not adapting (Line 19).

4.4.3.3 Outputs

At the end of each runtime iteration, the pseudocode suggests adapting granularity (Line 21) or keeping the current architecture (Line 23). The output is presented as a final verdict regarding pursuing granularity adaptation ($Suggestion_{ri}$) and the total real option values used to arrive at this verdict (Line 25). Following the process's suggestion in this case is a cautious approach to avoid unjustified adaptation that will not introduce added value to the architecture but rather incur unnecessary re-structuring and maintenance costs. In this case, all the inputs to the current runtime iteration are carried over to the next one.

On the other hand, if the total real option values corresponding to adapting are greater than not adapting (Line 20), the pseudocode outputs a suggestion to pursue adaptation in runtime iteration *ri* (Line 21) along with the total real option values used to arrive at this verdict (Line 25). Following the process's suggestion to pursue adaptation in this case is a cautious approach to introduce the added value in the real

options rather than taking a reactive route which could result in missing opportunities for economic gains.

It is worth discussing the scalability of our evaluation process with respect to increases in inputs to it. In particular, it is worth discussing the extent to which our evaluation process can accommodate an increase in the number of assessed claims (i.e. in the number of QoS attribute and runtime evidence variable pairs). For a microservice with q QoS attributes of concern and r runtime evidence variables of concern, there are $q*r$ claims to violate or verify at runtime using $q*r$ iterations of AddedValueUpdate. However, goal-oriented processes can accurately map high-level goals (i.e. QoSs) to their measurable metrics (i.e. runtime evidence variables) [343, 88, 22, 23, 175]. Therefore, it is seldom the case that q QoS attributes and r runtime evidence variables actually require $q*r$ claims to reason about at runtime. Therefore to answer EQ3, we can envision that the ability of our process to withstand an increase in the number of runtime evidence variables can at least be partially guaranteed if there is an accurate mapping from QoS attributes to runtime evidence variables. Agile design and development in our context calls for small, iterative changes to the architecture allowing for learning from experience. Therefore, only few QoS attributes are of concern in each runtime iteration of AddedValueUpdate. We acknowledge that this discussion regarding scalability can be further strengthened in future work using empirical a systematic scalability analysis process.

4.5 EVALUATION

In this section we evaluate the extent to which our contribution addresses the problems of concern (listed in Section 4.1). In particular we aim to answer the following evaluation questions:

- **EQ1:** What unique benefit does our evaluation process provide when evaluating the potential added value of granularity adaptation decisions under uncertainty?
- **EQ2:** How explicitly does our process capture runtime dynamics of microservice architectures and reflect them on the potential added value of granularity adaptation?

- **EQ3:** To what extent is output of the economic analyser sensitive to changes in the input provided to it?

We facilitate investigating EQ1 and EQ2 by setting up a controlled experiment on a Filmflix case study. Using case studies can provide empirical and practical evidence regarding the appropriateness of software engineering processes to a particular application domain [205]. We consider controlled experiment on Filmflix to be the most appropriate evaluation means for investigating EQ1 and EQ2; it provides significant scale for answering them. In particular, looking at microservice-related literature (e.g., [146, 177, 8, 215, 304, 226, 121, 176, 135, 94]), we observed that microservice-specific proposed architectural design support processes can be evaluated on case studies of comparable scale and scope of functionality to Filmflix.

We implement the initial Filmflix architecture using AWS Lambda. AWS Lambda "is a computing service that lets you run code (which fits different templates — including a microservice template) without provisioning or managing servers [30]." AWS Lambda wraps the source code as a Lambda function. We inject three different trends of end user workload (stepwise, single spike and multiple spikes) into the Lambda function at runtime. This variation in trends allows us to show clearly how the process reflects runtime dynamics on the calculation of added value of granularity adaptation (helping to answer EQ2 above). We thereafter record the QoS attribute values using AWS CloudWatch [19] for each workload trend and use the records for our evaluation.

In Section 4.5.1 we describe a controlled experiment setting in which we apply our process and traditional binomial analysis to the AWS Lambda Filmflix implementation when injected with each workload trend described above.

In Section 4.5.2 we compare our process's output against representative state-of-the-art microservice runtime monitoring tools in this controlled experiment setting. The comparison shows tangible evidence of our process's benefits in linking the technical decision of granularity adaptation to its potential added value (thereby answering EQ1 above).

In Section 4.5.3, we use the setting from Section 4.5.1 to compare our evaluation process with traditional binomial real options analysis. Our comparison shows that

our process goes beyond traditional real options analysis by successfully performing real time updates of added value in response to runtime evidence variable values (e.g. workload variations). Successfully reflecting runtime dynamics means our process can track and update at runtime the potential added value of microservice granularity adaptation (thereby answering EQ2 above).

In Section 4.5.4, we report on a Java implementation of our approach to show its feasibility as a tool support to microservice adopters in industry.

In Section 4.5.5 further discuss the stability of our process's output in response to incremental changes in the inputs provided to the process (Section 4.5.5.2) using a controlled experiment (Section 4.5.5.1). We consider stability as an assessment of how sensitive the output of a process is to changes in input to it. This allows pinpointing the potential sensitivity points in our contribution more accurately. Assessing stability is significant to achieve two goals: 1) perfecting estimations of the most critical input parameters and, 2) providing at least a partial guarantee that the fluctuations in total real option values are due to runtime evidence variable fluctuations rather than the manual estimation updates of the input parameters. Both goals feed into addressing EQ3 above.

4.5.1 Experimental Setup

The initial architecture of Filmflix contains three microservices: ReviewRegulation — implementing the regulation of movie reviews, ReviewUpload — managing the user input requirements when uploading a review, and MovieReview — capturing input from the user through an interface. We consider that the microservitization in this application is driven by enhancing the performance of Filmflix where the expected runtime workload (i.e. number of reviews sent to Filmflix) is high.

The utility of concern is Filmflix performance in response to workload fluctuation. Therefore, the invocation duration of a microservice (i.e. time elapsed from when the microservice code starts executing as a result of an invocation to when it stops executing [353]) and the length of each review received by Filmflix are useful variables to observe at runtime and use them to inform the evaluation of granularity adaptation decisions.

We consider the following setting of parameters for a controlled experiment that applies our process's calculations on the initial architecture. Since we are applying our process in a controlled experiment setting, we consider tighter than typical values for the claim thresholds on invocation duration and review string lengths provided to our process as input. These tight values give an insight into how our process operates on both significant and non-significant Bayesian surprises given different workload trends. These insights therefore can also be transferable to cases where the claim thresholds are typical rather than tight.

- *Microservice:* Since MovieReview encapsulates other microservices in the initial architecture, we choose to monitor MovieReview at runtime to get a comprehensive view of the system's behaviour given different workload trends. However, we do acknowledge that monitoring larger or smaller parts of the architecture can serve other rationales (e.g., pinpointing the cause of a malfunctioning system).

- *QoS attribute:* We assume microservitization is driven by enhancing high performance provision. We measure performance using the invocation duration of MovieReview, monitored at every runtime iteration of our evaluation process.

- *Workload:* Each review string corresponds to a separate call to the MovieReview AWS Lambda function. Since MovieReview encapsulates 2 other microservices, each review string corresponds to two internal invocations to ReviewUpload and ReviewRegulation. To show how our process reflects the workload fluctuations on the calculation of added value at runtime, we consider workload as the written review string submitted by the end user. Furthermore, we control this workload at runtime by varying each review string length. We separately inject three different trends of review string lengths (step-wise increase in lengths, single spike, multiple spikes with complying and non-complying review strings); the values we inject in each trend are listed in Table 4.5. In the real world, the workload will be monitored rather than controlled. However, controlling one of the runtime evidence variables by injecting different trends of end user workload allows evaluating the effectiveness of our process in capturing different cases of runtime dynamics and reflecting them on the added value calculations.

- *QoS attribute threshold:* To ensure that significant Bayesian surprises are triggered at runtime, we assume a tight invocation duration threshold of 1 millisecond when articulating the claim to be assessed at runtime.

- *Workload threshold:* To ensure that a range of significant and un-significant Bayesian surprises are triggered at runtime, we use 100 sentences as the workload threshold when articulating the claim. We infer this threshold by testing our implementation of the initial architecture. The testing shows that for reviews more than 100 sentences long, MovieReview calls for extra regulatory reviews from ReviewRegulation.

- *Claim:* Given the thresholds above, we consider the claim to be assessed at runtime as a claim using if and only if ($\leftrightarrow$): "review string length $>$ 100 sentences $\leftrightarrow$ Invocation Duration $>$ 1 millisecond". We acknowledge that these claims can be set in the future as alarms within in AWS CloudWatch to alert our evaluation process whenever a claim is violated. In that case, our evaluation process would be more event-based rather than iterative.

- *Surprise tolerance threshold:* To stress test our process, we assume that improving performance is critical to the software architects, calling for a tight Bayesian surprise tolerance threshold (15%).

- *Runtime iteration duration:* Assuming that improving the performance is critical to software architects and given that the QoS attribute threshold is in the order of milliseconds, the length of each runtime iteration of our process needs to be in the order of milliseconds as well. Therefore, we assume the runtime iteration is 5 milliseconds long. We inject a single review string in each runtime iteration.

- *Option expiry date:* The option expiry date dictates the number and depth of constructed binomial trees. Therefore, we consider three timestamps as the option expiry date to avoid prohibitive computational costs in each iteration of our process. In other words, an option enabled by adapting or retained by not adapting will be available for three runtime iterations after pursuing adaptation.

- *Maintenance cost of the adapted architecture:* We use CostHat [196] (explained in Section 4.4.1) to estimate the maintenance cost if granularity is adapted. We define the service call graph for a sample adapted architecture — where MovieReview and ReviewRegulation are merged into a single microservice — and calculate the maintenance cost for this sample. In particular, there would be calls cascaded from MovieReview to and from ReviewUpload for each request from the end user to the application. Cascading calls in each direction cost an average of £150 when all four parameters of CostHat are considered. Therefore, the overall maintenance cost of the adapted architecture would be £300. We acknowledge that this value can be further elaborated if the specific manner of adapting granularity is known and/or other more fine-grained cost models are used.

- *Maintenance cost of the un-adapted architecture:* we use CostHat to estimate the maintenance cost of keeping the initial architecture. In our evaluation, we assume the architects use the microservice-specific CostHat model [196]. It can be used for any Lamda-backed or instance-backed microservice application. This model elicits the average maintenance cost over the lifetime of an application based on a service call graph which captures how a request to the microservice application's front-end cascades to other business logic microservices. CostHat is a parametric cost model which considers four parameters of maintenance cost: "1) compute costs (payment for the CPU time necessary to process service requests), 2) costs of API calls (per-request payment for each API call), 3) costs of I/O operations (payment for, for instance, database or file system writes) and, 4) costs of additional options (e.g., payment for elastic IP addresses) [196, p.3]." For our case, this is the maintenance cost given that the MovieReview microservice cascades calls to both the ReviewUpload and ReviewRegulation microservices in the initial architecture and then ReviewUpload cascades a call back to MovieReview. Each cascaded call costs on average £170 using CostHat, totalling to an average of £500. We acknowledge other cost models can be used to estimate this cost.

- *Re-structuring cost:* We assume architect judgement is used to estimate re-structuring cost in our evaluation. There are three microservices in the initial architecture: Re-

viewUpload, ReviewRegulations and MovieReview. Amending each microservice can require an average £500, totalling to £1500 for re-structuring the initial architecture. This cost can alternatively be elicited more accurately from publicly available cross-company data. However, this experiment is more concerned about illustrating the impact of runtime dynamics in our evaluation rather than soundness of the underlying cost estimations.

This setting covers situations where we expect to see nasty surprises, good surprises and the claim being verified.

4.5.2 Results: Capturing Added Value Under Uncertainty

In this section we highlight the unique benefit of our evaluation process thereby answering EQ1. Figure 4.3 shows the invocation durations of MovieReview when monitored using AWS CloudWatch for 14 runtime iterations in the case of stepwise workload increase. In Figure 4.3 every two increments on the x-axis corresponds to increasing the length of the review string by 30 sentences. After 20:00:00.25, the invocation duration decreases despite continuing to increase the workload stepwise. It is worth noting that the output of AWS CloudWatch is produced in milliseconds rather than in monetary terms (as is the case in our process). AWS Cloudwatch does not show the ramification on added value when answering the question of when to adapt microservice granularity (or not) at runtime.

Figure 4.4 presents the output of utilising our process; it shows the total potential real option values which can be enabled by adapting and by not adapting the granularity in the same runtime iterations given the invocation durations in Figure 4.3. Figure 4.4 provides a value-driven basis to reasoning about granularity adaptation decisions. For example, runtime iteration 5 shows the total potential real option value which would be enabled by adapting to be higher than the value retained by not adapting. The ability to take added value into account is a unique benefit of our approach.

Comparing Figures 4.3 and 4.4, only the latter provides value-driven justification for pursuing adaptation by presenting its ramifications related to added value. If Figure 4.3 is used alone to answer the question of whether to pursue adaptation or not in

iteration 13, intuitively the architects would decide to pursue granularity adaptation to reduce the invocation duration which had increased over the previous 2 iterations. On the other hand, Figure 4.4 suggests keeping the current granularity level in iteration 13 despite the increase in invocation duration. Therefore, relying on AWS Cloudwatch alone gives less informative insight compared to our contribution's insights regarding the ramifications of granularity adaptation decisions.

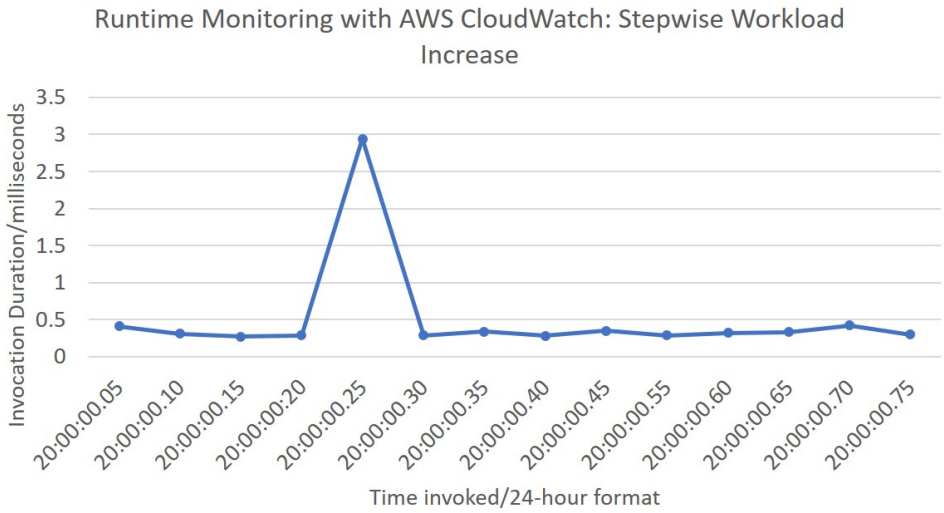


Fig. 4.3 AWS CloudWatch output for stepwise workload increase (i.e. increasing the review string by 30 sentences) injected to Filmflix in Section 4.5.1

In Figure 4.3 there is no spike in invocation duration at 20:00:00.40. However, the corresponding iteration 8 in Figure 4.4 promotes adapting since the maintenance cost of the un-adapted architecture is higher in this runtime iteration upon examining them using CostHat. If only AWS Cloudwatch is used to answer the question of whether to adapt or not in iteration 8, the intuitive answer would be not to adapt since there is no apparent technical benefit. The technical decision ignores the potential option value, that if factored in, would change the adaptation decision. Factoring cost factors in value-driven assessment means our contribution provides a more effective means to reason about granularity adaptation leading to a different decision altogether.

Looking at other industrial runtime monitoring tools, the X-Pack plugin provides the most flexibility for monitoring the health of microservice applications at runtime [107] to our knowledge. In particular, this plugin is compatible with cloud container providers (e.g. Elastic Search, Docker), logging tools (e.g. Logstash) and visualisation

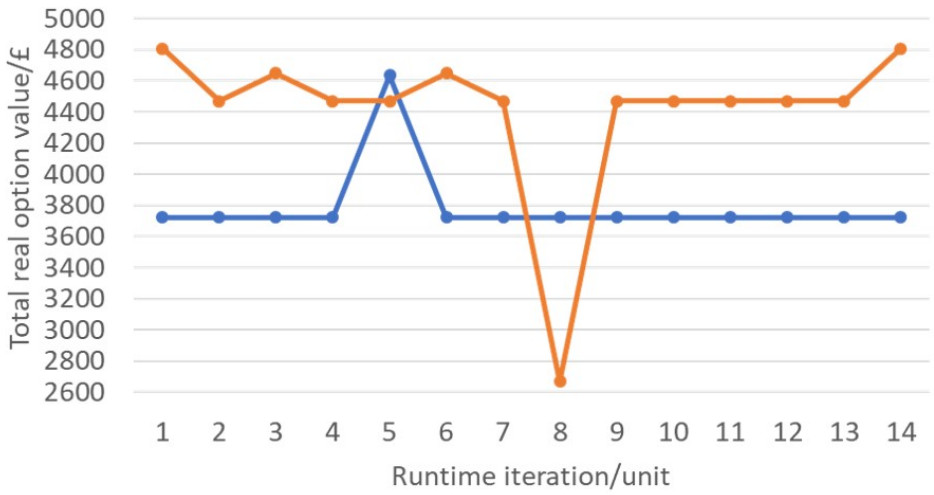


Fig. 4.4 Corresponding output of our process over same time period as in Figure 4.3; 1 runtime iteration = 5 milliseconds; the blue line refers to the real option values of adapting granularity while the orange line refers to the real option values of keeping the granularity level unchanged

tools (e.g. Kibana) popular among microservice adopters. Depending on the logging and visualisation settings, different forms of data analytics can be carried on recorded values of these metrics (e.g. aggregation). However, it does not explicitly perform any value-driven analysis on the recorded raw data like we do in our process. Riemann [281] and IBM Bluemix [161] take an event-driven approach to monitoring by setting alerts on specific events or diagnosing event streams. However, they lack the ability to perform value-driven analysis. Nevertheless, these tools are complementary rather than alternative to our process. They are all very suitable candidates to flexibly record a plethora of runtime evidence variables which can be used in our process. Other state-of-the-practice microservice orchestration platforms which can benefit from our value-driven analysis include Kubernetes [124] and Istio [29].

4.5.3 Results: Capturing Microservice Runtime Dynamics

To check the effectiveness of our approach in capturing runtime dynamics, we compare it against traditional binomial real options analysis by investigating a number of questions (which are a breakdown of EQ2):

- When is a surprise triggered in our approach? In particular, we aim to evaluate whether our contribution behaves as the following expectations:
  - We expect the stepwise increase in workload to trigger a nasty surprise if the invocation duration is greater than 1 millisecond for strings less than 100 sentences long.
  - We expect a good surprise if the invocation duration is less than 1 millisecond for strings larger than 100 sentences long.
  - In the single spike workload, we expect to see the claim verified (i.e. invocation duration greater than 1 millisecond) when the review string is exceptionally long. Following that claim verification, we expect a good surprise to be triggered as the QoS improves (i.e. becomes less than 1 millisecond) after that spike. We expect to see a similar trend of claim verification followed by good surprises in the multiple spikes workload.
- Does triggering surprises lead to a more informed verdict than that of traditional binomial analysis?

We calculate the total real option values which can be enabled by adapting and or retained by not adapting over 14 runtime iterations for each workload trend. We assume that the cost estimations are constant across the runtime iterations except for 2 cases where we manually update them.

We use Bayesian networks [290] to calculate the claim posterior in each iteration. Bayesian networks allow us to capture the dependency between review string length, invocation duration and the truth of the claim.

The posterior probability of the claim holding true (*P(Claim | REV)* in Eq. (4.11)) is derived from frequency tables. Each record in the table has a combination of invocation duration and review string length (the *REV*) ranges and the corresponding number of times the claim held true ($t_{claim}^{StringLength,InvocationDuration}$) and was violated ($f_{claim}^{StringLength,InvocationDuration}$) within that range. The posterior probability is derived

from the correct record of the frequency table as:

$$P(Claim|REV) = \frac{t_{claim}^{StringLength,InvocationDuration}}{t_{claim}^{StringLength,InvocationDuration} + f_{claim}^{StringLength,InvocationDuration}} \quad (4.11)$$

	Monitoring Timeslot													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Review String Length/sentences	20	20	50	50	80	80	110	110	140	140	170	170	200	200
Invocation Duration/milliseconds	0.41	0.31	0.29	0.32	2.94	2.93	0.34	0.28	0.35	0.29	0.32	0.33	0.42	0.3
Bayesian surprise	0.194	0.06	0.147	0.07	0.246	0.147	0.07	0.66	0.07	0.07	0.07	0.07	0.07	0.194
Review String Length/sentences	20	20	50	50	80	80	110	110	140	140	1000	170	170	200
Invocation Duration/milliseconds	0.45	0.33	0.36	0.33	0.45	0.33	0.36	0.42	0.4	0.56	1.55	0.35	0.46	0.36
Bayesian surprise	0.195	0.08	0.147	0.08	0.246	0.147	0.06	0.06	0.07	0.66	0	0.195	0.06	0.05
Review String Length/sentences	20	20	50	50	80	80	110	1000	110	170	170	170	1120	170
Invocation Duration/milliseconds	0.53	0.38	0.41	0.46	0.51	0.4	0.43	1.55	0.35	0.46	0.35	0.28	2.9	0.35
Bayesian surprise	0.195	0.09	0.166	0.06	0.05	0.799	0.33	0	0.66	0.07	0.05	0.29	0	0.19

Fig. 4.5 Red cells = a nasty surprise is triggered; green cells = a good surprise; yellow cells = a surprise below the Bayesian surprise tolerance threshold T_s; grey cells = claim held true

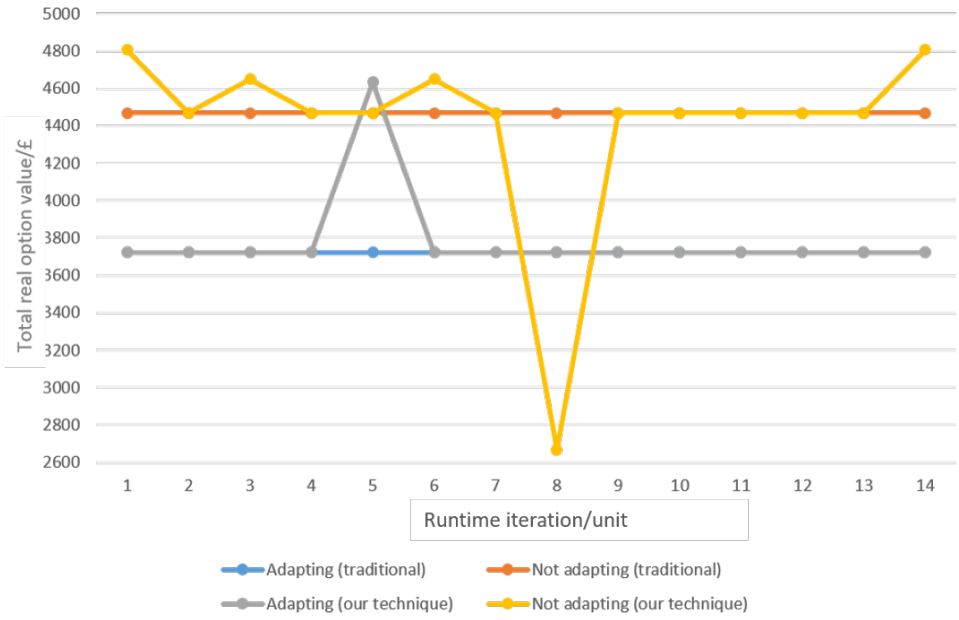


Fig. 4.6 Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with stepwise workload increase using our process against traditional analysis

Figure 4.5 shows the data used to generate our contribution's outputs (Figures 4.6, 4.7 and 4.8). In Figures 4.6, 4.7 and 4.8, our process reports fluctuation in the total real option values due to the real time calibration of the value enabled by adapting (or not). In contrast, traditional binomial analysis reports constant results for the total real option values.

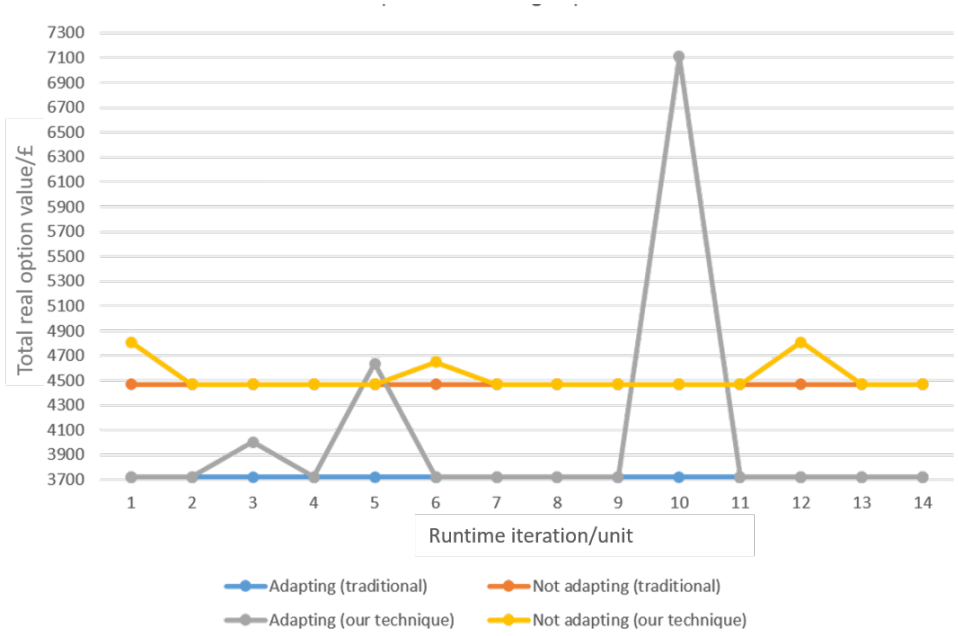


Fig. 4.7 Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with single spike workload using our process against traditional analysis

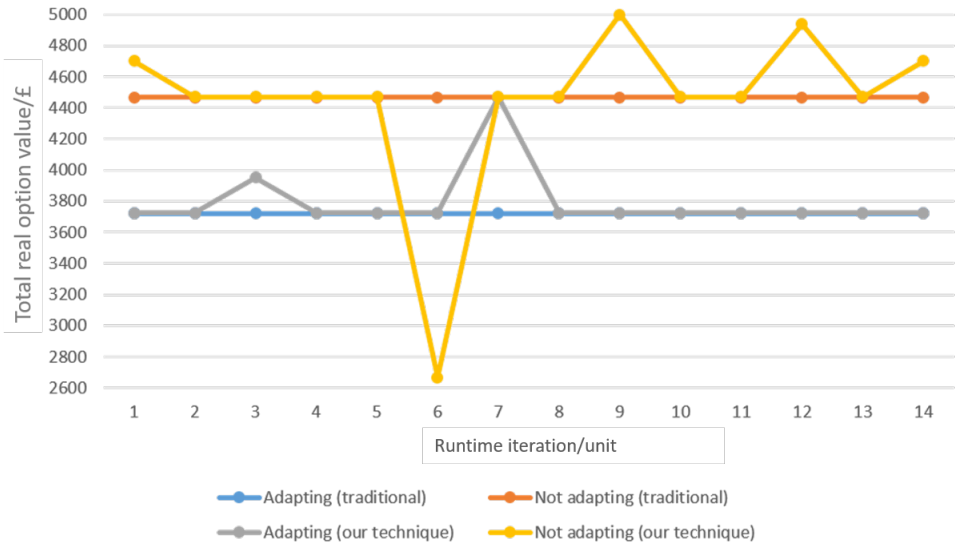


Fig. 4.8 Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with multiple outliers in workload using our process against traditional analysis

In Figure 4.6, runtime iteration 5 corresponds to a nasty surprise (0.246 in Figure 4.5). The claim is violated since the invocation duration was greater than 1 millisecond for a review string less than 100 sentences long. It indicates that the system is behaving worse than initially believed. The triggered Bayesian surprise is nasty as per our first

expectation regarding our contribution's behaviour. Therefore it increases the total real option value to be enabled by adapting, triggering a suggestion to adapt in this runtime iteration. It is at up to the architect to take or to leave this suggestion. Our contribution provides the suggestion along with the objective justification for it in terms of potential added value. Therefore, our contribution provides a more informative suggestion regarding granularity adaptation compared to traditional binomial real options analysis. From iteration 9 to iteration 13, the stepwise increase in workload does not trigger a significant surprise since the shifts in the observed invocation duration are insignificant given the shifts in review string lengths.

In Figure 4.7, between iterations 9 and 10, the invocation duration has fluctuated even though the review string length has not changed. This combination gives significant information about the system's runtime behaviour triggering a nasty surprise above the tolerance threshold in runtime iteration 10. In iteration 11, a string 1000 sentences long is injected and the invocation duration is greater than 1 millisecond. Therefore, the claim was verified in iteration 11 leading to zero Bayesian surprise, so the total real option values in our process and the traditional analysis were the same. The claim is then violated in iteration 12 triggering a good surprise (0.195) since the invocation duration was less than 1 millisecond for a review string longer than 100 sentences. This aligns with our second expectation regarding our evaluation process's behaviour. The Bayesian surprise in runtime iteration 12 therefore indicates that the situation in iteration 11 was a single spike and thereby our process calls for not adapting at iteration 12. This trend of claim verification followed by ag good surprise aligns with our third expectation regarding our contribution's behaviour. Even though the traditional binomial real options analysis leads to the same verdict for that iteration, the total real option value corresponding to not adapting is higher in our process. This shows our process can more confidently argue for the verdict in terms of a higher total real option value corresponding to not adapting due to the runtime evidence records for iterations 11 and 12.

In Figure 4.8, between iterations 11 and 12 in the multiple spikes workload, the fluctuation of invocation duration triggered a good surprise at iteration 12. Runtime iterations 9 and 14 show good surprises (0.66 and 0.19 respectively). In iterations 9

and 14 the claim is violated since the invocation duration is less than 1 millisecond after exceptionally long strings are injected in iteration 8 and 13 respectively. Similar to Figure 4.7 and as per our expectation, there claim verification is followed by a good surprise in the multiple spikes workload as well.

However, the total real option value corresponding to not adapting in runtime iteration 14 is less than in iteration 9. This indicates that the smaller surprise in iteration 14 is due to less confidence about the behaviour of the system when long strings ("spikes") are injected frequently. This shows that our process can explicitly capture changes in the confidence as runtime evidence accumulates. The software architect can find this accumulated information useful in possible updates of the claim threshold. This can be done by calculating the mean review string length and invocation durations observed after a window of runtime iterations and then mapping that to more relaxed claim thresholds.

Iteration 8 in Figure 4.6 and iteration 6 in Figure 4.8 show the impact of manual cost updates on the output of our process. Here we increase the maintenance cost of the un-adapted architecture when performing calculations using our process. The maintenance cost is kept constant when performing calculations using traditional binomial analysis. Despite the good surprises (0.66 in iteration 8 for stepwise workload increase and 0.799 for multiple spikes workload), our process promotes adapting for these iterations due to the high maintenance costs for the un-adapted architecture. This shows that our process can capture the runtime dynamics of cost as well. Traditional real options analysis might be able to only capture runtime dynamics if it is applied every time the costs are manually updated.

4.5.4 Results: Proving Feasibility for Tool Support

We implement the AddedValueUpdate pseudocode into a Java tool[1] to prove the feasibility of providing an interactive runtime decision support for microservice granularity adaptation grounded on our novel evaluation process. This implementation can also accelerate the adoption of our contributions in industry (source code submitted

[1]The source code of this tool is available at https://github.com/ssh195/AddedValueUpdate.git.

separately). The tool first takes the inputs related to calculating Bayesian surprises (Figure 4.9), including location of the files where the runtime evidence variables values will be streamed. The tool then takes the inputs related to construct the binomial trees for adapting (Figure 4.10) and not adapting the architecture (Figure 4.11) for the first runtime iteration. At the end of each iteration, the tool outputs a suggestion to adapt or not along with the real option value on which the suggestion is grounded (Figure 4.12). Even such a primitive implementation of our approach illustrates its feasibility as a tool support to microservice adopters in industry. Moreover, when we tested this tool support with the experiment from Section 4.5.1, we observed that its performance was acceptable in the context of an interactive tool where no re-structuring is done to the architecture on-the-fly.

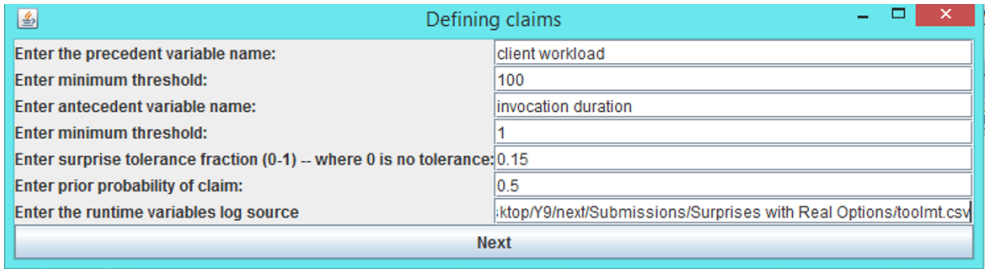


Fig. 4.9 Capturing inputs for calculating Bayesian surprises for the first iteration of our evaluation process used in the experiment in Section 4.5.3

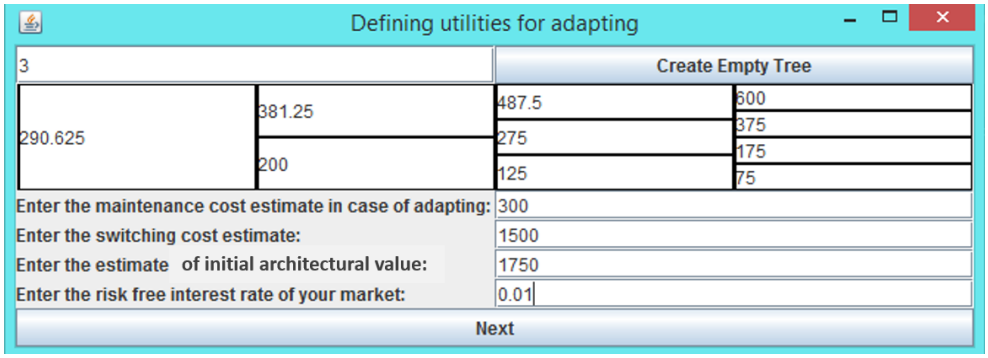


Fig. 4.10 Capturing the utility tree of adapting for the first iteration of our evaluation process used in the experiment in Section 4.5.3

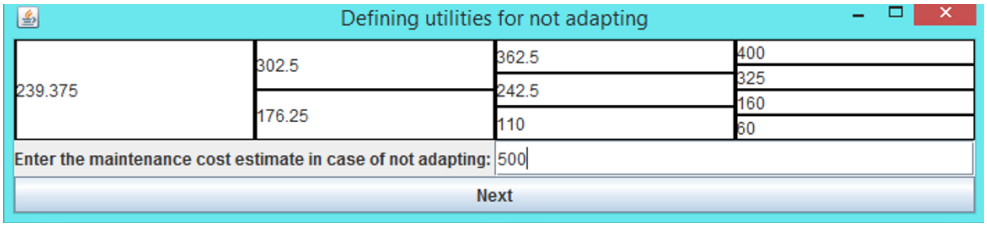


Fig. 4.11 Capturing the utility tree of not adapting for the first iteration of our evaluation process used in the experiment in Section 4.5.3

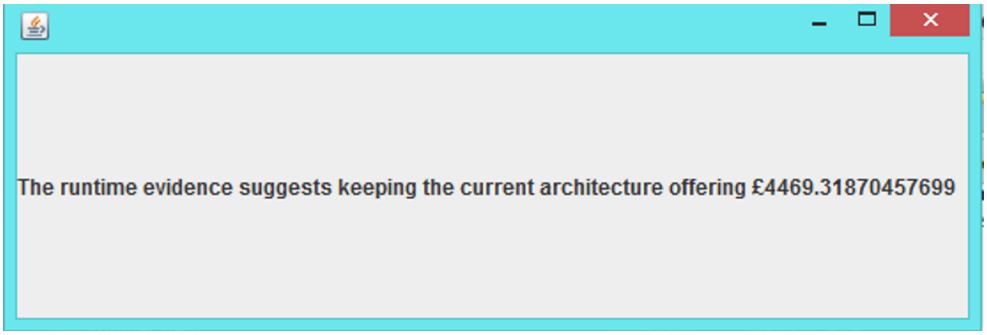


Fig. 4.12 Suggesting keeping the current architecture after the first iteration of our evaluation process

4.5.5 Proposed Process Stability Evaluation

4.5.5.1 Experimental Setup

To address EQ3 above, we assess the stability of our process's output in response to changes in the inputs it utilises. In other words, we investigate whether it is sensitive to changes in the estimated inputs, listed in the first column of Table 4.3.

For each input parameter, we increment/decrement it iteratively by a pre-defined factor starting from the actual estimates of the initial architecture. We elicit this factor from the sensitivity analysis approach in [106] which utilises seminal work [37, 188, 295] to measure the quality of multi-parameter decision functions; the economic analyser is the decision function in our case.

A decision function in [106] always outputs the candidate with maximum aggregated value where the aggregated value is a function of weighted input parameter values. In that context, the candidates are adapting and not adapting granularity. The aggregated values are the total real option values for adapting and not adapting for a runtime iteration of the planning engine. The input parameters used to calculate this aggregated value are those listed in Table 4.3 (not weighted in our case). Ultimately,

the economic analyser outputs the verdict (i.e. "candidate") with the maximum total real option value (i.e. maximum aggregated value).Therefore, we consider the evaluation setting in [106] a neat fit to inspire defining the increment/decrement factor. We assess the stability of output when the input is incremented or decremented by $\pm 5\%$, $\pm 10\%$, $\pm 15\%$ and $\pm 20\%$ in our controlled experiment. Accurate determination of this factor in terms of the input parameters and their respective weights is itself an open research challenge [106] that can be addressed by specialised domain analysis. Nevertheless, we fluctuate the parameters along a range of deviation bounds to stress test our evaluation process.

We increment/decrement each input parameter over 10 iterations. For each iteration, the increment/decrement is done on the previous iteration's value. This avoids unrealistically large deviations from the actual inputs used in our case study (e.g., a cost estimation which is 200% above/below the value used in the experimental set up from Section 4.5.1) while still stress testing stability of the economic analyser. For each increment or decrement of an input parameter, we keep all the other inputs on the list constant to pin point the source and extent the evaluation process's stability in a controlled experiment setting. Moreover, we expose the assessed architecture to same workload trend for the same number of runtime iterations for each increment or decrement of an input parameter.

Inspired by [106], we calculate the deviation bounds for each input parameter from the starting point. As long as the parameter's value is within these bounds, the output suggestion of the economic analyser is not affected.

4.5.5.2 Results: Stability in Response to Input Changes

Using Table 4.3 and Figure 4.13 we can reflect upon the stability analysis goals above. Across all the increment/decrement factors, the evaluation process is most tolerant of inaccurate estimations in QoS thresholds of the claim representing the architects' expectations of the runtime system behaviour and the risk-free interest rate representing possible market fluctuations. To guarantee that the process's output is impacted by runtime evidence variables, the architects' estimations for each input

Input parameter	Increment/Decrement factor/±%			
	±5%	±10%	±15%	±20%
	Deviation bound/±%			
QoS threshold in articulated claim	107.763	112.253	117.865	120.110
Risk free interest rate	106.640	112.253	117.865	123.478
Claim prior probability	39.039	55.771	78.079	100.387
Not adapting utility tree	16.8	28	42	50.4
Maintenance cost of the adapted architecture	10.1902	25.4755	38.21325	48.40345
Adapting utility tree	10.775	21.55	34.48	43.1
Maintenance cost of the un-adapted architecture	10.775	21.55	32.325	45.255
Claim workload threshold	7.42	18.55	29.68	38.955
Surprise tolerance threshold	9.275	18.55	31.535	40.81
Re-structuring cost	9.275	18.55	29.68	38.955
Initial architectural value	5	10	16	23

Table 4.3 Result of stability analysis of the economic analyser for the initial architecture

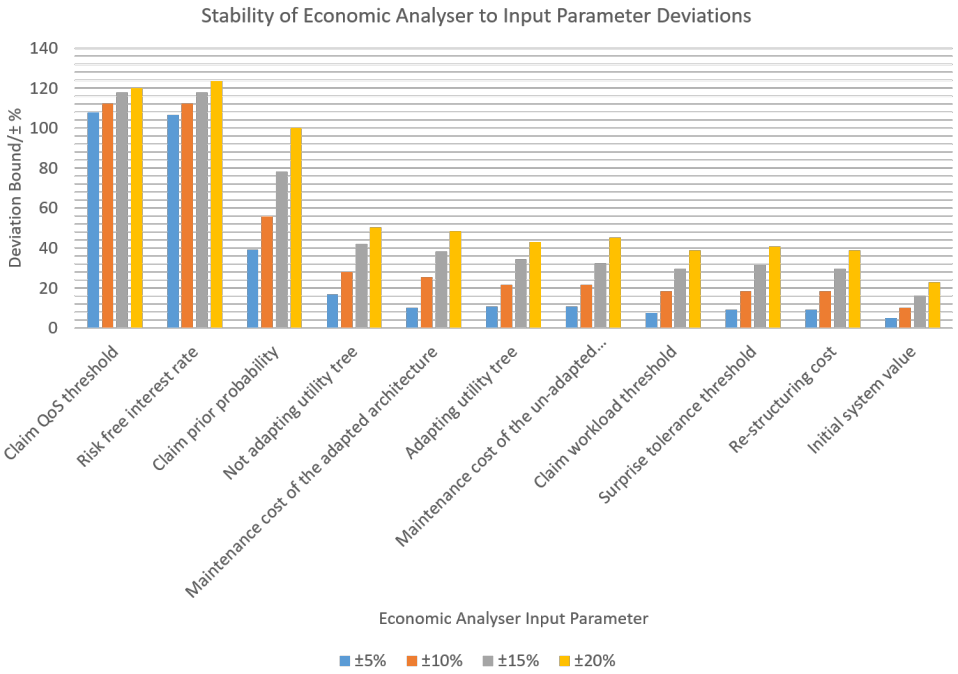


Fig. 4.13 The deviation bounds — from Table 4.3 — within which changes to each input parameter do not impact the output suggestion of the evaluation process

parameter should vary only within the corresponding deviation bound. In this context, the deviation bound is an indicator of the confidence level in the estimated value.

By comparing the deviation bounds for the factors in each parameter in Figure 4.13, we can determine whether the impact of the input parameter on the stability

is almost linear or constant. For example, the deviation bounds for the claim QoS threshold and risk-free interest rate only slightly deviate from each other across the increment/decrement factors. This implies an almost constantly low impact of these parameters on the process's output. On the other hand, the deviation bounds of parameters such as the re-structuring cost and the initial architectural value have an almost linear relationship with the increment/decrement factors. For example, the deviation bound for the $\pm 5\%$ factor in the re-structuring cost (9.275%)is almost half the deviation bound for $\pm 10\%$ factor (18.55%).

4.6 CONCLUSION

In this chapter, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem inspired by the theory of Baldwin and Clark [36]. We propose a novel dynamic evaluation process that combines binomial real options analysis with Bayesian surprises to assess the added value of adapting granularity (or not). Our evaluation process is a novel hybrid combination of binomial real options theory [338] and the concept of Bayesian surprises [44]; it is a novel process customised for the context of microservice granularity adaptation. Our work is the first to:

- Dynamically evaluate the potential added value under uncertainty related to pursuing microservice granularity adaptation.
- Enable runtime, dynamic application of binomial real options analysis as opposed to its traditional static design time usage.
- Apply the concept of Bayesian surprises in the context of binomial real options analysis.
- Apply the concept of Bayesian surprises in the context of microservice granularity adaptation.

We compare four representative industrial microservice runtime monitoring tools — AWS Cloudwatch, IBM Bluemix, Riemann and X-Pack — against our evaluation process. The results provide tangible evidence on the benefits of linking runtime

granularity adaptation decisions to their added value under uncertainty. We also use an AWS Lambda Filmflix implementation to compare our process against traditional binomial real options analysis. The results show how our process goes beyond traditional application of real options by revealing the impact of runtime variations in workload on the added value which can be enabled adapting (and not adapting) granularity. We further show the feasibility of our approach to assist microservice adopters in industry by reporting on a Java implementation of it. The Java implementation captures the inputs needed for our evaluation process, processes them and produces an value-driven output regarding whether to pursue granularity adaptation at a runtime iteration or not. Additionally, we evaluate the sensitivity of output of our evaluation process relative to changes in the input provided to it.

Our novel evaluation process opens up the path to answering another crucial related question in the next chapter: Which granularity adaptation strategy (e.g. decomposing a microservice with unrelated features) is suitable at a certain point at runtime?

Chapter 5

A Value-driven Planning Engine for Runtime Microservice Granularity Adaptation

In Chapter 4, we contribute to a novel runtime, iterative, dynamic value-driven architecture evaluation process to assess the added value of adapting granularity (or not) and thereby to determine if it worth pursuing adaptation. However, there is no existing comprehensive decision support system for software architects regarding combines the role of our evaluation process from Chapter 4 with suggesting suitable granularity adaptation strategies where possible. This problem gives rise to Research Question 4: How can microservice granularity adaptation can be effectively engineered at runtime, combining systematic modelling of granularity adaptation strategies with objective reasoning about added value? In this chapter, we address this research question by contributing to a runtime, interactive, iterative planning engine aiming to provide this support by incorporating the evaluation process which we contribute in Chapter 4. Our planning engine provides software architect(s) with recommendations on suitable granularity adaptation strategies or on revisiting unrealistic strategies defined at design time. These strategies are captured using the modelling approach we contribute in Chapter 3. In other words, the planning engine combines systematic modelling of granularity adaptation strategies (using Chapter 3's contribution) with

objective reasoning about the predicted cost and added value of the exercise (using Chapter 4's contribution). The planning engine thereby helps effectively engineer runtime microservice granularity adaptation (addressing Research Question 4). We use Filmflix to evaluate our planning engine in multiple controlled scenarios. We then discuss the planning engine's efficiency in providing decision support given the dynamism of microservice environments. In particular, we discuss the planning engine's reaction to changes in the input provided to the planning engine.

5.1 INTRODUCTION

In this chapter, we contribute to a systematic, effective decision support system which suggests (where possible) granularity adaptation strategies that can enable added value to the microservice architecture (thereby addressing Research Question 4). This contribution addresses a gap characterised by the absence of a decision support system for reasoning about microservice granularity adaptation at runtime. As explained in Chapter 4, added value quantifies the difference between potential benefit (regarding microservitization utilities of concern) and cost of pursuing granularity adaptation (or not).

This chapter aims to orchestrate our contributions from chapters 3 and 4 into a decision support system which answers the following questions related to runtime granularity adaptation:

- Is it worth pursuing granularity adaptation in the first place?
- If it is worth adapting, which granularity adaptation strategy can enhance microservitization utilities of interest?

Given the output of the planning engine, the architects can enforce granularity adaptation at runtime. The candidate adaptation strategies are not incepted on-the-fly by the planning engine. Therefore, we view runtime granularity adaptation is planned rather than on-the-fly in our usage context.

The novel contribution of this chapter is *an interactive, iterative, runtime planning engine which orchestrates our dynamic evaluation process from Chapter 4, microservice ambients and granularity adaptation aspects from chapter 3.* The planning engine

comprises three components: an economic analyser, an architectural configuration chooser and a granularity adaptation aspect chooser. The input of the planning engine are candidate granularity adaptation strategies that have been defined by the architect(s) at design time and reasoned about in a simulated environment. We utilise our work in Chapter 3 on modelling microservices using ambients and aspects to define these strategies. At runtime, the engine iteratively solicits runtime evidence variable values regarding the deployed architecture. It then reasons about the solicited values using our evaluation process. If it calls for pursuing adaptation within the current iteration of the planning engine, the planning engine provide software architect(s) with recommendations on suitable adaptation strategies or on revisiting unrealistic strategies defined at design time. Given the output of the planning engine, the architects can enforce granularity adaptation at runtime. The candidate adaptation strategies are not incepted on-the-fly by the planning engine. Therefore, we view runtime granularity adaptation is planned rather than on-the-fly in our usage context.

While our dynamic evaluation process proposed in chapter 4 is sufficient to determine if it is worth adapting a microservice granularity level or not in a value-driven manner at runtime, it does not provide the architect with any insight regarding how this adaptation should be pursued. On the other hand, microservice ambients and granularity adaptation aspects are sufficient only to show how adaptation would be pursued. Without our evaluation process choosing a strategy modelled using our modelling approach would be triggered based purely on technical observations rather value-driven justification. Therefore, our planning engine provides a comprehensive value-driven decision support — to determine both if it worth pursuing granularity adaptation and to suggest how it should be pursued. Though we present the planning engine as an interactive decision support system for granularity adaptation, our contribution can be extended and applied in a self-adaptive usage context.

Regardless the context in which the planning engine is used, its output can benefit microservice adopters in many ways. Consider that a fictional running microservice-based application with a functionality and scale similar to Netflix — called NetWatch — is using our planning engine. If it suggests keeping the current architecture without adapting, this spares NetWatch an unjustified adaptation that will not unlock added

value to the architecture but rather incur unnecessary re-structuring and maintenance costs. On the other hand, if the engine suggests a single favourable granularity adaptation strategy, it confirms the architects' design time beliefs about the expected runtime workload or system behaviour on which they grounded their inputs to the planning engine. This confirmation can inform NetWatch architects regarding prioritisation of specific microservitization utilities, exploitation of a competitive advantage they have in the architecture (e.g., by entering new markets, supporting more user segments), or releasing new media content at strategic times which can increase their economic gains. If the planning engine suggests multiple suitable granularity adaptation strategies, it provides the architects with value-driven proof that the suitable strategy is within planning engine's output. This is significant to large industries like NetWatch where re-structuring the architecture can be a costly exercise. By having an objective justification that incurring this cost will unlock added value to the architecture, software architects can justify such re-structuring to business and/or financial personnel. Finally, if the planning engine calls for re-visiting unrealistic strategies it exposes the software architects to an emergent behaviour which they have not considered at design time. This is a natural situation in the highly dynamic environment where microservice(s) operate. Such exposure is critical to better understanding the market in which industries such as NetWatch operate. Thereafter this can highly assist planning future media content releases and/or marketing strategies.

We consider that realising a granularity adaptation strategy in our context involves temporarily taking off-line the concerned microservices to encode and configure the strategy. The adapted microservices are then deployed without changing the physical infrastructure. For example, if NetWatch were to apply a granularity adaptation strategy on its currently running microservice architecture, the architects would temporarily shut down the microservices of concern, carry out activities that enforce the strategy, then deploy the adapted microservices on the same physical infrastructure supporting NetWatch. Activities required to enforce the strategy include but are not limited to code refactoring, configuration, securing communications and implementing data translation layers.

We evaluate the planning engine by answering the following question: Does our planning engine result in a granularity adaptation strategy which *can introduce more added value* when compared to the initial architecture? We apply the planning engine on a Filmflix implementation in several controlled scenarios to answer this question. Analysing the planning engine output in these scenarios helps confirm that it suggests strategies which indeed can introduce more added value compared to the initial architecture.

Moreover, we assess how the planning engine provides *efficient support for dynamic microservice environments*. More specifically, we discuss the following question: How *stable* is the planning engine in response to the runtime changes in the inputs to each component? To answer this question, we asses the sensitivity of the planning engine's output to variation in the inputs fed to its constituent components. Our assessment shows that the architects' level of confidence in the inputs influences the planning engine's stability.

In Section 5.2 we use Filmflix to set the context for this chapter's contribution using the modelling primitives we contribute in Chapter 3 and the dynamic evaluation process we contribute in chapter 4. In Section 5.3 we present the problem formulation of this chapter given the motiving scenario in Section 5.2. In Section 5.4 we present the inputs, processing and output of our planning engine. Section 5.5 evaluates each planning engine component using an implementation of Filmflix along the dimensions mentioned above. In Section 5.6 we conclude by summarising the content of this chapter.

5.2 MOTIVATING SCENARIO

Consider the same Filmflix motivating scenario from Chapter 4 where two Filmflix architects defined at design time two alternative granularity adaptation strategies both motivated by enhancing Filmflix performance in response to workload fluctuation but each promoting this enhancement differently (i.e., by merging and decomposing). The architects defined both strategies using microservice ambients and granularity adaptation aspects from Chapter 3. They assumed the invocation duration and number

of reviews received by Filmflix as the runtime evidence variables. They expect high request volumes at runtime calling for a strict threshold on invocation duration in the granularity adaptation aspects to ensure high performance provision and avoid losing users to competing applications. We use this context to motivate the contribution of this chapter.

Each architect decides to take a different approach to inform granularity adaptation at runtime:

- Architect 1 decides to utilise our dynamic evaluation process from Chapter 4. He uses his expectation about the runtime workload and Filmflix invocation duration to articulate the following claim: "number of received reviews $>$ 1000 reviews per second $\leftrightarrow$ invocation Duration $>$ 1 millisecond." He considers the claim's prior probability to be 0.5. Furthermore, he uses relevant cost models and observations of the initial Filmflix architecture to estimate the input parameters needed by the evaluation process. In iterations where the evaluation process suggests pursuing adaptation, he ad-hocly chooses a granularity adaptation aspect without checking whether the trigger for it is met in the respective runtime iteration.

- Architect 2 decides to pursue granularity adaptation whenever the following condition is met: invocation duration $>$ 1 millisecond. He relies solely on stress testing the candidate granularity adaptation strategies under different workloads in a simulated environment and decides to adopt the strategy which withstands the most stress tests.

Our evaluation process gives architect 1 an objective, value-driven dynamic answer to the following question: Is it worth pursuing granularity adaptation in the first place? On the other hand, architect 2 uses a purely technical threshold on invocation duration to answer the same question at runtime. However, our evaluation process does not provide both architects with any insight regarding the following question: Which granularity adaptation strategy has more added value compared to the initial architecture? To answer this question, architect 1 ad-hocly chooses a granularity

adaptation aspect without comparing the observed runtime variables against the triggers of the granularity adaptation aspects defined at design time.

On the other hand, architect 2 analyses the alternative strategies in the simulated environment to determine which granularity adaptation strategy can enhance the microservitization utilities of interest. Microservice ambients and granularity adaptation aspects allow architect 2 to stress test the candidate strategies in the simulated environment supporting these modelling primitives. Stress testing gives some insight regarding how granularity adaptation should be pursued. However, the range of tests depends on the architects' intuition regarding the expected runtime environment in which Filmflix will operate. The highly dynamic environment in which microservice(s) operate means it is natural for architect 2 to miss some emergent behaviour scenarios when setting these tests at design time. Therefore, stress testing does not inform architect 2 regarding 1) whether or not design time beliefs about the expected runtime workload or Filmflix behaviour are realistic and, 2) whether or not the candidate strategies are realistic for the dynamic, runtime environment.

5.3 PROBLEM FORMULATION

Both the approaches above have the following problems and rooms for extension:

- *Problem 1 — inability to provide value-driven justification for pursuing granularity adaptation:* The modelling primitives we contribute in Chapter 3 do not capture the added value which can be enabled pursuing runtime granularity adaptation. Therefore, our dynamic evaluation process from Chapter 4 is needed to capture this added value and hence justify pursuing granularity adaptation in accordance with a chosen strategy captured using microservice ambients and granularity adaptation aspects.

- *Problem 2 — inability to assess which granularity adaptation strategies defined at design time are realistic and/or suitable at runtime:* Even if our modelling primitives are enriched with our evaluation process, these enrichments will not reassess the thresholds in granularity adaptation aspects based on runtime observations of a microservice environment and inform the simulated environment using these

re-assessments. In our motivating scenario, granularity adaptation thresholds are inferred at design-time based on high expectations of runtime workload. However, if runtime workload observations are different, then these thresholds were biased and need to be updated. If no re-visiting is required, then runtime observations can be used to direct the architects towards favourable granularity adaptation strategies from an input bank of candidate strategies defined by the architects at design time.

In this chapter, we contribute to an interactive planning engine which incorporates the runtime updates of added value calculations using our dynamic evaluation process from Chapter 4 (addressing Problem 1 above) and thereafter suggests suitable granularity adaptation strategies or revisiting the strategies if none of the candidates are suitable (addressing Problem 2 above).

5.4 PLANNING ENGINE

The planning engine is an interactive decision support system that informs the architects' granularity adaptation decisions given evidence for a single runtime iteration. Before every runtime iteration of the planning engine, the architects decide the scope of its target — the whole application, a single microservice, or a sub-system within the application. Determining the scope is inferred from the architects' judgement about the dependencies across microservices. The effort and time invested to determine the target scope can be highly influenced by operational/maintenance logs, time pressure to troubleshoot for likely degradations in the architecture and cost/effort needed to determine the target scope. Overall, the target scope should include only microservices that are interdependent. Such a scope ensures that the planning engine captures all the microservices where the added value of granularity adaptation will be manifested. The target scope needs to be determined following every iteration where granularity adaptation has been pursued.

The planning engine includes three components: an economic analyser, architectural configuration chooser and a granularity adaptation aspect chooser. The economic analyser uses our evaluation process from Chapter 4 to address Problem 1 in Section 5.2). If adaptation is to be pursued, the architectural configuration chooser is consulted

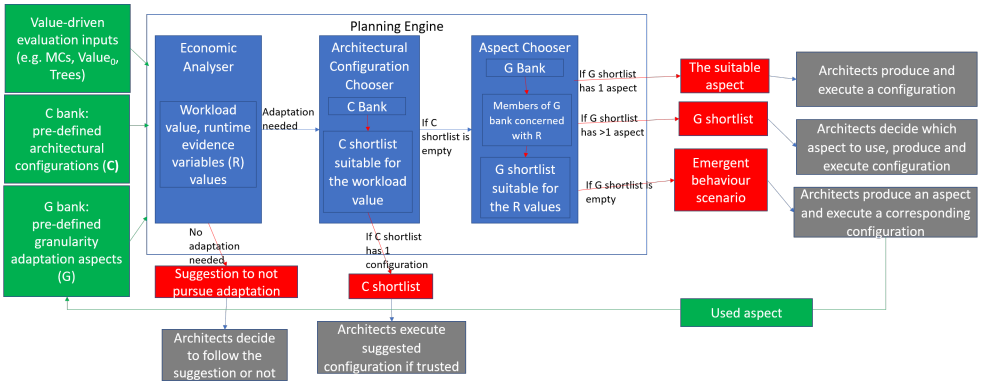


Fig. 5.1 Inputs (green boxes), components (blue boxes) and outputs (red boxes) for one runtime iteration of the planning engine; grey boxes are the actions required by the architects given the outputs; blue arrows refer to the action flow from one box to the next; red arrows refer to action flow leading to an output; green arrows refer to the flow of an input to a the planning engine; we carry over evaluation of the economic analyser from Chapter 4 and we evaluate the two choosers in this chapter

to suggest suitable candidate architectural configuration(s). If the architectural configuration chooser does not output any suitable configurations(s), the third component is consulted to suggest suitable granularity adaptation aspect(s). These two components address Problem 2 in Section 5.2. In this Section, we describe the input, process and output of the economic analyser (Section 5.4.1), architectural configuration chooser (Section 5.4.2), and the granularity adaptation aspect chooser (Section 5.4.3). The different possible outputs of the planning engine are numbered 1-5 across Sections 5.4.1, 5.4.2, and 5.4.3.

5.4.1 Economic Analyser (Chapter 4 contribution)

5.4.1.1 Economic Analyser Inputs

The architects provide the inputs required to carry out only the first iteration of our dynamic evaluation process (as listed in Section 4.4.3.1 of Chapter 4). As long as the target remains unchanged across iterations (i.e. adaptation is not pursued), the output of the previous AddedValueUpdate iteration is carried over as input to the next one. Otherwise, new input is needed to conduct our dynamic evaluation process on the newly deployed microservice(s).

5.4.1.2 Economic Analyser Process

Our evaluation process is a novel runtime, dynamic combination of real options and Bayesian surprises. It iteratively (in every planning engine runtime iteration) solicits and monitors runtime evidence then feeds it into an added value prediction process. Pseudocode 1 — AddedValueUpdate — in Chapter 4 presents the process of our proposed evaluation process. The economic analyser uses it to suggest whether adaptation should be pursued or not for a runtime iteration in the planning engine.

5.4.1.3 Economic Analyser Outputs

Output Case 1: At the end of each runtime iteration, if the economic analyser suggests keeping the current architecture, the output is presented as a final verdict regarding pursuing granularity adaptation and the total real option values used to arrive at this verdict. In this case, all inputs to the current iteration are carried over to the next one and the other planning engine components are not consulted.

On the other hand, if the economic analyser outputs a suggestion to pursue adaptation, the other planning engine components are consulted iteration. Following the economic analyser's suggestion to pursue adaptation is a cautious approach to unlock the added value in the real options rather than taking a reactive route which could result in missing opportunities for economic gains. Realising granularity adaptation ultimately involves code refactoring, configuration, securing communications, and implementing data translation layers among other activities on the same physical infrastructure of the running architecture. Finally, the adapted microservices are re-deployed without changing the physical infrastructure.

5.4.2 Architectural Configuration Chooser

If the suggestion is to pursue adaptation, the architectural configuration chooser is triggered.

5.4.2.1 Architectural Configuration Chooser Input

A bank of architectural configurations (*C bank* in Figure 5.1) which are defined at design time is fed by the architects into the planning engine. Each architectural configuration can be visualised using microservice ambients. Each architectural configuration is annotated with meta-data which defines the range of workload values which the architects expect the respective configuration to be suitable for. We assume that the code required to enforce the chosen architectural configuration (if any) is encoded by the architects after this planning engine component provides its output.

5.4.2.2 Architectural Configuration Chooser Process

Given the annotated *C bank*, the architectural configuration chooser compares the observed workload value against the meta-data attached to the configurations in the *C bank*. This comparison produces a configuration shortlist (*C shortlist* in Figure 5.1) which includes a suitable architectural configuration in this iteration, if any.

5.4.2.3 Architectural Configuration Chooser Output

Output Case 2: If the *C shortlist* has one configuration, the planning engine's output is a visual representation of that configuration modelled using microservice ambients. In this case, the planning engine confirms that the architects attached realistic *meta-data* to the single configuration in the *C shortlist*. Thereafter the architects simply decide whether to enforce the suggested configuration. During the deployment, the granularity adaptation aspect used by this configuration is solicited from the *G bank* from the granularity adaptation aspect chooser. After enforcing the suggested configuration, its suitability in the environment described in *meta-data* can be observed to inform future *C bank* formations. For example, consider that the architects observe that the suggested configuration performs badly when deployed and the economic analyser indicates further granularity adaptation should be pursued. This observation can be used when forming the *C bank* by either: (1) attaching other *meta-data* to configurations similar to the one that performed badly or (2) discard configurations similar to the bad-performing one altogether. For the following iteration, the architects

provide input for the economic analyser and a new *C bank* to the next iteration which suits the newly deployed configuration.

5.4.3 Granularity Adaptation Aspect Chooser

If the *C shortlist* is empty, the granularity adaptation aspect chooser is triggered.

5.4.3.1 Granularity Adaptation Aspect Chooser Input

A bank of granularity adaptation aspects (*G bank* in Figure 5.1) pre-defined at design time driven by different utilities to be enhanced and/or beliefs about the system's runtime behaviour is fed by the architects into the planning engine. We assume that the code required to enforce each aspect is encoded in the initial architecture of Filmflix after the output of this component is provided. The *G bank* should include multiple aspects corresponding to the same utility (e.g., aspect A calling for decomposing a microservice to enhance performance and aspect B calling for merging microservices to enhance performance). This variety is required to capture the architects' uncertainty at design time regarding the suitable granularity adaptation strategy. We acknowledge however that this bank would not be exhaustive since it could not encounter every runtime scenario that can arise in dynamic microservice environments. Nevertheless, we assume this bank would target what the architects perceive to be the most critical microservices whose granularity adaptation can have the most impact on the added value introduced to the microservice architecture.

Figure 5.2 shows an example of a granularity adaptation aspect which calls for decomposing a microservice to isolating likely design vulnerabilities in the microservice architecture. It considers the failure rate of a microservice ambient (*"self"*) as the variable indicating the presence of a design vulnerability in one its children architectural elements (*"mem1" and "mem2"*). Figure 5.2 calls for monitoring the failure rates of *"mem1" and "mem2"* using the *getReading* interface service imported from the *IMonitor* interface. In cases where the failure rate of *"mem1"* is greater than a pre-defined *"FailureRateThreshold"*, the granularity adaptation aspect triggers the *DECOMPOSE(self)* transaction to form a new microservice ambient (*"vulnerable"*) that encapsulates *"mem1"*. The *Valuation* section of Figure 5.2 ensures the monitoring

```
Granularity Aspect VulnerIsolation using IMonitor

Attributes
Sources: Set (Source);
SourceNum:Integer
FailureRateThreshold:Integer
....
Transactions
        DECOMPOSE (input AESet: Set(ArchitecturalElement))
        MERGE (input AESet: Set(ArchitecturalElement))
Valuations
        [self.addChild(vulnerable)] Sources.add(new Source("vulnerable",vulnerable,new
Parameter("FailureRate",Integer,[])));
        [vulnerable.addChild(mem1)] Sources.delete (mem1);
        [vulnerable.addChild(mem1)] SourcesNum=SourceNum-1;
Triggers
        DECOMPOSE({self})
when
        Sources.ParamIter().includes("FailureRate") &
        getReading(self,"mem1","FailureRate")>=FailureRateThreshold
End_Aspect
```

Fig. 5.2 An example granularity adaptation aspect that triggers decomposing *"self"* by encapsulating the functionality of *"mem1"* into a new microservice ambient *"vulnerable"* if the monitoring the *"FailureRate"* of *"mem1"* indicates the failure rate is *geq "FailureRateThreshold"*

hierarchy is updated after the new ambient is formed. All the configurations in the *C bank* use granularity adaptation aspects from the *G bank*. However, not all the aspects in the *G bank* are used by configurations in the *C bank*. All the architectural configurations in the *C bank* fulfil the functional requirements of the target scope.

5.4.3.2 Granularity Adaptation Aspect Chooser Process

This component filters the *G bank* in two stages. First it short-lists the bank members which are concerned with the same runtime evidence variables that the economic analyser used for that iteration as runtime. Next, this short list is further narrowed down to a shorter list (*G shortlist* in Figure 5.1) suitable granularity adaptation aspects whose triggers are met; *G shortlist* could be empty.

5.4.3.3 Granularity Adaptation Aspect Chooser Output

Output Case 3: If the *G shortlist* has one aspect, the planning engine's output is a textual representation of that aspect. In this case, the runtime evidence variable values

confirm that one of the pre-defined triggers are indicative of the runtime microservice(s)'s behaviour to inform designing future *G bank* members. For example, if a microservice's response time never exceeds four milliseconds but occasionally exceed two milliseconds, then future *G bank* members need not include a trigger threshold of four milliseconds but should include a trigger threshold of two milliseconds. If architects decide to use the suggested aspect, then they design and encode an architectural configuration that uses it. The economic analyser input and *C bank* of the following iteration are re-defined according to the deployed configuration while the *G bank* remains unchanged and the *C bank* is extended with the designed architectural configuration.

Output Case 4: If the *G shortlist* has more than one aspect, the planning engine's output is their textual representations. The significance of the planning engine in this case is objectively pruning the solution space of possible granularity adaptation aspects by objectively comparing the runtime evidence variables against the triggers in the *G bank* rather than relying on static design time judgements to produce the *G shortlist*. Thereafter, the architect(s) explore the *G shortlist* (manually or in a simulated environment) to choose one of its members. Ultimately, the architects enforce an architectural configuration that uses the chosen member from the *G shortlist*. Pruning reduces the manual work required by the architects since only a single architectural configuration needs to be designed in this case. The economic analyser input and *C bank* of the following iteration are re-defined according to the deployed configuration while the *G bank* remains unchanged and the *C bank* is extended with the designed architectural configuration.

Output Case 5: If the *G shortlist* is empty, the output is a tuple of the observed workload value and corresponding runtime evidence variable(s) value(s). In this case the planning engine exposes the architects to an instance of emergent behaviour which they have not considered at design time. This is a natural implication of the highly dynamic environment in which microservice(s) operate. Therefore, it is inevitable that they can not accurately predict at design time the runtime behaviour of the target microservice(s). Given the output tuple, the architects need to re-visit the C and G banks. This includes: 1) defining a new granularity adaptation aspect with altered

trigger thresholds and 2) designing an architectural configuration that uses this aspect and 3) enforcing the newly incepted configuration. In this case, the *G and C banks* to the following iteration are extended to include the newly defined aspect. Extending rather than re-filling the *G and C banks* is a cautious approach to against aggressively discarding aspects and configurations that can become useful in the future iterations. The economic analyser input can re-defined for the next iteration as with output cases 2-4.

It is worth noting that it is not possible to directly conduct our evaluation process on the *C and G banks* since this would require a prohibitively costly exercise of designing architectural configurations utilising all the *G bank* members then running and observing them in parallel along with all the members of *C bank*.

5.5 EVALUATION

We apply the planning engine to the AWS Lambda implementation of Filmflix from Chapter 4 in different controlled scenarios and then use them to evaluate the architectural configuration chooser (Section 5.5.2) and granularity adaptation aspect chooser (Section 5.5.3). Since our dynamic evaluation process comprises the economic analyser, we summarise results of the evaluation in Chapter 4 to show the economic analyser's benefits.

For the other two components, our evaluation shows that for each controlled experiment, there is at least partial guarantee that the suggested configuration or aspect(s) can introduce more added value than the initial architecture. We also discuss the stability of each chooser in terms of its ability to withstand changes in *C and G banks*. Assessing the stability in this white-box, pipelined manner — where each component is examined separately — allows pinpointing the potential sensitivity points in the planning engine more accurately.

Overall, our evaluation helps show the effectiveness of our planning engine in providing decision support that can help realise microservice granularity adaptation at runtime; effectiveness is the property we aimed for in Research Question 4 of the thesis: How can microservice granularity adaptation be effectively engineered at

runtime, combining systematic modelling of granularity adaptation strategies with objective reasoning about added value?

5.5.1 Economic Analyser Evaluation

Since our dynamic evaluation process constitutes the economic analyser, evaluation results from Chapter 4 (Section 4.5) provide tangible evidence that the economic analyser 1) links runtime granularity adaptation decisions to their added value under uncertainty (Section 4.5.2), 2) goes beyond traditional application of real options by revealing the impact of runtime variations in workload on the added value which can be enabled adapting (or retained by not adapting) granularity (Section 4.5.3), 3) it is feasible to implement the economic analyser to assist microservice adopters in industry (Section 4.5.4) and, 4) the evaluation process is most tolerant of inaccurate estimations in QoS thresholds of the claim representing architects' expectations of the runtime system behaviour and the risk-free interest rate representing possible market fluctuations (Section 4.5.5).

5.5.2 Architectural Configuration Chooser Evaluation

In Section 5.5.2.1, Figures 4.3 and 4.4 are used to illustrate a scenario where the architectural configuration chooser would be triggered and an example *C bank* is presented to illustrate output case 2 from Section 5.4. We then use this scenario to evaluate the architectural configuration chooser along the following dimension: does it suggest an architectural configuration which can introduce more added value than the initial architecture (Section 5.5.2.2). We then discuss the architectural configuration chooser's stability in response to changes in the input *C bank*.

5.5.2.1 Experimental Setup

Consider iteration 5 of Figures 4.3 and 4.4 as an example, where the economic analyser output explicitly shows the total real option value which can be enabled adapting to be higher than not adapting. This iteration corresponds to the spike of invocation duration in 20:00:00.25 of Figure 4.3. This spike was triggered when the review string was 80 sentences long; we suspect that the spike was due to the

complexity of this string which required a long review of it. Therefore, the economic analyser outputs the verdict that the adaptation should be pursued for this iteration to introduce the corresponding total potential real option value and provide better invocation duration. Subsequently, the architectural configuration chooser should be consulted in this iteration. Given that adaptation is to be pursued in iteration 5, the architectural configuration chooser is consulted given a *C bank* which is defined by the architects at design time. In the subsections below, we present a *C bank* example to illustrate output case 2 from Section 5.4.

The *C bank* considered here includes two members, both with same design (visually shown in Figure 5.3). It is motivated by encapsulating ReviewRegulation and ReviewUpload in one microservice since they are expected to communicate frequently. Each member is annotated with non-overlapping ranges of workload values (review strings 0-50 and 51-100 sentences long respectively). In other words, the architects are uncertain whether the configuration in Figure 5.3 is more suitable for review strings 0-50 or 51-100 sentences long. Therefore, they include both range estimations in the *C bank*. The intuition behind these range choices is to equally split the range of possible string lengths below the threshold articulated in the claim across the *C bank* members. It is the planning engine's role to inform the architects using runtime observation of the workload values which range is more representative of the microservice runtime environment.

At iteration 5, MovieReview received a review that was 80 sentences long. When the architectural configuration chooser in consulted in iteration 5, it compares the observed value against the meta-data in the *C bank*. The observed workload value falls within *meta-data* range. Hence, the *C shortlist* in this case includes one configuration, namely Figure 5.3 annotated with "51-100 sentences". Thereafter, the architects deploy this configuration at the end of iteration 5. Furthermore, they are now aware that (at least temporarily) the workload values are more likely to fall into the 51-100 range and they can use this to inform future *C bank* compilations.

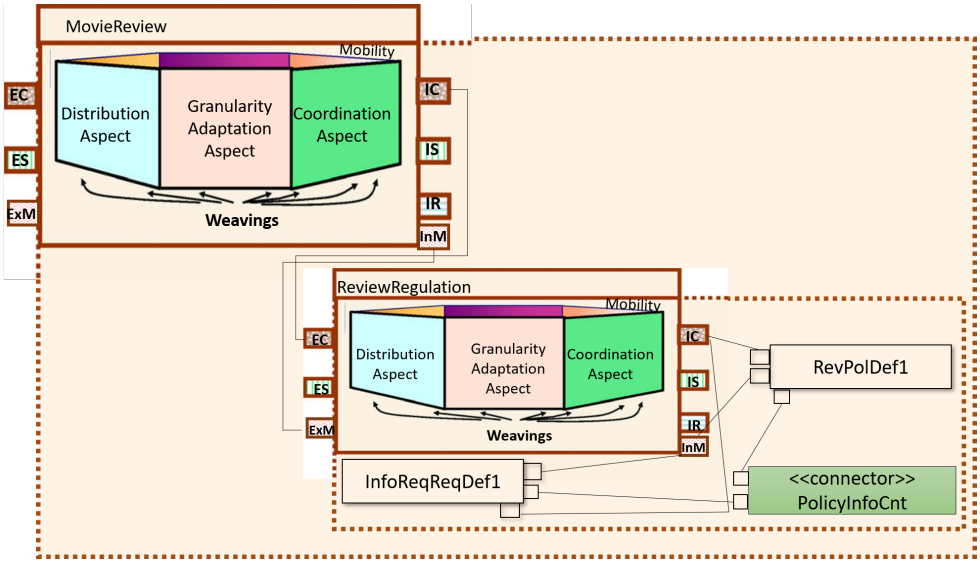


Fig. 5.3 Candidate architectural configuration if the initial Filmflix architecture is to be adapted;the *C bank* includes replicas of this candidate each annotated with different workload range values

5.5.2.2 Results: Suggesting an Architectural Configuration with more Added Value

We use our dynamic evaluation process from Chapter 4 to compare the total real option values in the initial architectural configuration against those in Figure 5.3 when exposed to the same workload trend. The total real option values for both configurations are plotted in Figure 5.4. For the initial configuration, we use the same inputs to the binomial analysis here as those fed to the economic analyser in Section 4.5.1 for the "not adapting" case. For the merged configuration, some of the inputs to the analysis are the same as those used for the initial configuration and some differ. We justify the differences and similarities for these inputs as follows:

- Utility tree: We use the same adapting utility tree here as used in the economic analyser for Section 4.5.1. Intuitively, the utility tree corresponding to adaptation captures the value of enforcing Figure 5.3.

- Option expiry date: The same expiry date as the initial configuration is used for fair comparison. We assume the options for both configurations expire after the same number of timestamps. The intuition is that we need to compare the real options within the same fixed window in both configurations.

- Monitored microservice: The same microservice is monitored as in the initial configuration.
- Monitored QoS attribute: For fair comparison, we use the same runtime evidence variable for the initial and merged configurations — invocation duration.
- Workload: the same review input strings are fed into both configurations for fair comparison of their added value under the same workload trend.
- Claim: Looking at the merged configuration, we initially consider the following claim: The invocation duration is > 0.5 milliseconds $\leftrightarrow$ review string length > 100 sentences. Comparing this to the claim for the initial configuration, we decided to use this claim which has a more strict view of the system to ensure that even under such a view the adapted configuration has more added value.
- Claim prior probability: Since the architect has no prior observation of the merged configuration and the claim remains unchanged in Section 4.5.1, the claim prior probability is 0.5 for both the initial and adapted configurations.
- Tolerance threshold: The same QoS is monitored for both configurations and the tolerance threshold is dependent on the criticality of the QoS attribute. Therefore, the same Bayesian surprise tolerance threshold should be used in both cases.
- Risk free interest rate: This is an indicator of the expected fluctuation in the system's runtime environment. Since both configurations are deployed in the same environment, the same risk free interest rate is used.
- Initial architectural value: Since the merged configuration has less logical dependencies, we consider it has a higher initial system value than the initial configuration.
- Maintenance costs: Since the merged configuration has less logical dependencies, we consider it has a lower maintenance cost than the initial configuration.
- Re-structuring cost: Since the merged configuration has less logical dependencies, we consider it has a lower re-structuring cost than the initial configuration if it were to be adapted.

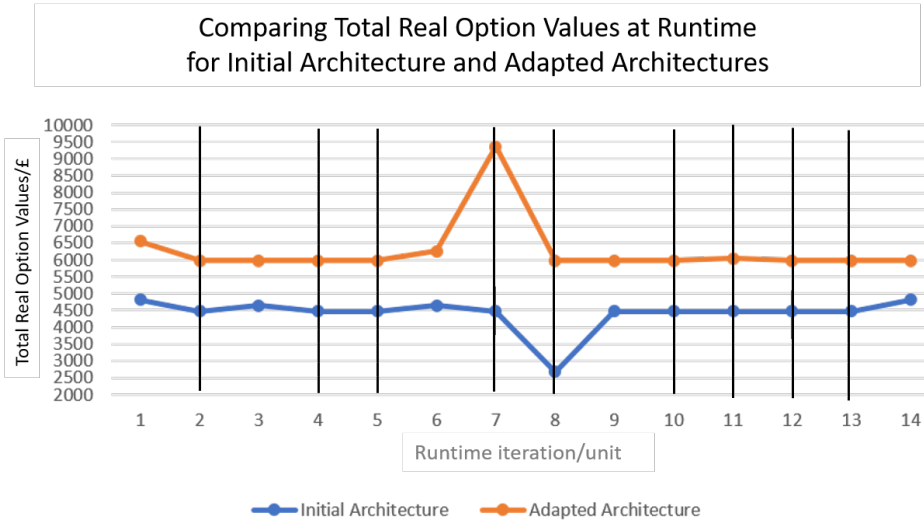


Fig. 5.4 Output of our dynamic evaluation process when applied to the initial Filmflix architecture and Figure 5.3; the black lines indicate when the process suggested adapting the initial architecture

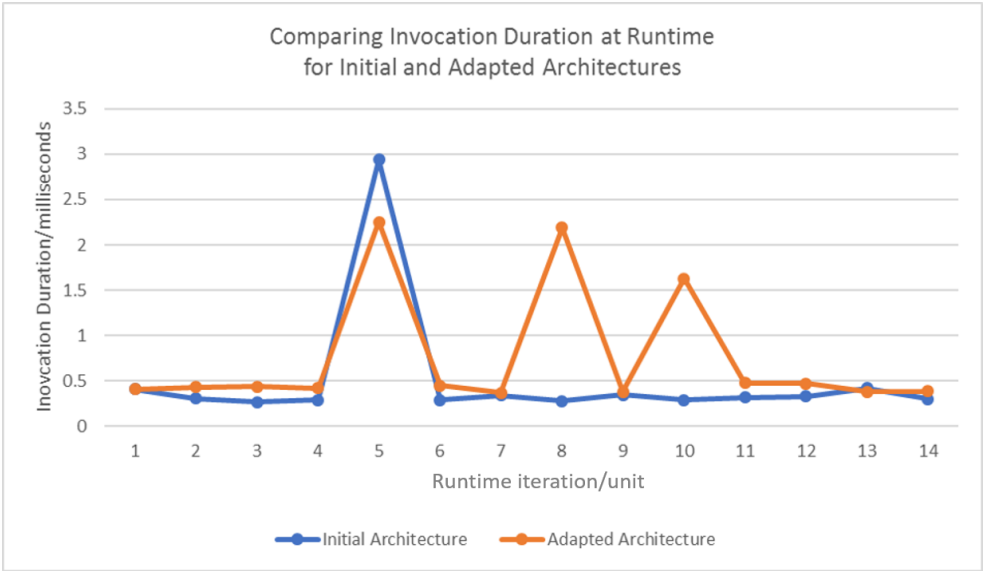


Fig. 5.5 Comparing invocation durations for the initial Filmflix architecture and Figure 5.3 used to plot Figure 5.4

Figure 5.4 shows the result of our dynamic evaluation process given the setting described above and Figure 5.5 shows the invocation durations for both configurations given the same workload. Overall, the total real option value which can be enabled by the merged configuration suggested by our planning engine is higher than the initial configuration. *Therefore, our architectural configuration chooser did propose a granularity adaptation decision which can enable more added value than the initial*

configuration. Moreover, the vertical black lines in Figure 5.4 show the iterations where the evaluation process suggested adapting the initial architecture. In 9 out of the 14 iterations, economic analyser output allowed the architects to benefit from the added value enabled by the architectural configuration chooser's output. It is worth noting here that we do acknowledge the possibility of different results given a different controlled experiment setting (e.g., by having claim thresholds that would be met by all architectural configurations in the *C bank*). This is inevitable however due to the nature of our contribution where it is infeasible to compare all the possible candidate architectural configurations against the initial architecture. Our controlled experiment does however provide a partial guarantee for the benefit of this component of the planning engine.

5.5.2.3 Results: Stability in Response to *C bank* Changes

Reflecting on Section 5.5.2.1, we can conclude that the combination of workload ranges comprising the *meta-data* within a *C bank* is the sensitivity point in the architectural configuration chooser. This sensitivity point reflects the software architects' uncertainty regarding the expected workload the configuration will be exposed to. Thereafter, this variation in meta-data is necessary to realise the advantages of the planning engine presented in output case 2 in Section 5.4. When the planning engine confirms design time expectations regarding expected workload this is in essence a reduction of uncertainty regarding expected workload.

5.5.3 Granularity Adaptation Aspect Chooser Evaluation

In this section, we consider a scenario where the *C shortlist* is empty when the planning engine is applied to the Filmflix implementation. Hence, the granularity adaptation aspect chooser is consulted; in Section 5.5.3.1 we present *G bank* examples to illustrate output cases 3,4 and 5 respectively. We then use this scenario to evaluate granularity adaptation aspect chooser along the following dimension: does it suggest a granularity adaptation aspect(s) which can introduce more added value than the initial architecture (Section 5.5.3.2). We then discuss the granularity adaptation aspect chooser's stability in response to changes in the input *G bank*.

5.5.3.1 Experimental Setup

5.5.3.1.1 Planning Engine Output Case 3 The *G bank* members below (Aspects 1 and 2) which are defined at design time both promote merging when invocation duration exceeds a certain threshold; the thresholds differ however. Both members are encoded in the architecture to be triggered when suitable at runtime. At runtime, the planning engine solicits and monitors the invocation duration of the running architecture and uses it to inform the architects when merging is suitable. In essence, the planning engine uses runtime observations to inform the architects which threshold is more indicative of the system's runtime behaviour to inform future *G bank* formations.

Aspect 1: Intuitively, this aspect is motivated by merging architectural elements whenever an overall performance deterioration is encountered (measured by the invocation duration of the parent microservice ambient — "self"). Underlying this intuition is that an invocation duration worse than three milliseconds in the parent microservice ambient implies unnecessary communication overhead across the children architectural elements ("mem1" and "mem2"). This implication is captured in the trigger of Figure 5.6 and sought for at runtime by monitoring the invocation duration of the parent microservice ambient (which is MovieReview in our initial architecture). The parent microservice ambient (self) utilises the getReading interface service for this monitoring. When the trigger is verified at runtime, merging mem1 and mem2 (ReviewUpload and ReviewRegulation in our initial architecture) into a single microservice ambient — "cluster" — is enforced. Subsequently, "self" would monitor "cluster" newly instead of monitoring "mem1" and "mem2" directly.

Aspect 2: This aspect takes a more cautious approach to the system's behaviour. In this aspect (Figure 5.7) the merging is triggered when the overall system performance deteriorates beyond two milliseconds. Underlying this stricter threshold is a higher expectation of the system's runtime behaviour.

Looking at the invocation duration of MovieReview in runtime iteration 5, only the trigger of aspect 2 is met (2.94 milliseconds). Therefore, *G shortlist* includes only Aspect 2. The architects then design and deploy the architectural configuration in Figure 5.3; it implements the transaction in Aspect 2. The related functional elements

```
Granularity Aspect Clustering using IMonitor

Attributes
Sources: Set (Source);
SourceNum:Integer
....
Transactions
        DECOMPOSE (input AESet: Set(ArchitecturalElement))
        MERGE (input AESet: Set(ArchitecturalElement))
Valuations
        [self.addChild(cluster)] Sources.add(new Source("cluster",cluster,new
Parameter("InvocationDuration",Double,[])));
        [self.addChild(cluster)] SourcesNum=SourceNum+1;

        [self.removeChild(mem1)] Sources.delete (mem1);
        [self.removeChild(mem1)] SourcesNum=SourceNum-1;

        [self.removeChild(mem2)] Sources.delete(mem2);
        [self.removeChild(mem2)] SourcesNum=SourceNum-1;
Triggers
        MERGE({mem1,mem2})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 3

End_Aspect
```

Fig. 5.6 Planning engine output case 3, Granularity adaptation aspect 1

```
Granularity Aspect Clustering using IMonitor

....
Triggers
        MERGE({mem1,mem2})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 2

End_Aspect
```

Fig. 5.7 Planning engine output case 3, Granularity adaptation aspect 2; it uses the same attributes and valuations as in Figure 5.6

are now encapsulated by a single microservice ambient at the end of runtime iteration 5.

5.5.3.1.2 Planning Engine Output Case 4

The *G bank* members below have the same triggers but promote merging in different manners. Moreover, the trigger takes a very cautious approach to the system's behaviour.

Aspect 1: In this aspect (Figure 5.8) the merging is triggered when the overall system performance deteriorates beyond one millisecond. Underlying this stricter threshold is an even higher expectation of the system's runtime behaviour than Figure 5.7.

```
Granularity Aspect Clustering using IMonitor
....
Triggers
        MERGE({mem1,mem2})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 1

End_Aspect
```

Fig. 5.8 Planning engine output case 4, Granularity adaptation aspect 1; it uses the same attributes and valuations as in Figure 5.6

```
Granularity Aspect Clustering using IMonitor
Transactions
        DECOMPOSE (input AESet: Set(ArchitecturalElement))
        MERGE (input AESet: Set(ArchitecturalElement))
Valuations
        [self.removeChild(mem1)] Sources.delete (mem1);
        [self.removeChild(mem1)] SourcesNum=SourceNum-1;

        [self.removeChild(mem2)] Sources.delete(mem2);
        [self.removeChild(mem2)] SourcesNum=SourceNum-1;

Triggers
        MERGE({mem1,mem2,self})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 1

End_Aspect
```

Fig. 5.9 Planning engine output case 4, Granularity adaptation aspect 2

Aspect 2: This aspect (Figure 5.9) calls for merging the architectural elements of the initial architecture into the single parent microservice MovieReview ("self") if the performance deteriorates beyond one millisecond. Therefore, no new microservice will be created if this aspect is executed. Moreover, this merge will require changes in the monitoring hierarchy of ambients to be captured when defining the MERGE transaction.

Looking at the invocation duration of MovieReview in runtime iteration 5, the triggers for aspects 1 and 2 are both met. The *G shortlist* therefore includes both aspects. The architects further explore this shortlist in terms of the refactoring overhead after which they proceed with aspect 1. Ultimately, they design and deploy Figure 5.3 at the end of iteration 5.

5.5.3.1.3 Planning Engine Output Case 5 The *G bank* members below promote opposite granularity adaptation strategies when the same trigger is met. At runtime,

the planning engine solicits and monitors the invocation duration of the running architecture and uses it to inform the architects whether this trigger is realistic or not. If it is not realistic, the *G shortlist* is empty; instead the planning engine outputs a tuple of the observed workload and corresponding runtime evidence variable to inform the architects in re-visiting this trigger. In essence, planning engine here would be calling for re-visiting all the strategies in the *G bank* since they are not realistic.

Aspect 1: Similar to Figure 5.6, merging the child microservice ambients "mem1" and "mem2" into a single microservice ambient called "cluster" is triggered here if the invocation duration is beyond four milliseconds.

```
Granularity Aspect Clustering using IMonitor
...
Triggers
        MERGE({mem1,mem2})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 4

End_Aspect
```

Fig. 5.10 Planning engine output case 5, Granularity adaptation aspect 1; it uses the same attributes and valuations as in Figure 5.6

Aspect 2: On the contrary, this aspect (Figure 5.11) is motivated by decomposing a microservice when the overall system performance deteriorates beyond four milliseconds. Underlying this intuition is that overall performance deterioration indicates unrelated functionalities being wrongly clustered together in the initial architecture. Additionally, an assumption is made here that the ReviewUpload microservice ("mem1" in Figure 5.11) is more likely to suffer this wrong clustering. Therefore, a deterioration of the invocation duration in MovieReview is an indicator that the DECOMPOSE transaction needs to be triggered on ReviewUpload.

Looking at the invocation duration of MovieReview in runtime iteration 5, the aspect chooser outputs the following tuple: (80 sentences, 2.94 milliseconds). When defining the triggers of aspects 1 and 2 at design time, the architects put too relaxed thresholds in the triggers.

Architects revisit aspect 1 to tighten invocation duration threshold which would trigger the merge transaction from four to 2.5 milliseconds (Figure 5.12). Upon

```
Granularity Aspect Clustering using IMonitor

Attributes
Sources: Set (Source);
SourceNum:Integer
....
Transactions
        DECOMPOSE (input AESet: Set(ArchitecturalElement))
        MERGE (input AESet: Set(ArchitecturalElement))
Valuations
        [self.addChild(cluster)] Sources.add(new Source("cluster",cluster,new
Parameter("InvocationDuration",Double,[])));
        [self.addChild(cluster)] SourcesNum=SourceNum+1;

        [cluster.addChild(mem1)] Sources.delete (mem1);
        [cluster.addChild(mem1)] SourcesNum=SourceNum-1;
Triggers
        DECOMPOSE({mem1})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 4

End_Aspect
```

Fig. 5.11 Planning engine output case 5, Granularity adaptation aspect 2

revisiting aspect 2 the architects decide that they do not have enough evidence to support the assumption that ReviewUpload would always be the source of performance deterioration. Therefore, they decide to discard this aspect altogether.

```
Granularity Aspect Clustering using IMonitor

Attributes
Sources: Set (Source);
SourceNum:Integer
....
Transactions
        DECOMPOSE (input AESet: Set(ArchitecturalElement))
        MERGE (input AESet: Set(ArchitecturalElement))
Valuations
        [self.addChild(cluster)] Sources.add(new Source("cluster",cluster,new
Parameter("InvocationDuration",Double,[])));
        [self.addChild(cluster)] SourcesNum=SourceNum+1;

        [self.removeChild(mem1)] Sources.delete (mem1);
        [self.removeChild(mem1)] SourcesNum=SourceNum-1;

        [self.removeChild(mem2)] Sources.delete(mem2);
        [self.removeChild(mem2)] SourcesNum=SourceNum-1;
Triggers
        MERGE({mem1,mem2})
when
        Sources.ParamIter().includes("Invocation Duration") &
        getReading(self,"self","InvocationDuration") > 2.5
End_Aspect
```

Fig. 5.12 Revisited Granularity Adaptation aspect due to observing emergent runtime behaviour

We consider for simplification that the newly derived aspect is enforced as Figure 5.3. In reality, further manual analysis can lead to enforcing it differently (e.g. different attachments, ports and weaving relationships) from Figure 5.3.

5.5.3.2 Results: Suggesting Granularity Adaptation Aspects with More Added Value

The analysis used in Section 5.5.2.2 can be applied in to evaluate the granularity adaptation aspect chooser given the simplified setting we considered in Section 5.5.3.1. All the aspects output by this components in Section 5.5.3.1 can result in Figure 5.3 which is compared to the initial architecture in Section 5.5.2.2. *Therefore, we can infer that the granularity adaptation aspect chooser does propose aspect(s) or alert toward re-visiting aspects in a manner than can ultimately introduce more added value that the initial architecture.* We acknowledge however that this inference is only a partial guarantee which needs can only be confirmed if all the possible architectural configurations utilising the suggested aspect(s) are compared to the initial architecture.

To pave the way towards that, we implement another architectural configuration (modelled in Figure 5.13) which fulfils the transactions in Figures 5.7, 5.8, 5.9 and 5.12. Then we use the same setting described in Section 5.5.2.2 and our dynamic evaluation process to compare the total real option values which are can be enabled by the initial configuration against those that can be enabled by Figure 5.13 when exposed to the same workload trend. The result of the comparison is captured in Figure 5.14.

Overall, the trend in total real option values which can be enabled by Figure 5.13 is same as for Figure 5.3. The black vertical lines in Figure 5.14 indicate where our dynamic evaluation process suggests adapting the initial architecture, thereby allowing reaping the benefits of the adapted configuration. Since the process is invoked on the same initial architecture in both Figures 5.4 and 5.14, it suggests adapting at the same runtime iterations in both Figures. The point-wise difference between the total real option values corresponding to the initial configuration and the adapted configuration in Figure 5.4 is larger. In runtime iteration 4 for example, the point-wise difference between the total real option values is 1325£ in Figure 5.4. However, the point-wise

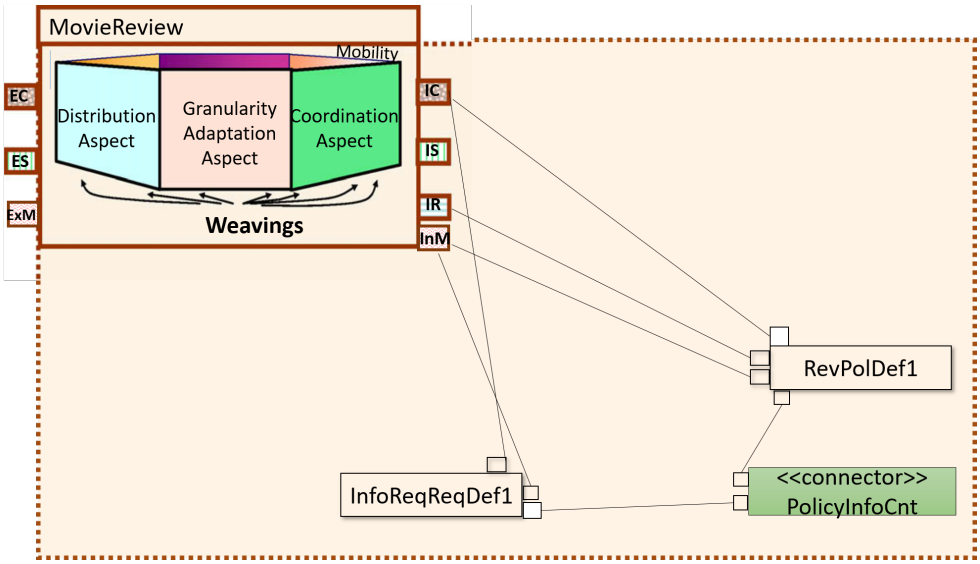


Fig. 5.13 Alternative architectural configuration that fulfils the transactions in Figures 5.7, 5.8, 5.9 and 5.12

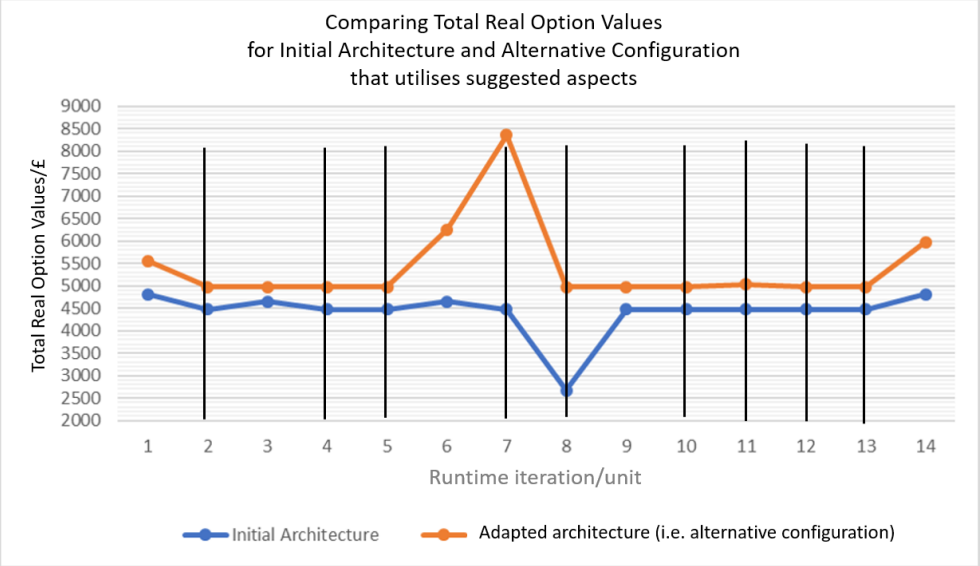


Fig. 5.14 The output of our dynamic evaluation process applied to the initial architectural configuration against Figure 5.13; the black lines indicate when the process suggested adapting the initial architecture

difference is 325£ in runtime iteration 4 of Figure 5.14. *Nevertheless, this comparison further strengthens our inference that the granularity adaptation aspect chooser suggests aspects that can ultimately be utilised by configurations which result in more added value than the initial architecture.* Similar to the comment in Section 5.5.2.2, we acknowledge that a different controlled experiment setting can lead to different results for this component's evaluation. This is inevitable due to the infeasibility of

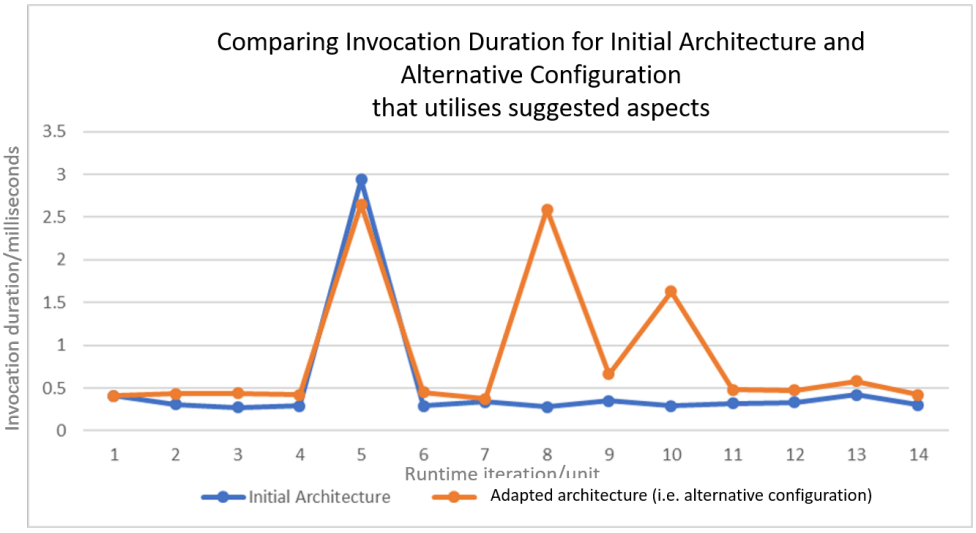


Fig. 5.15 The invocation duration of the initial architectural configuration and Figure 5.13 used to plot Figure 5.14

compiling all the possible granularity adaptation aspects which architects could think of regarding the initial Filmflix architecture. Our evaluation here shows an instance of the benefit of the granularity adaptation aspect chooser.

5.5.3.3 Results: Stability in Response to *G bank* Changes

Reflecting on Section 5.5.3.1, we can conclude that the combination of thresholds in the triggers on runtime evidence variables of concern is the sensitivity point in this component. For example, in *Output Case 3* the thresholds on invocation duration are varied across the *G bank* which led to the *G shortlist* including one *G bank* member but not the other. However, in *Output Cases 4 and 5*, the trigger thresholds are kept constant while the granularity adaptation strategies are varied which meant outputting or discarding both aspects respectively.

It is worth noting that high sensitivity is essentially a weakness in the economic analyser but a strength for the architectural configuration chooser and the granularity adaptation aspect chooser. The planning engine updates the software architects' uncertainty in an value-driven manner using runtime evidence to inform their decision-making. In the economic analyser this is achieved through a multi-parameter calculation — our dynamic evaluation process in Chapter 4 (summarised in Algorithm 1). These parameters include the runtime evidence values. Therefore, due to the multi-

parameter nature of the calculation, it is essential to ensure that the uncertainty update is genuinely due to fluctuations in runtime evidence rather than other parameters whose manual fluctuation do not reflect runtime evidence.

In the two other planning engine components however, the uncertainty update is achieved by examining *meta-data* and granularity adaptation aspect trigger thresholds respectively only given the runtime evidence. Therefore, we can guarantee that uncertainty updates in these components is due to runtime evidence fluctuations. The sensitivity points in these components serve rather than hurt the planning engine's role — updating the architects' uncertainty.

5.6 CONCLUSION

In this chapter we contribute to a runtime, interactive, iterative planning engine which acts as a decision support system for microservice adopters; it suggests granularity adaptation strategies that can enable added value to the microservice architecture (thereby addressing Research Question 4)

The planning engine takes as input the parameters needed for conducting our dynamic evaluation process and pre-defined banks of granularity adaptation aspects and annotated architectural configurations that may use these aspects. Members of both banks represent alternative granularity adaptation strategies.

The planning engine includes three components: an economic analyser, an architectural configuration chooser and a granularity adaptation aspect chooser. The economic analyser dictates whether the two other components need to be consulted or not for a runtime iteration. Consequently, the other two components are used to suggest suitable architectural configuration(s), granularity adaptation aspect(s) or to output a tuple representing an emergent behaviour scenario which architects were unaware of at design time.

We apply the planning engine to the AWS Lambda implementation of Filmflix from Chapter 4 in different controlled scenarios and then use them to evaluate the architectural configuration chooser and granularity adaptation aspect chooser.

The results of Chapter 4's evaluation show the economic analyser's benefits in linking runtime granularity adaptation decisions to their added value under uncertainty, revealing the impact of runtime variations in workload on the added value which can be enabled adapting (and not adapting) granularity and assisting microservice adopters in industry. We further discuss the stability of economic analyser output in response to changes in the the economic analyser inputs. The results show that the stability of the economic analyser can be guaranteed provided the architects level of confidence in their input estimations.

For the other two components, our evaluation shows that for each application scenario of the planning engine to Filmflix, there is at least partial guarantee that the suggested configuration or aspect(s) can introduce more added value than the initial architecture. We also discuss the stability of each chooser in terms of its ability to withstand changes in *C and G banks*. Our discussion shows that the choosers' outputs are highly sensitive to the input bank of granularity adaptation strategies. Nevertheless, we conclude that this sensitivity is essential to reap the advantages of the planning engine.

Along with stability, another orthogonal but equally important dimension of the planning engine is its ability to provide decision support in dynamic microservice environments even in large-scale microservice applications. The scale at which microservice applications operate gives rise to a further challenge which we address in Chapter 6: Without systematic guidance regarding the scalability dimensions and metrics that can affect reasoning about microservice granularity adaptation, it is not possible to efficiently accelerate its industrial adoption. This challenge calls for microservice-specific guidance to alert microservice architects regarding potential scalability obstacles as well as a systematic analysis approach to justify the criticality of these obstacles and to determine how they can be resolved.

Chapter 6

Microservice-specific Guidance for Scalability Analysis: Scalability-aware Decision Support for Microservice Granularity Adaptation

In Chapter 5, we contribute to a runtime, interactive, iterative planning engine aiming to provide microservice granularity adaptation decision support. It is essential that the planning engine (and potentially other systems that support reasoning about microservice granularity adaptation) considers the scale at which microservices operate since the main adopters of microservices are highly competitive industries which are characterised by scale. This necessity gives rise to Research Question 5: How is it possible to systematically analyse the extent to which a decision support system can render scalability-aware granularity adaptation decisions? Considering scale entails systematically analysing the dimensions and metrics which are important for the scalability of microservice architectures. Subsequently, these dimensions and metrics need to be considered by microservice granularity decision support systems to render scalability-aware granularity adaptation decisions. Scalability of the decision

support system itself also needs to be systematically analysed to assess the extent to which it can scale to consider all the dimensions which are critical for rendering scalability-aware granularity adaptation decisions.

To our knowledge, the state-of-the-practice in decision supports systems for microservice granularity adaptation either: 1) is unaware of the microservice-specific scalability dimensions and metrics or, 2) does not utilise systematic scalability analysis to render scalability-aware adaptation decisions. Moreover, the state-of-the-practice is short of a systematic approach to analyse scalability of the decision support systems themselves.

In this chapter, we address the aforementioned problems using a two-fold contribution. Firstly, we contribute to a working catalogue of microservice-specific scalability dimensions and metrics. Our catalogue helps identify dimensions and metrics which are important for the scalability of a given microservice architecture; they need to be considered in order to render scalability-aware granularity adaptation decisions for it. We compile our catalogue by: 1) digesting the state-of-the-practice in decision support systems for microservice granularity adaptation and, 2) reflecting on the sample strategies formulated in Appendix B to infer additional potentially important scalability dimensions and/or metrics. Secondly, we apply scalability goal-obstacle analysis [104, 91] in the context of reasoning about microservice granularity adaptation. Applying scalability goal-obstacle analysis to a given microservice architecture helps identify obstacles along each dimension of importance from our catalogue. Applying scalability goal-obstacle analysis to a given decision support system helps analyse the extent to which it can scale to consider all these dimensions and metrics.

Both contributions together aim to address Research Question 5. Our catalogue and application of scalability goal-obstacle analysis provide coherent guidance for: 1) identifying and systematically analysing the dimensions and metrics which are important for the scalability of a microservice architecture and, 2) systematically analysing the extent to which a decision support system can scale to consider all these dimensions and metrics.

We evaluate and discuss our contributions by comparing their usage to both initial Filmflix architecture in Section 1.1 and the planning engine against ad-hoc scalability

assessment of both cases. This aims to highlight how both contributions can aid rendering scalability-aware granularity adaptation decisions. Finally, we discuss how scalability-goal obstacle analysis can be applied to other microservice architectures and other decision support systems for microservice granularity adaptation.

6.1 INTRODUCTION

In this chapter, we address the inadequacy of a systematic approach for analysing the extent to which a decision support system can render scalability-aware granularity adaptation decisions (related to Research Question 5).

Among the main adopters of microservices are highly competitive industries (e.g., Netflix [268], Amazon [353], and BBC [145]) which are characterised by scale. Therefore, scalability of a microservice architecture is among the architecturally significant requirements of microservice adopters. Henceforth, it is essential that a decision support system for microservice granularity adaptation shall be aware of the dimensions and metrics that are important for the scalability of a microservices architecture. Subsequently, the decision support system should consider these dimensions and metrics to render a scalability-aware granularity adaptation decision. The inclusion/exclusion of these dimensions and/or metrics as input to a decision support system can render different granularity adaptation strategies. Examples of dimensions that are important for the scalability of a microservice architecture include: end user base size, the number of logical and physical dependencies across microservices, domain-specific factors (e.g., copyright costs for media content streaming) and the volume of data shared across microservices.

Addressing Research Question 5 entails: 1) identifying and systematically analysing the dimensions and metrics which are important for the scalability of a given microservice architecture and, 2) systematically analysing the extent to which a decision support system can scale to consider all these dimensions and metrics.

Currently however, there is an inadequacy of conscience among microservice adopters regarding the scalability dimensions and metrics that are important for a microservice architecture. Moreover, in the few attempts for rendering scalability-

aware granularity adaptation decisions, this has been only done ad-hocly (e.g. through a single case study or by merely discussing possible scalability dimensions and/or metrics that can affect granularity adaptation decisions). Furthermore, the state-of-the-practice is short of a systematic approach to analyse the scalability of the decision support systems themselves. The state-of-the-practice and -art therefore leaves room to address two problems:

Problem 1 — inadequacy of guidance regarding the dimensions and metrics which are important for the scalability of a microservices architecture and shall be considered to render scalability-aware granularity adaptation decisions: As scalability of a microservice architecture is one of the architecturally significant requirements of microservice adopters, it is essential that a decision support system shall be aware of the factors that are critical for the scalability of the microservices architecture itself. "The problem is that scalability is so dependent on the application domain and the system's goals that accommodating all dimensions in pre-defined categories is very challenging, if not impossible [91, p.18]." This challenge is critical in microservice architectures; a huge number of simultaneous inevitable scalability dimensions and metrics can affect the scalability of a microservice architecture.

Consider a fictional running microservice-based application with a functionality and scale similar to Netflix — called NetWatch. The dimensions affecting NetWatch's scalability include logical interdependencies across microservices and the volume of data shared across them. Further consider that microservice granularity adaptation decisions in NetWatch are supported by a fictional system — MicroAdapt; the input to MicroAdapt is determined by NetWatch architects. If the input to MicroAdapt only dictates it should only consider scalability with respect to logical interdependencies, then the granularity adaptation decisions suggested by MicroAdapt can be less scalability-aware and might lead to less added value for NetWatch. A relatively more scalability-aware decision would have been suggested if the input dictated considering scalability with respect to both the logical dependencies and shared data volumes. Informing the input to MicroAdapt requires guidance to identify and systematically analyse the dimensions and metrics important for the scalability of NetWatch.

Problem 2 — inadequacy of a systematic approach for systematically analysing the extent to which a decision support system can scale to consider all the dimensions which are important to render scalability-aware granularity adaptation decisions: Scalability dimensions and metrics of importance are in essence input parameters to decision support systems for microservice granularity adaptation. Therefore, it is essential to systematically analyse the extent to which a decision support system can scale with respect to these input parameters.

Consider the context above where MicroAdapt supports granularity adaptation of NetWatch. Systematically analysing the scalability of MicroAdapt can determine the extent to which can handle both logical dependencies and volume of shared data as input parameters to it. Consequently, appropriate action (e.g., refining the design of MicroAdapt) can be taken to ensure MicroAdapt can scale to handle the input necessary for rendering scalability-aware granularity adaptation decisions.

To address Problem 1, this chapter aims to aid: 1) identifying the microservice-specific dimensions and metrics which are important for the scalability of a microservice architecture and thereby essential for rendering scalability-aware granularity adaptation decisions and, 2) systematically identifying and analysing the obstacles along each identified dimension. These obstacles are indications of risks that can obstruct satisfiability of goals of interest to a microservice architecture. Henceforth, this systematic analysis justifies the importance of considering a given scalability dimension to render a scalability-aware granularity adaptation decision. To address Problem 2, this chapter aims to apply a systematic scalability analysis approach to the new context of decision support systems for microservice granularity adaptation.

The novel contribution of this chapter is two-fold:

- An attempt for a working catalogue of the microservice-specific scalability dimensions and metrics. We compile this catalogue by: 1) digesting the state-of-the-practice in decision support systems for microservice granularity adaptation and, 2) reflecting on the sample strategies formulated in Appendix B to infer additional potentially important scalability dimensions and/or metrics. This catalogue aims to aid identifying the microservice-specific dimensions and met-

rics which are important for the scalability of a given microservice architecture and thereby essential for rendering scalability-aware granularity adaptation decisions (partially addressing Problem 1).

- An application of scalability goal-obstacle analysis [104, 91] in the new context of reasoning about microservice granularity adaptation. Scalability goal-obstacle analysis was first attempted in [91, 103]; it is inspired by Keep-All-Objectives-Satisfied (KAOS) goal-oriented modelling [165]. The input to this analysis is a refined KAOS goal-oriented model of system. Consequently, scalability goal-obstacle analysis aims to systematically identify, assess and resolve potential obstacles which can obstruct the system from satisfying its goals if it were scaled along relevant dimensions. Scalability goal-obstacle analysis "attempts to establish a uniform notion of scalability that can be applied in a wide variety of application domains and can support analysis of scalability with respect to a wide variety of system qualities [103, p.383]." The generality of scalability goal-obstacle analysis makes it more applicable to our context than other domain-specific approaches that assess scalability. (e.g., [105], [169], and [170]). Systems in those domains are fundamentally different from those targeted in our context so scalability assessment approaches developed for these domains are non-transferable to our context. Furthermore, scalability goal-obstacle analysis by definition has distinct steps to identify, assess and resolve scalability-obstacles of a system. In this chapter, applying scalability goal-obstacle analysis to a given microservice architecture helps analyse obstacles along each important dimension taken from our catalogue (partially addressing Problem 1 above). Furthermore, applying scalability goal-obstacle analysis to a given decision support system helps analyse the extent to which it can scale to consider all the dimensions and metrics important to a microservice architecture it supports (addressing Problem 2 above).

Together both contributions coordinate to provide coherent, specialised guidance that aids analysing the ability of a decision support system to render scalability-aware granularity adaptation decisions (addressing Research Question 5).

On one hand, our catalogue acts as a pre-requisite for scalability goal-obstacle analysis. The catalogue alerts microservice architects regarding the important scalability dimensions and metrics thereby scoping the goal-obstacle analysis of a microservice architecture. Consequently, scalability goal-obstacle analysis identify and analyse obstacles along the dimensions from our catalogue. Ultimately, the dimensions and metrics which are deemed critical by the analysis should be provided as input to decision support systems to render scalability-aware granularity adaptation decisions.

On the other hand, applying scalability goal-obstacle analysis to decision support systems can analyse the extent to which they can scale to consider all the dimensions and metrics necessary for rendering scalability-aware granularity adaptation decisions.

In Section 6.2 we use the planning engine's application to Filmflix from Section 5.5 to motivate the need for this chapter's contributions. In Section 6.3 we report on our digestion of the state-of-the-practice literature and of strategies from Appendix B then present our microservice-specific catalogue of scalability dimensions and metrics. In Section 6.4 we provide an overview of KOAS goal-oriented modelling and scalability goal obstacle analysis. In Section 6.5, we evaluate and discuss our contributions by comparing their application to the initial Filmflix architecture from Section 1.1 against its ad-hoc scalability assessment; we carry out the same comparison for our planning engine from chapter 5. Both evaluation routes aim to illustrate how our catalogue and scalability goal-obstacle analysis can address Problems 1 and 2 above. In Section 6.6 we conclude by summarising the chapter contents.

6.2 MOTIVATING SCENARIO

Consider Section 5.5 where we apply the planning engine to the AWS Lambda implementation of Filmflix from Chapter 4 in different controlled scenarios. In Section 5.5, each constituent component of the planning engine takes the following inputs for a runtime iteration:

- The economic analyser takes a single set of parameters for our dynamic evaluation process presented in Chapter 4,

- The architectural configuration chooser takes a *C bank* which has two members for output case 2 of the planning engine,
- The granularity adaptation aspect chooser takes a *G bank* which has two members for each of output cases 3, 4 and 5 of the planning engine.

For each controlled scenario, granularity adaptation strategies are suggested driven by enhancing high performance provision in Filmflix, measured in terms of invocation duration.

The scale at which Filmflix operates in reality means it is inevitable that multiple scalability dimensions need to be considered by the planning engine to suggest a granularity adaptation strategy that can indeed introduce added value. For example, the volume of data shared across Filmflix microservices and the number of geographical locations served by Filmflix are two dimensions which can affect the scalability of Filmflix. Consequently, both dimensions need to be reflected in the inputs above. Claims capturing each scalability dimension need to be provided as input to the economic analyser. The *G and C banks* also need to consider both dimensions.

Informing the input will consequently aid rendering more scalability-aware granularity adaptation decisions. If only the volume of shared data is provided as input and thereby considered by the planning engine, the output might suggest merging microservices to reduce the data exchange calls across them and enhance the overall Filmflix added value. If only the geographical distribution of the end users is considered, the planning engine might suggest decomposing microservices so that functionalities related to each geographical area are encapsulated in the same microservice. Therefore, the planning engine needs to consider both dimensions simultaneously in order to suggest a scalability-aware granularity adaptation strategy. Guidance is therefore essential to identify the important scalability dimensions and metrics for Filmflix then identify and analyse potential obstacles for the satisfiability of Filmflix's goals if it is scaled along these dimensions.

Considering all the relevant scalability dimensions in the input however raises a challenge regarding scalability of the planning engine itself. For example, if multiple claims are provided as input to the economic analyser, then multiple sets of parameters

are required for each claim. Moreover, informing the input can mean much larger *C and G banks* would be provided as input to the planning engine. If it can not scale to consider these inputs, the planning engine can not converge to a scalability-aware granularity adaptation decision. It is therefore essential to systematically analyse the extent to which the planning engine can scale to handle these input parameters. Henceforth, appropriate action can be taken (e.g., refining the design of the planning engine) based to its scalability analysis results.

Given the scenario above, in this chapter we call for: 1) identifying the microservice-specific dimensions and metrics which are important for the scalability of a microservice architecture and thereby essential for rendering scalability-aware granularity adaptation decisions and, 2) systematically analysing the obstacles along each identified dimension. These obstacles are indications of risks that can obstruct satisfying goals of interest to a microservice architecture. Henceforth, this systematic analysis justifies the importance of considering a given scalability dimension to render a scalability-aware granularity adaptation decision. Furthermore, we call for applying a systematic scalability analysis approach to the new context of decision support systems which are aimed for microservice granularity adaptation.

We argue it is sensible to derive a catalogue of microservice-specific scalability dimensions and metrics from microservice-specific literature and strategies from Appendix B to render specialised guidance fit for addressing Problem 1 in Section 6.1. The generality of scalability goal-obstacle analysis [103] makes it flexible enough to be applicable in our context as opposed to other domain-specific scalability assessment approaches. For example, state space size is a scalability indicator of techniques used to tackle state explosion in model checking [105]. Effectiveness calculated as a function of throughput and QoS is an indicator of scalability in parallel computing [169]. Cost-effectiveness of scalability calculated as a function of the system's power and cost of this power is a scalability indicator in distributed systems [170]. Despite their power in the domains they target, these domains are fundamentally different from microservice architectures and decision support systems for microservice granularity adaptation. Moreover, scalability goal-obstacle analysis by definition has distinct steps (explained in Section 6.4) making it fitting to for Problems 1 and 2 from

Section 6.1. Both contributions aim to provide coherent, specialised guidance that aids analysing the ability of a decision support system to render scalability-aware granularity adaptation decisions (addressing Research Question 5).

6.3 WORKING CATALOGUE OF MICROSERVICE-SPECIFIC SCALABILITY DIMENSIONS AND METRICS

In this section we report on our digestion of the state-of-the-practice literature (Section 6.3.1) and strategies from Appendix B (Section 6.3.2) then present our microservice-specific catalogue of scalability dimensions and metrics (Section 6.3.3). Our digestion of microservice-specific literature and complementing that with strategies from Appendix B show our best effort to make the catalogue as specific yet comprehensive as possible.

6.3.1 State-of-the-practice Scalability Dimensions

Tables 6.1 and 6.2 summarise whether and how scalability is considered when rendering granularity adaptation decisions. The publications summarised in Table 6.2 and 6.1 are representative examples yielded from our systematic mapping study in Chapter 2; they are selected from the "processes" in Figure 2.6. In essence, these processes help reason about microservice granularity adaptation so it is fitting to use them as a basis for deriving the dimensions and metrics of our catalogue. We manually examined the remaining "processes" in Figure 2.6 to ensure that Table 6.2 is indeed representative of how scalability is assessed in publications from Figure 2.6. We further categorise the representative examples according to their density; Table 6.1 summarises publications which present a set (rather than a single) of microservice architectural design patterns that can support granularity adaptation decisions. Along with each pattern we summarise the scalability dimensions and/or metrics which can affect adopting them. When compiling Table 6.1, we made our best effort to not duplicate patterns that have been mentioned in multiple publications. On the other hand, Table 6.2 summarises publications where a single "process" that supports reasoning about granularity adaptation decisions is presented.

Where the "process" (i.e. decision support system) strives for scalability-aware decisions, the dimensions and/or metrics considered are presented in last two columns of Tables 6.1 and 6.2. A scalability dimension can be regarded as any characteristic of the system design and/or its running environment which can exhibit a wide variation in values during the system's lifetime [91]. Scalability metrics on the other hand can be regarded as any measurable or computable variables that can measure the ability of a system to meet its goals even when it is stressed along one or more scalability dimensions [169].

Decision Support System	Scalability Considered	Scalability Assessment Approach	Pattern Supporting Granularity Adaptation Decisions	Scalability Dimensions Considered	Scalability Metrics Considered
Based on industrial surveys and interviews, a brief catalogue of microservice-specific bad smells "which negatively affect software quality attributes such as understandability, testability, extensibility, reusability, and maintainability of the system under development [331, p.59]." The solutions adopted to fix these bad smells are in essence drivers for microservice granularity adaptation.	✓	Discussion	Semantic versioning of APIs	number of codebase versions	
			API gateways	number of cyclical dependencies across microservices, number of interdependent microservices	
			Lightweight communication mechanism adoption	volume of enterprise service buses connecting microservices	
			Microservice discovery adoption	number of available IP addresses	
			Merging microservices accessing same databases	volume of data shared across microservices	
			Polyglot persistence	volume of data shared across microservices	
			Shared library extraction	number of libraries shared across microservices	
			Systematic standardisation of development tools	number of development technologies used	
			"Clear analysis of business processes and the need for resources [331, p.60]"	number of cross-cutting business capabilities	
Microservice architecture design patterns some of which affect the granularity of microservices (e.g., bulkheads); the process of adopting each pattern and the considerations related to it are discussed[191]	✓	Discussion	Service registry/Self-registration/Third-party registration	Number of interdependent microservices	
			API Keys and two-factor authentication	security costs	

Decision Support System	Scalability Considered	Scalability Assessment Approach	Pattern Supporting Granularity Adaptation Decisions	Scalability Dimensions Considered	Scalability Metrics Considered
			Centralised and decentralised logging	logging costs	
			Command Query Responsibility Segregation (CQRS)	volume of data shared across microservices	
			Multiple instances per host/single server instance per host/single server per virtual machine/single instance per container	containerisation/ virtualisation costs	
			Message broker	caching costs	
A guide book of the best practices for architecting microservice-based systems, including how to modernize legacy applications into microservice architectures; this transition drives granularity adaptation decisions[363]	✓	Discussion	Message routing	Network latency across microservices	time of convergence to an adaptation decision, ease of independent development after pursuing adaptation, system throughput/response time
			DevOps and NoOps	operational and infrastructure costs	
			Immutable server/installation scripts	deployment pipeline costs, number of interdependent deployment configuration setting	
			Sidecar	number of development technologies used, number of interdependent teams working on the architecture, number of codebase versions	
			Reference environments/ stubs/ consumer-driven contract tests	number of integration test cases spanning multiple microservices, number of consumer-driven contracts to be met	
			Internal and external interfaces	number of user-facing interfaces	
			Containerisation of service registries	number of microservice registries	
			Transferring/Extracting shared functionalities	number of shared libraries across microservices	

Decision Support System	Scalability Considered	Scalability Assessment Approach	Pattern Supporting Granularity Adaptation Decisions	Scalability Dimensions Considered	Scalability Metrics Considered
			Client-based load balancing/service discovery/Load balancer per microservice	number of load balancers per microservice	
			Team-based documentation/ Microservice-based documentation/Documentation versioning/Microservice templates	volume of documentation	
			Edge-side Includes/Server-side Includes	number of front-end servers	
			Content enricher	number of light-weight communication mechanisms used	
			Event sourcing	number of event buses	
			Circuit breakers	monitoring costs	
			Database replication	number of independent databases, volume of queries received by the data store	
A guide book for how to design and architect applications for maximum uptime, performance, and return on investment; these properties are common drivers of microservitization and henceforth they impact granularity adaptation decisions. Therefore the guidance in this book can be regarded as a process supporting granularity adaptation decisions [238]	✓	Discussion	In-process method calls/Interprocess communication/Remote procedure calls/message-oriented middleware/tuple spaces	Number of interdependent middleware	
			Bulkheads	volume of multi-threading	
			Database clustering	number of available virtual IP addresses	

Decision Support System	Scalability Considered	Scalability Assessment Approach	Pattern Supporting Granularity Adaptation Decisions	Scalability Dimensions Considered	Scalability Metrics Considered
A presentation of the current solutions for modelling, integrating, testing, deploying, and monitoring microservices; all these dimensions affect or are affected by granularity adaptation decisions hence the presentation in this book can act as procedural guidance for granularity adaptation decision-making[233]					
A series of microservice decomposition strategies mostly based on isolating independent bounded contexts[276]	✓	Discussion	Sagas for managing data transactions	Volume of data transactions	
			Request/Asynchronous One-way (i.e. notifications)	volume of inter-microservice communications	
			Publish/Subscribe and Publish/Asynchronous responses	volume of published events	
			Decomposition by domain, sub-domain or scenarios	number of bounded contexts	
			Request/response messages	volume of requests	
A presentation of best practices that can ensure QoS provision through the microservitization process; some patterns can drive granularity adaptation decisions (e.g., clustering services according to the client zone they serve)[265]	✓	Discussion	Service routing into zones	geographical spread of end user base	

Table 6.1 Summarising whether and how scalability is considered when representative examples from the "processes" in Figure 2.6 (i.e. decision support systems) suggest granularity adaptation decisions; the examples in this table present a set of patterns and/or best practices that can entail granularity adaptation; for each pattern the scalability dimensions which are deemed relevant to it (as discussed in the respective publication) are presented

Decision Support System	Scalability Considered	Scalability Assessment Approach	Scalability Dimensions Considered	Scalability Metrics Considered

Decision Support System	Scalability Considered	Scalability Assessment Approach	Scalability Dimensions Considered	Scalability Metrics Considered
A case study describing an industrial experience for extracting services (i.e. making granularity adaptation decisions) from a monolith and transforming them into microservices[256]				
An industrial experience reporting the process of data-driven granularity adaptation decisions[257]				
An industrial experience reporting on a monolithic subsystem migration to microservices[348]	✓	Case study application	End user base size, number of countries the application is serving, audit compliance considerations, data transport and synchronisation costs	downtime rate after pursuing adaptation
An industrial experience advocating for microservitization through "Everything-as-a-Service"; an approach where granularity adaptation decisions start from identifying and preparing for the slow moving tasks in the application then transforming the technology stack in accordance to this preparation[162]	✓	Discussion	technology migration costs, number of interdependent teams	
A case study describing procedural extraction of microservices from a monolithic architecture, where the challenges in each step of the procedure are listed[267]	✓	Discussion	Number of interdependent database tables,number of RESTful APIs	
Describing a pattern for extracting microservices from monoliths based on incrementally building new functionalities surrounding already existing ones[126]				
Describing a pattern for extracting microservices driven by the event flow throughout the architecture[125]				

Decision Support System	Scalability Considered	Scalability Assessment Approach	Scalability Dimensions Considered	Scalability Metrics Considered
Proposal of a technique which identifies candidates for microservice decomposition depending on whether they are client-related, server-related or data-related[197]	✓	Case study application	Code base size (measured in lines of code), number of shared database tables, number of cross-cutting business functionalities, number of microservices per cross-cutting business functionality	
Issues related to componentisation, organisation, endpoints and messaging mechanisms in the microservice architecture are discussed; these issues affect granularity adaptation hence this discussion can support the decision-making process[34]				
An experience reporting on a migration process to decompose an existing application into microservices and experiences from applying this process in an ongoing legacy modernization project[186]	✓	Case study application	Technology migration costs, number of bounded contexts, number of access points to back-end microservices, number of non-user related business functionalities, volume of read/write database operations, infrastructure platform migration costs	transactional consistency after pursuing adaptation
A microservice decomposition approach based on extending the usage of web usage mining techniques and clustering algorithms to characterise the workloads received by a microservice application[225]	✓	Case study application	Volume of requests received by the application	
An experience reporting on a migration process to decompose an existing application into microservices and hte lessons learnt from this transition[62]	✓	Discussion	end user base size, complexity of end user requirements	

Table 6.2 Summarising whether and how scalability is considered by the "processes" (i.e. decision support systems) in Figure 2.6; a check mark in the second column means scalability is considered when the respective system suggests granularity adaptation decisions; the third column shows dimensions which the respective system deems important to consider for rendering scalability-aware decisions

Looking at Table 6.2, we can make two observations: 1) scalability has not been considered for several examined publications and, 2) where scalability has been considered, there is little consensus regarding the dimensions and/or metrics across the examined publications. These observations leave room for our first contribution — the working catalogue of microservice-specific scalability dimensions and metrics aiming to provide specialised guidance to identify the dimensions and metrics which are important for the scalability of a microservice architecture. Consequently, these dimensions are essential to consider in order to render a scalability-aware granularity adaptation decision regarding a specific microservice architecture.

6.3.2 Scalability Dimensions derived from Granularity Adaptation Strategies

Appendix B shows example granularity adaptation strategies that can be formulated in the *GranAdapt* aspect of a microservice ambient. The driving forces, rationale and/or residual issues of each described strategy can inspire scalability dimensions and/or metrics to be considered in order to render a scalability-aware granularity adaptation decision. Table 6.3 summarises the dimensions and/or metrics derived from the strategies outlined in Appendix B.

Granularity Adaptation Strategy	Derived Scalability Dimensions	Derived Scalability Metrics
Cross-functional teams	number of teams responsible for each microservice, volume of shared knowledge across teams	
Polyglot persistence	Data translation costs, computation resource costs, database maintenance costs, number of interdependent database tables	
Extensibility point	Number of consumer-driven contracts	ease of feature introduction after adaptation, standardisation across interfaces after adaptation, stability of the architecture after adaptation

Granularity Adaptation Strategy	Derived Scalability Dimensions	Derived Scalability Metrics
Bulkhead		failure rate of adapted architecture

Table 6.3 Summarising potentially important scalability dimensions and metrics reflecting on granularity adaptation strategies described in Appendix B

Comparing Tables 6.2 and 6.3, we observe additional scalability dimensions and metrics in Table 6.3. Therefore, our reflection on strategies in Appendix B further serves the comprehensiveness of our catalogue. Nevertheless, there is an overlap between the dimensions in both tables (e.g., number of interdependent database tables, number of consumer-driven contracts, and number of teams responsible for each microservice). For this reason, we are inclined to think that our catalogue of scalability dimensions and metrics can be transferable across multiple application domains. However, this is a hypothesis that requires further research to be validated.

6.3.3 Catalogue of Microservice-Specific Scalability Dimensions and Metrics

Compiling Tables 6.2 and 6.3, we present our catalogue of scalability dimensions and metrics in Tables 6.4 and 6.5 respectively; dimensions and metrics are categorised according to their nature (e.g., organisational, data-related and developmental).

Category	Scalability Dimensions
Architectural	Number of interdependent microservices, number of user-facing interfaces, number of microservice registries, number of load balancers per microservice, number of event buses, number of interdependent middleware, number of bounded contexts
Deployment	containerisation/virtualisation costs, deployment pipeline costs, number of front-end servers, number of interdependent deployment configuration settings, number of RESTful API gateways, number of configuration files, computation resource costs
Security	security costs, number of access points to back-end microservices
Data	volume of shared data across microservices, number of independent databases, volume of queries received by the data store, data transport and synchronisation costs, volume of data transactions, volume of read/write database operations, data translation costs, database maintenance costs

Category	Scalability Dimensions
Testing	Number of integration test cases spanning multiple microservices
Logging	logging costs, caching costs
Communication	Network latency across microservices, number of light-weight communication mechanisms used, number of available virtual IP addresses, volume of published events in publish-subscribe communication mechanisms, volume of requests in synchronous communication mechanisms
Operational	Operational and infrastructure costs
Developmental	Number of development technologies used, number of codebase versions, number of shared libraries across microservices, volume of documentation, volume of multi-threading, technology migration costs
Organisational	Number of interdependent teams working on the architecture, volume of shared knowledge across teams
Monitoring	monitoring costs
QoS provision	number of consumer-driven contracts to be met, end user base size, complexity of end user requirements
Geographical	number of countries the application is serving
Legal	audit compliance considerations

Table 6.4 Working catalogue of microservice-specific scalability dimensions compiled from Tables 6.2, 6.1 and 6.3; it is potentially essential to consider these dimensions to render scalability-aware granularity adaptation decisions

Category	Scalability Metrics
Developmental	ease of feature introduction after adaptation, ease of independent development after pursuing adaptation
Architectural	Standardisation across interfaces after adaptation, stability of the architecture after adaptation
Data	Data consistency after adaptation, transactional consistency after pursuing adaptation
QoS provision	system performance (in throughput/response time), failure rate of adapted architecture

Table 6.5 Working catalogue of microservice-specific scalability metrics compiled from Tables 6.2 and 6.3; relevant scalability metrics can measure the ability of a microservice architecture to scale along the relevant dimensions from Table 6.4

It is worth noting that not all scalability dimensions will lead to scalability problems. In practice, some of them will vary within small ranges, imposing very little

impact on scalability. The same can be said about scalability metrics; the relevance of a scalability metric highly depends on its criticality to the microservice architecture. Nevertheless, this catalogue can be useful when the relevant dimensions and metrics from the catalogue are systematically linked (through our second contribution — scalability goal-obstacle analysis) to the system's goals, and consequently to the likelihood and criticality of scalability obstacles. In other words, the dimensions from our catalogue can scope the scalability-goal obstacle analysis. On the other hand, the analysis justifies the criticality of dimensions picked from the catalogue given a microservice architecture.

6.4 SYSTEMATIC SCALABILITY ANALYSIS FOR MICROSERVICE GRANULARITY DECISION SUPPORT

6.4.1 KAOS Goal-Oriented Modelling

The Keep All Objectives Satisfied (KAOS) framework allows modelling a software system as a collection of top-level goals operationalised through a hierarchy of AND/OR refinements to relate top-level goals to lower level sub-goals which ensure them [344]. An AND-refinement relates a goal to a set of sub-goals; this means that satisfying all sthe ub-goals in is necessary for achieving the parent goal. OR-refinement links relate a goal to a set of alternative sub-goals (which may include further refinements); achieving one of the alternative sub-goals is sufficient for achieving the parent goal. Each goal has is a prescriptive statement including its pattern (e.g.,Achieve, Maintain, Avoid), name and natural language definition [104]. Each goal is assessable by satisfaction criteria and/or metrics.

Each goal is connected to an agent(s) through a responsibility link. Agents are active system components, such as humans, hardware devices and software components, that are capable of goals they are responsible for. Goals range from high-level business objectives whose satisfaction involves multiple agents (e.g., providing efficient decision-making support), to fine-grained technical properties involving fewer agents (e.g., monitoring runtime evidence variables related QoSs of concern) [104].

Unlike goals that are prescriptive, domain hypotheses and assumptions are descriptive statements about the system or its usage context which are subject to change but their validity is necessary for goal achievement [104].

6.4.2 Scalability Goal-Obstacle Analysis

Given a refined KAOS goal-oriented model of a system, scalability goal-obstacle analysis aims "to take a pessimistic view of the model elaborated so far by systematically considering how the actual system might deviate from the model [104, p.4]". This entails the following steps [104]:

1. Identifying as many scalability obstacles as possible by systematically considering all leaf goals and assumptions in the goal graph;

2. Assessing the relative importance of the identified obstacles in terms of their likelihood and criticality to top-level goals;

3. Resolving the highly risky obstacles (which are both highly critical and highly likely) using obstacle resolving tactics. These include modifying existing goals, requirements and assumptions, or by introducing new ones so as to prevent, reduce or tolerate the obstacles.

A scalability obstacle is a condition that obstructs the goal from being satisfied when the load imposed by the goal on agents involved in its satisfaction exceeds the capacity of the agents. Each goal is connected to the obstacles obstructing it using an obstruction link. A scalability obstacle uses the concept of goal load and agent capacity to denote measures that characterize the amount of work needed and the amount of resources available to the agent to satisfy the goal, respectively [104]. Therefore, a scalability obstacle takes the form *Goal Load Exceeds Agent Capacity*. Goal loads are also referred to as scaling dimensions — application domain and system design properties that may exhibit a wide variation in values during the lifetime of the system.

Assessing the relative importance of a scalability obstacle is a product of its likelihood and criticality inspired by the risk analysis matrix technique [241]. In this

matrix, the likelihood an obstacle is estimated qualitatively on a scale from low to high and a similar scale is used to estimate criticality. This technique has been used in the context of scalability goal-obstacle analysis in [9, 104] so we utilise the same technique for consistency. We envision however that objective techniques such as those used in [175, 294] can also be applied to assess scalability obstacles.

Resolving scalability obstacles can be done using a range of tactics that satisfy the following strategies [345, 91]: goal substitution, agent substitution, obstacle prevention, goal weakening, obstacle reduction, goal restoration, obstacle mitigation, and do-nothing. The obstacle prevention strategy for example can be satisfied using tactics such as introducing either a domain assumption to be satisfied by some agent or a scalability goal. A scalability goal is a quality goal constrained by expected variations on the scaling dimensions [104].

6.4.3 Applicability to Microservice Granularity Decision Support

The results of scalability goal-obstacle analysis can inform reasoning about microservice granularity decision support along a number of dimensions:

- The scalability obstacle resolution techniques can suggest assessing multiple claims simultaneously using our evaluation process from Chapter 4.
- Identifying the critical scalability dimensions and metrics among ones in the catalogue related to a microservice architecture can inform compiling the *C and G banks* to provide input to our planning engine from Chapter 5.
- Identifying the critical scalability dimensions and metrics among ones in the catalogue related to our planning engine from Chapter 5 can help enhance its design to accommodate larger-scale architectures.

6.5 CASE STUDY APPLICATION

In this section we use our initial Filmflix architecture from Section 1.1 and our planning engine as case studies for showing how our contributions address the problems of concern in Section 6.1. For Filmflix, we first ad-hocly discuss the dimensions

and metrics which can affect its scalability. Then we apply scalability goal-obstacle analysis to Filmflix; the analysis is scoped to dimensions from our catalogue which we deemed relevant to Filmflix's scalability. For the planning engine, we first ad-hocly discuss the dimensions and metrics which can affect its scalability with respect to input parameters fed to the engine. Then we apply scalability goal-obstacle analysis to the planning engine to systematically identify and analyse its potential scalability obstacles.

Reflecting on utilising our catalogue for Filmflix and the planning engine, we discuss the potential comprehensiveness of Tables 6.4 and 6.5. Nevertheless, we acknowledge that further investigation is needed to extend and/or refine our catalogue. Comparing both scalability assessment approaches of Filmflix, we show how scalability goal-obstacle analysis leads to much more informed results than ad-hoc scalability assessment (Section 6.5.5).

6.5.1 Filmflix Ad-hoc Scalability Assessment

Our scalability assessment in this section is inspired by our observations of FilmFlix's initial architecture from Section 1.1. It contains three microservices: ReviewRegulation — implementing the regulation of movie review, ReviewUpload — managing the user input requirements when uploading a review, and MovieReview — capturing input from the user through an interface.

We regard the size of "foul" terms blacklist, the number of user input fields when uploading a review, and the number of reviews submitted simultaneously to Filmflix as the dimensions that can potentially affect Filmflix's scalability. This is grounded on the intuition that only the inputs are directly relevant to the functionality of microservice in Filmflix can affect its scalability. ReviewRegulation compares each submitted review against each term in the blacklist, so its size affects the ability of Filmflix to perform acceptably. Depending on the number of input fields required by Filmflix architects, the performance of ReviewUpload can be affected. MovieReview is the user-facing interface through which movie reviws are submitted; the number of submitted reviews impacts Filmflix's ability to perform acceptably. Furthermore, we

regard the response time of Filmflix as the only critical scalability metric. Thereafter, our ad-hoc assessment identifies three potential scalability obstacles:

- As the number of foul terms in the blacklist increases, the response time of the Filmflix initial architecture can deteriorate to an unacceptable extent.
- As the number of input fields required by Filmflix architects increases, the response time of the Filmflix initial architecture can deteriorate to an unacceptable extent.
- As the number of simultaneous reviews submitted to Filmflix increases, the response time of the Filmflix initial architecture can deteriorate to an unacceptable extent.

6.5.2 Filmflix Systematic Scalability Analysis

In this section, we apply scalability goal-obstacle analysis to the Filmflix initial architecture. We then discuss how the analysis results can inform inputs to the planning engine if it were to suggest granularity adaptation decisions in Filmflix. This section presents a usage scenario of our contributions and serves the following purposes:

- Illustrating how goal-obstacle analysis can aid in systematically highlighting and justifying dimensions which need to be considered when suggesting scalability-aware granularity adaptation decisions (one dimension of our contributions' benefits presented in Section 6.1).
- Serving as an example of how microservice architects can get more informed granularity adaptation decisions if they provide more well-rounded input to the system providing decision support to reason about before suggesting those decisions.
- Serving as an example for microservice architects to replicate our guidance on other microservice architectures.

6.5.2.1 KOAS Goal-Oriented Modelling of Filmflix

Filmflix has seven goals assigned to five agents; the goal model is presented and refined in Figure 6.1 using the notation explained in Figure 6.10. In this subsection, we describe the role of each agent to justify their responsibilities for different goals.

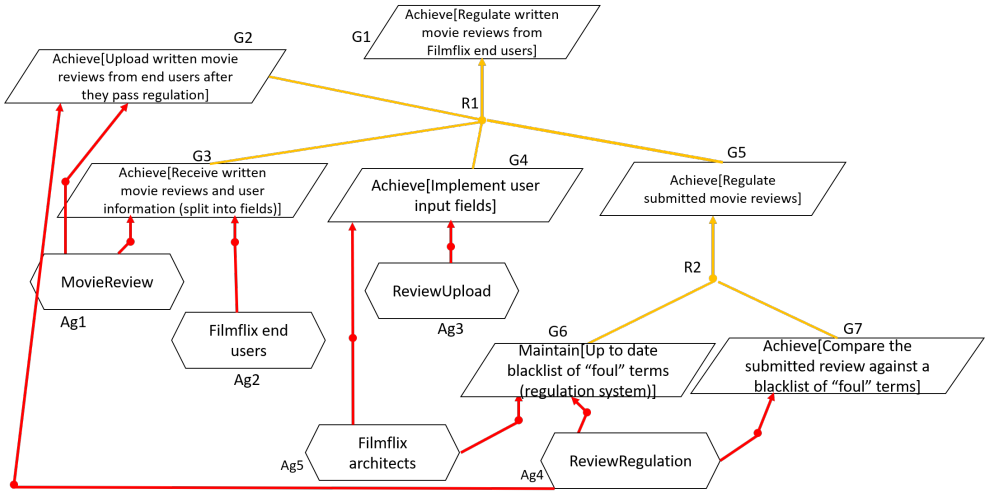


Fig. 6.1 Refinement of the *Achieve[Regulate written movie reviews from Filmflix end users]* goal in Filmflix's initial architecture

MovieReview: This agent is user-facing interface of Filmflix; it is responsible for receiving input and displaying output related to the high-level goal (G1 in Figure 6.1). Receiving input is represented by G3 — *Achieve[Receive written movie reviews and user information (split into fields)]*; displaying output is represented by G2 — *Achieve[Upload written movie reviews from end users after they pass regulation]*.

Filmflix end users: This agent refers to any active Filmflix end user which submits input to MovieReview. Therefore, active Filmflix end users share with the responsibility of achieving G3 in Figure 6.1 with MovieReview.

ReviewUpload: This agent is a Filmflix microservice which is not facing end users but implements Filmflix architects' requirements regarding the required end user input fields (e.g., name, age, ethnicity, email etc.). Therefore, ReviewUpload contributes to the design of MovieReview by achieving G4 — *Achieve[Implement user input fields]*.

ReviewRegulation: this agent is a Filmflix microservice which is core to regulating the output displayed by MovieReview. Given each review submitted by a Filmflix

end user, this microservice: 1) compares it to a pre-defined blacklist (G7 in Figure 6.1) and, 2) allows MovieReview to upload reviews which do not contain any term on the blacklist (contributing to G2). In parallel, ReviewRegulation has to maintain this blacklist (G6 in Figure 6.1) given any compliance requirements dictated by Filmflix architects.

Filmflix architects: This agent refers generically to any source which the architects utilise to determine input information required by Filmflix end users and foul terms that need to be included in the blacklist used by ReviewRegulation. These sources include but are not limited to data privacy laws, historical data, and compliance regulations. Therefore, Filmflix architects provide the input to achieve G4 and G6 in Figure 6.1.

6.5.2.2 Scalability Goal-Obstacle Analysis of Filmflix

Conducting scalability obstacle analysis on the goal model revealed five potential scalability obstacles related to four scaling dimensions. In this subsection we present the process of identifying, assessing and resolving these obstacles. Based on this systematic analysis, we discuss how goal obstacle resolution tactics can inform input to the planning engine if it were to be used for the initial Filmflix architecture. We compare this informed input to that used by the planning engine in Section 4.5.1. Tables 6.6 and 6.7 summarise the obstacle identification and assessment results.

Goal	Scalability metric	Scalability dimension	Influenced by
Achieve[Regulate written movie reviews from Filmflix end users]	MovieReview performance	volume of received reviews,number of "foul" terms in blacklist, number of Filmflix end users, number of Filmflix architects, number of user input fields	volume of received reviews <depends on> [number of Filmflix end users], number of "foul" terms in blacklist <depends on> [number of Filmflix architects], number of user input fields <depends on> [number of Filmflix architects]

Goal	Scalability metric	Scalability dimension	Influenced by
Achieve[Upload written movie reviews from end users after they pass regulation]	MovieReview performance	volume of reviews that passed the regulation	
Achieve[Receive written movie reviews and user information (split into fields)]	MovieReview performance	volume of received reviews, number of user input fields	
Achieve[Implement user input fields]	ReviewUpload performance	number of Filmflix architects, number of user input fields	number of user input fields <depends on> [number of Filmflix architects]
Achieve[Regulate submitted movie reviews]	ReviewRegulation performance	number of "foul" terms in blacklist, number of Filmflix architects, volume of received reviews, number of Filmflix end users	number of "foul" terms in blacklist <depends on> [number of Filmflix architects], volume of received reviews <depends on> [number of Filmflix end users]
Maintain[Up to date blacklist of "foul" terms (regulation system)]	ReviewUpload performance	number of "foul" terms in blacklist, number of Filmflix architects	number of "foul" terms in blacklist <depends on> [number of Filmflix architects]
Achieve[Compare the submitted review against a blacklist of "foul" terms]	ReviewRegulation performance	number of "foul" terms in blacklist	

Table 6.6 Identifying relevant scalability dimensions and metrics (guided by Tables 6.4 and 6.5) for the modelled goals of the initial Filmflix architecture; this table is used to identify the scalability obstacles in Table 6.7

Scalability Obstacle	Criticality	Likelihood	Rationale
Number of Filmflix end users exceeds MovieReview's ability to achieve acceptable performance	High	High	Based on the assumption from Section 1.1 that Filmflix on a scale similar to Netflix, there is a high likelihood of having a large number of active end users submitting reviews. This will affect MovieReviews ability to achieve G3 in Figure 6.1. Since MovieReview is the user-facing interface, it is critical for its performance to remain acceptable to avoid losing the interest of a large number of end users.
Volume of reviews which passed the regulation exceed MovieReview's ability to achieve acceptable performance	High	Low	Even for the scale at which Filmflix operates, only a fraction of the received reviews will be uploaded by MovieReview, so they likelihood of this obstacle is low. Nevertheless, if this obstacle where to occur it would be critical since it affects achieving a user-facing goal (G2 in Figure 6.1).
Number of user input fields exceeds MovieReview's ability to achieve acceptable performance	Low	Low	Since received user information is not subject to regulation and it is not uploaded to MovieReview along with an accepted review, the number of fields has little impact on MovieReview's performance.
Number of Filmflix architects exceeds ReviewUpload's capacity to achieve acceptable performance	Low	High	The likelihood of this obstacle depends on possibility of conflicts across international data privacy and compliance rules (both of which are sources for Filmflix architects to determine the input to ReviewUpload). Possibility of such conflicts is high given the potential geographical distribution of Filmflix end users. Even if this obstacle were to occur, its criticality to the overall goal G1 in Figure 6.1 is low since ReviewUpload does not utilise or regulate the submitted reviews.
Number of Filmflix end users exceeds ReviewRegulation's ability to achieve acceptable performance	High	High	The number of active end users determines the volume of reviews which have to regulated. For the scale of Filmflix, it is highly likely to have a large number of active end users leading to a large volume of reviews to be regulated and high likelihood of this obstacle. If ReviewRegulation does not achieve acceptable performance, then the user-facing G2 is potentially obstructed leading to the risk of losing end users' interest.

Scalability Obstacle	Criticality	Likelihood	Rationale
Number of Filmflix architects exceed ReviewRegulation's ability to achieve acceptable performance	High	High	Given the potential geographical spread of Filmflix end users, it is highly likely a large number of sources is used by Filmflix architects to determine the "foul" terms which ReviewRegulation has to maintain. There are two highly likely implications of this: 1) a large number of terms against which each review needs to be compared and 2) frequent updates to the blacklist during which no regulation can be done. Both implications are critical since they are obstruct achieving G2, G6 and G7 with acceptable performance.

Table 6.7 Assessing scalability obstacles of the initial Filmflix architecture

Reflecting on Table 6.7, there are three high risk obstacles that need to be resolved in order to ensure Filmflix's initial architecture achieves its goals with acceptable performance. In other words, these obstacles need to be considered when reasoning about adapting the granularity of Filmflix's initial architecture.

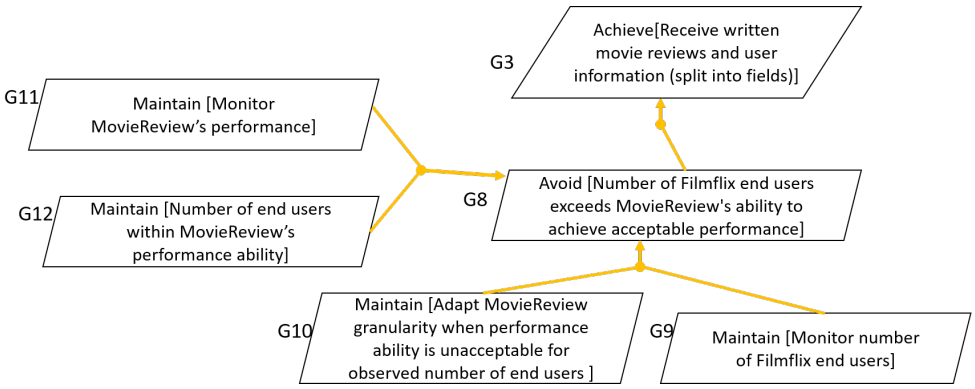


Fig. 6.2 Resolving the *Number of Filmflix end users exceeds MovieReview's ability to achieve acceptable performance* obstacle by introducing and refining the *Avoid [Number of Filmflix end users exceeds MovieReview's ability to achieve acceptable performance]* scalability obstacle prevention goal

Using resolution tactics from [104, 91], we propose preventing the *Number of Filmflix end users exceeds MovieReview's ability to achieve acceptable performance* obstacle from occurring by introducing and refining a scalability obstacle prevention goal:*Avoid [Number of Filmflix end users exceeds MovieReview's ability to achieve acceptable performance]*. This goal can be achieved via two routes, illustrated as

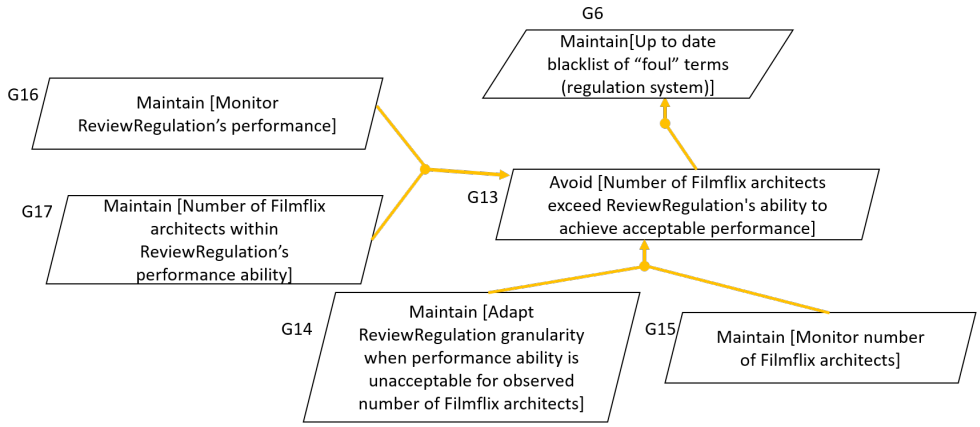


Fig. 6.3 Resolving the *Number of Filmflix architects exceed ReviewRegulation's ability to achieve acceptable performance* obstacle by introducing and refining the *Avoid [Number of Filmflix architects exceed ReviewRegulation's ability to achieve acceptable performance]* scalability obstacle prevention goal

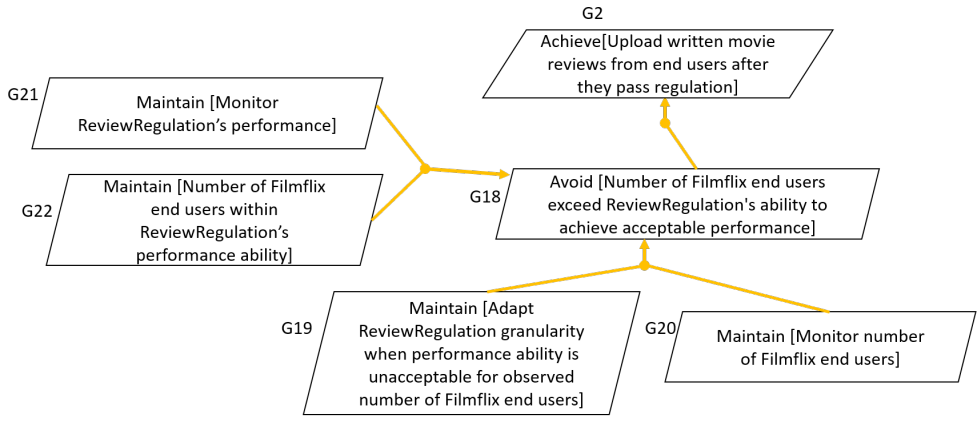


Fig. 6.4 Resolving the *Number of Filmflix end users exceed ReviewRegulation's ability to achieve acceptable performance* obstacle by introducing and refining the *Avoid [Number of Filmflix end users exceed ReviewRegulation's ability to achieve acceptable performance]* scalability obstacle prevention goal

an OR-refinement of G8 in Figure 6.2. On one hand, it can be achieved by ensuring the MovieReview's granularity level enables it to perform acceptably given observed numbers of Filmflix end users (G9 and G10 in Figure 6.2). If Filmflix architects were to use support of our planning engine to achieve G9 and G10, architects can use them to justify articulating a claim that captures the relationship between number of Filmflix end users and MovieReview performance. In Section 4.5.1, we appreciate the need for a claim that monitors workload (i.e. number of Filmflix end users) and MovieReview performance. However, we replaced the number of Filmflix end users with review string length due to the controlled experiment setting of Section 4.5.1. It is

worth noting that by conducting what-if analysis on simulations of the initial Filmflix architecture a quantifiable threshold can be derived for "acceptable performance"; it can henceforth be used as the QoS threshold of the claim input to the planning engine.

Alternatively, G8 in Figure 6.2 can be achieved if Filmflix architects can ensure that the number of Filmflix end users never stresses MovieReview beyond its performance ability (G12 in Figure 6.2). This ability is derived from monitoring MovieReview's performance (G11 in Figure 6.2). This route of achieving G8 does not involve adapting MovieReview's granularity and therefore it is does not require using microservice granularity adaptation decision support.

To resolve the *Number of Filmflix architects exceed ReviewRegulation's ability to achieve acceptable performance* obstacle, we propose introducing and refining a scalability obstacle prevention goal:*Avoid [Number of Filmflix architects exceed ReviewRegulation's ability to achieve acceptable performance]*. This goal can be achieved via two routes, illustrated as an OR-refinement of G13 in Figure 6.3. Although G13 is presented as a refinement of G6 in Figure 6.3, we appreciate that G13 is also relevant to achieving G2 and G7 from Figure 6.1. G13 can be achieved by ensuring the ReviewRegulation's granularity level enables it to perform acceptably given observed numbers of Filmflix architects (G14 and G15 in Figure 6.3). The number of Filmflix architects is a generic term referring to the number of sources which are consulted to build the blacklist used by ReviewRegulation.

If Filmflix architects were to use support of our planning engine to achieve G14 and G15, architects can articulate a claim that captures the relationship between number of Filmflix architects and ReviewRegulation performance. Our controlled experiment setting in Section 4.5.1 however does not articulate and assess that claim. Therefore, goal-obstacle analysis allows Filmflix architects to uncover and justify the necessity to provide this claim (along with that related to G9 and G10) as input to the planning engine. Assessing both claims simultaneously can lead to suggesting granularity adaptation strategies which are different from those suggested if only one of the claims had been assessed.

Consider a case where only claim capturing the relationship between MovieReview and number of end users is assessed. Further consider that the planning engine

suggests merging MovieReview with ReviewRegulation in this case aiming to enhance MovieReview's high QoS provision utility. However, it is possible that adopting this strategy leads to worse ReviewRegulation performance. The intuition behind this possibility is that updating the blacklist in ReviewRegulation can be interrupted or delayed to prioritise completing user-facing functionality which is related to MovieReview. These interruptions can be become critical when there is large number of Filmflix architects demanding many updates to the blacklist. It is possible to assess the accuracy of this possibility by simultaneously assessing both: 1) the claim relating ReviewRegulation's performance to the number of Filmflix architects and 2) the claim relating MovieReview's performance to the number of Filmflix end users. Thereafter, the planning engine suggests a granularity adaptation strategy that manages the trade-off between MovieReview's and ReviewRegulation's performance.

G13 in Figure 6.3 can also be achieved if Filmflix architects can ensure that the number of sources used to compile the blacklist in ReviewRegulation never stresses that microservice beyond its performance ability (G17 in Figure 6.3). This ability is estimated from monitoring ReviewRegulation's performance (G16 in Figure 6.3). This route of achieving G13 does not involve adapting ReviewRegulation's granularity and therefore it is does not require using microservice granularity adaptation decision support.

To resolve the *Number of Filmflix end users exceeds ReviewRegulation's ability to achieve acceptable performance* obstacle, we propose introducing and refining a scalability obstacle prevention goal:*Avoid [Number of Filmflix end users exceed ReviewRegulation's ability to achieve acceptable performance]*. This goal can be achieved via two routes, illustrated as an OR-refinement of G18 in Figure 6.4.

G18 can be achieved by ensuring the ReviewRegulation's granularity level enables it to perform acceptably given observed numbers of Filmflix end users (G19 and G20 in Figure 6.4). If Filmflix architects were to use support of our planning engine to achieve G19 and G20, architects can use them to justify articulating a claim that captures the relationship between number of Filmflix end users and ReviewRegaulation performance. Our controlled experiment setting in Section 4.5.1 however does not articulate and assess that claim. Therefore, goal-obstacle analysis allows Filmflix

architects to uncover and justify the necessity to provide this claim (along with that used in Section 4.5.1) as input to the planning engine. Assessing all three claims simultaneously can lead to suggesting granularity adaptation strategies which are different from those suggested if only one or two of the claims had been assessed. In Chapters 4 and 5 we use a single claim to evaluate just as an illustration of using our dynamic evaluation process and planning engine.

G18 in Figure 6.4 can alternatively be achieved if Filmflix architects can ensure that the number of sources used to compile the blacklist in ReviewRegulation never stresses that microservice beyond its performance ability (G22 in Figure 6.4). This ability is derived from monitoring ReviewRegulation's performance (G21 in Figure 6.4). This route of achieving G18 does not involve adapting ReviewRegulation's granularity and therefore it is does not require using microservice granularity adaptation decision support.

6.5.3 Planning Engine Ad-hoc Scalability Assessment

Our scalability assessment in this section is inspired by observations of the planning engines input, processing and output from Chapter 5. In particular, we regard inputs to each planning engine iteration (i.e, economic analyser parameters, *C and G banks*) as the only dimensions that can potentially affect its scalability. This is grounded on an intuition that these are the only loads which directly impact the planning engine components. Thereafter, we regard the response time of the planning engine tool support (reported on briefly in Section 4.5.4) as the only scalability metric for the planning engine. This is grounded on an intuition that to microservice architects overall performance of the planning engine is the most critical in microservice environments which dynamically scale at runtime. Thereafter, this ad-hoc assessment identifies three potential scalability obstacles:

- As the number of economic analyser input parameters per runtime iteration increases, the response time of the planning engine for each iteration can deteriorate to an unacceptable extent.

- As the number of *C bank* members per runtime iteration increases, the response time of the planning engine for each iteration can deteriorate to an unacceptable extent.
- As the number of *G bank* members per runtime iteration increases, the response time of the planning engine for each iteration can deteriorate to an unacceptable extent.

6.5.4 Planning Engine Systematic Scalability Analysis

In this section, we apply to scalability goal-obstacle analysis to our planning engine. We argue that this application is gives stronger insight than separate applications to the economic analyser and microservice ambients since our planning engine orchestrates both our former contributions. Application to our planning engine shows another usage scenario of our contributions and serves the following purposes:

- Illustrating how goal-obstacle analysis can be applied to aid in systematically analysing the scalability of systems which support reasoning about microservice granularity adaptation.
- Serving as an example for microservice architects to replicate the analysis approach on other systems that support reasoning about microservice granularity adaptation.
- Evaluating the ability of each constituent component in our planning engine to withstand increase in its respective inputs (i.e. economic analyser input parameters, *C and G banks*) using the scalability goal-obstacle analysis.

6.5.4.1 KOAS Goal-Oriented Modelling of the Planning Engine

The planning engine has nine goals assigned to five agents; the goal model is presented and refined in Figures 6.5,6.6,6.7,6.8,6.9. The KAOS modelling notation is explained in Figure 6.10. In this subsection, we describe the role of each agent to justify their responsibilities for different goals.

Economic Analyser: This agent is the component of the planning engine responsible for determining whether it is worth pursuing granularity adaptation under

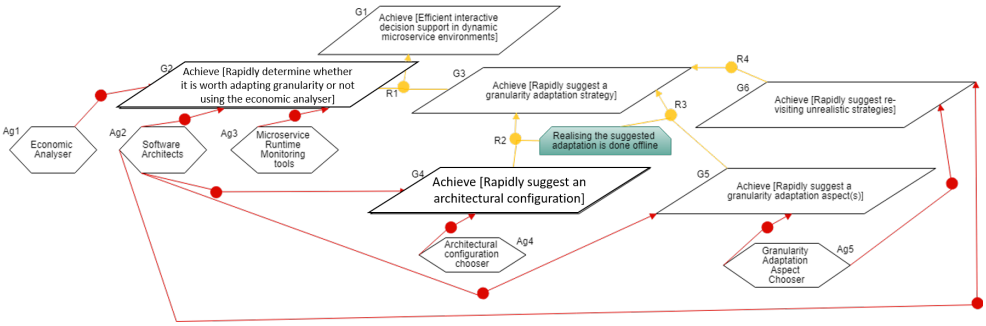


Fig. 6.5 Refinement of the *Achieve[Efficient interactive decision support in dynamic microservice environments]* goal in the planning engine

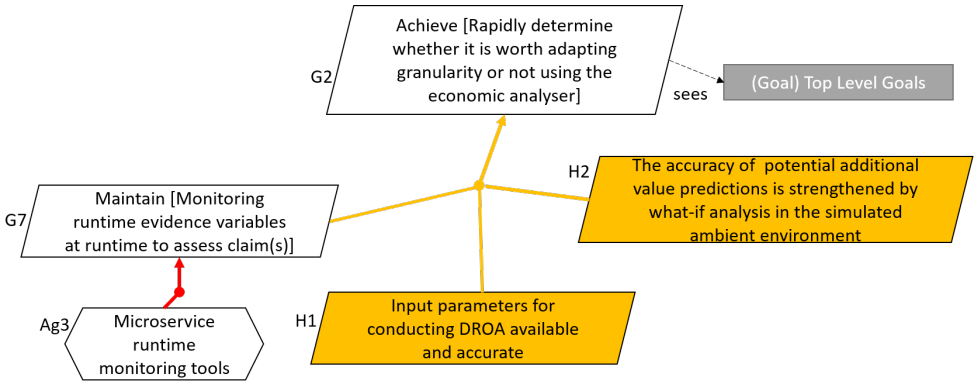


Fig. 6.6 Refinement of the *Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser]* goal from Figure 6.5

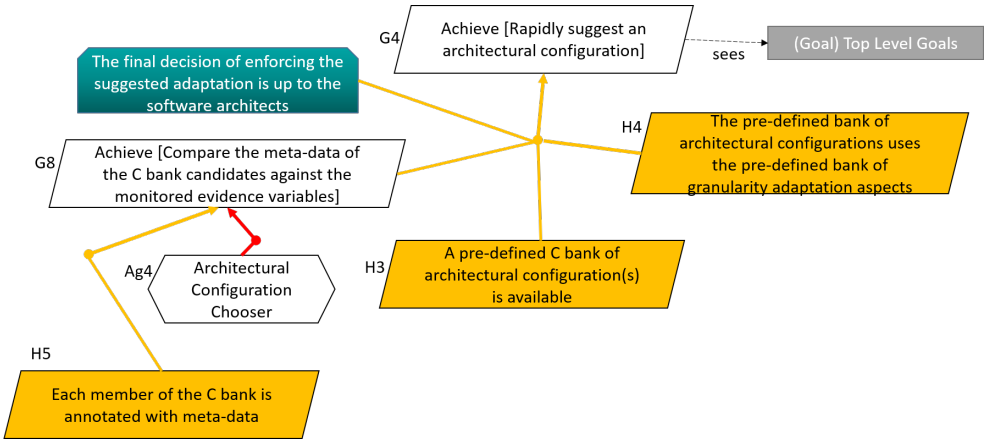


Fig. 6.7 Refinement of the *Achieve [Rapidly suggest an architectural configuration]* goal from Figure 6.5

uncertainty at runtime using our dynamic evaluation process explained in Section 4.4.3. Therefore, this agent is responsible for the *Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser]* goal (Figures 6.5,6.6).

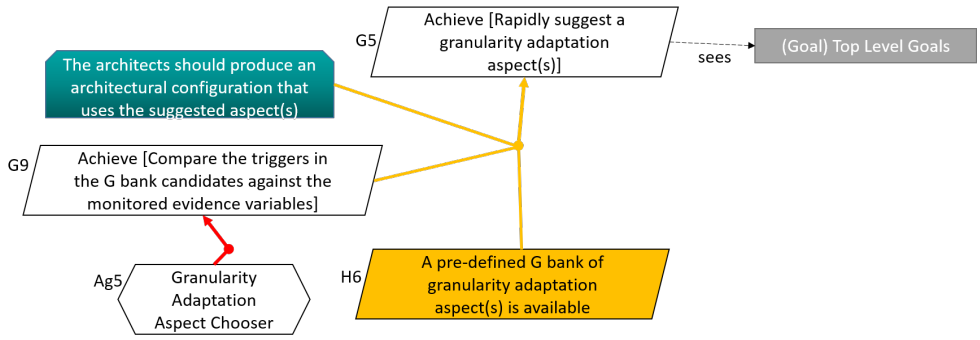


Fig. 6.8 Refinement of the *Achieve [Rapidly suggest a granularity adaptation aspect(s)]* goal from Figure 6.5

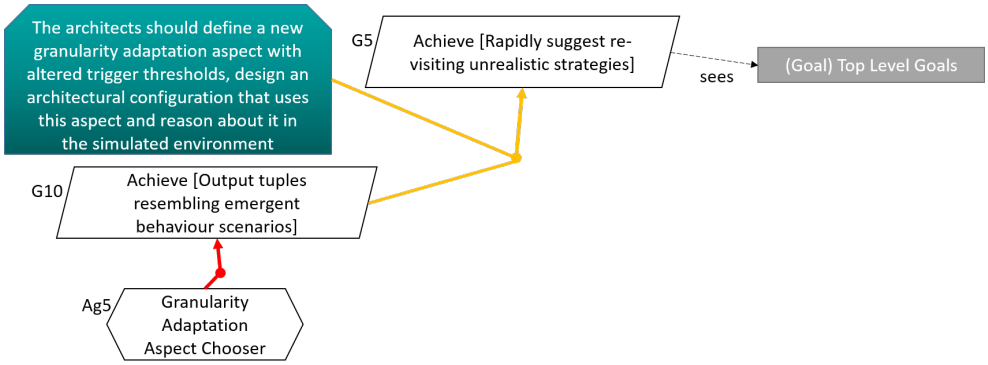


Fig. 6.9 Refinement of the *Achieve [Rapidly suggest re-visiting unrealistic strategies]* goal from Figure 6.5

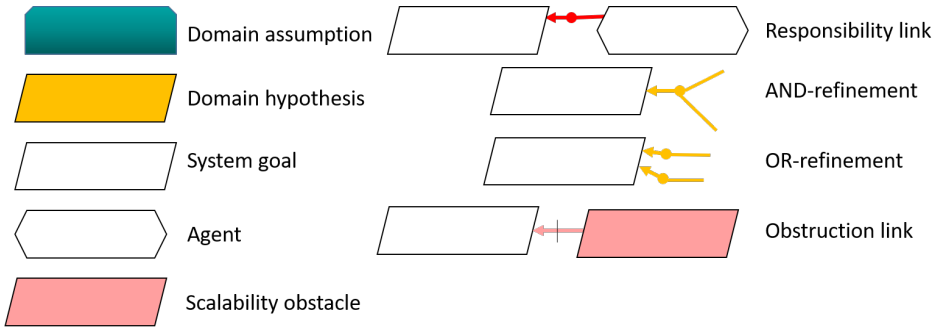


Fig. 6.10 Legend for the KAOS modelling notation

Software architects: This agent refers generically to any source which the architects utilise to provide the required input to constituent components of the planning engine. These sources include but are not limited to experts judgements, personal expertise, benchmarks, historical data, cost models, and/or simulated ambient environments. The domain hypotheses in Figures 6.6,6.7,6.8 elaborate on the inputs which need to be provided by the architects to each constituent component.

Due to the interactive nature of the planning engine, this agent has the final decision in enforcing the suggestions of the planning engine; this is shown as assumptions in Figures 6.7,6.8,6.9. Therefore, this agent has the dual responsibility of providing input and enforcing output to help achieve the following goals: *Achieve[Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser], Achieve [Rapidly suggest an architectural configuration], Achieve [Rapidly suggest a granularity adaptation aspect(s)], and Achieve [Rapidly suggest re-visiting unrealistic strategies]* (Figure 6.5). Nevertheless, other agents also have responsibilities towards achieving these goals.

Microservice runtime monitoring tools: This agent refers generically to any tool which the software architects utilise to monitor the runtime evidence variables that are solicited by the economic analyser. For the economic analyser to achieve the goal it is responsible for, a sub-goal needs to be maintained: *Maintain [Monitoring runtime evidence variables at runtime to assess claim(s)]* (Figure 6.6). The runtime monitoring tools are responsible for maintaining this sub-goal.

Architectural configuration chooser: This agent is the component of the planning engine responsible for suggesting a suitable architectural configuration after assessing the meta-data attached to each candidate in the *C bank* as explained in Section 5.4. Therefore, this agent is responsible for the *Achieve [Rapidly suggest an architectural configuration] and Achieve [Compare the meta-data of the C bank candidates against the monitored evidence variables]* goals (Figures 6.5,6.7).

Granularity Adaptation Aspect Chooser: This agent is the component of the planning engine responsible for suggesting suitable granularity adaptation aspect(s) from the *G bank* to the architect in case there are no suitable architectural configurations in the *C bank*. The suggestion is based on comparing the runtime evidence variables to each candidate in the *G bank* which sets a trigger on them. Therefore, this agent is responsible for achieving *Achieve [Rapidly suggest a granularity adaptation aspect(s)] and Achieve [Compare the triggers in the G bank candidates against the monitored evidence variables]*. If there are no suitable candidates in the *G bank* this agent is responsible for alerting the architects to emergent behaviour scenarios they didn't consider at design time. Therefore, this agent is also responsible for achieving

Achieve [Rapidly suggest re-visiting unrealistic strategies] and Achieve [Output tuples resembling emergent behaviour scenarios].

6.5.4.2 Planning Engine Scalability Goal-Obstacle Analysis

Conducting scalability obstacle analysis on the goal model revealed five potential scalability obstacles related to four scaling dimensions. In this subsection we present the process of identifying, assessing and resolving these obstacles for each constituent component. Based on this systematic analysis, we can at least partially guarantee the scalability of the planning engine if high risk scalability obstacles are resolved as suggested in this subsection. We explain the process in detail for the goal assigned to the economic analyser in Sections 6.5.4.2.1,6.5.4.2.2,6.5.4.2.3. The same rationale has been applied and summarised in Tables 6.8 and 6.9 for all the goals.

6.5.4.2.1 Identifying Scalability Obstacles The economic analyser is responsible for *Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser].* This goal is satisfied if all the claims for a runtime iteration can be assessed in a timely manner without exceeding the computation power capacity of the economic analyser. The load of this goal is determined by the number of claims the economic analyser has to assess (shown as the scalability obstacle *Claims exceed Economic Analyser Capacity* in Figure 6.11). The number of assessable claims is a product of the number of QoS attributes of concern, the number of runtime evidence variables related to each QoS attribute and the number of microservices for which a claim needs to be assessed. These factors are shown as sub-obstacles in Figure 6.11.

We regard performance (in throughput/response time) of the planning engine (reported on briefly in Section 4.5.4) as the only relevant scalability metric. This is grounded on an intuition that to microservice architects it is the overall performance of the planning engine that is most critical in microservice environments which dynamically scale at runtime.

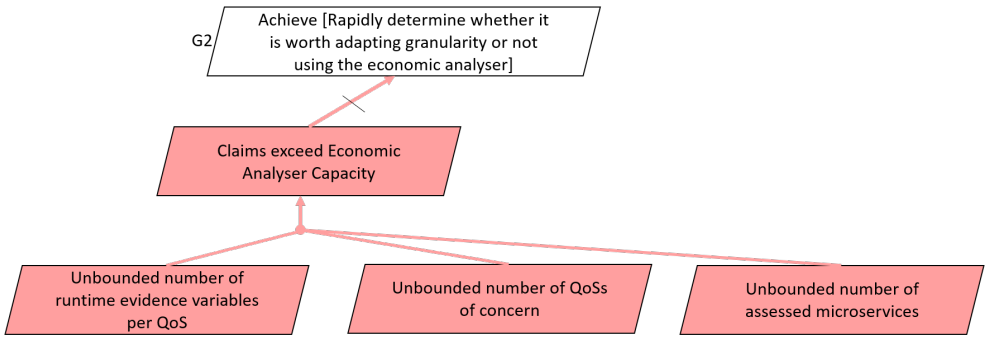


Fig. 6.11 Identifying and refining scalability obstacles for the *Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser]* goal using the notation in Figure 6.10

6.5.4.2.2 Assessing Scalability Obstacles Looking at Figure 6.11, there are three leaf scalability obstacles to be assessed: *unbounded number of QoS attributes of concern*, *unbounded number of runtime evidence variables per QoS attribute* and *unbounded number of assessed microservices*. Since all obstacles contribute to the number of claims that would need to be assessed at runtime, they all have a high criticality. However, the likelihood of each obstacle differs. Microservice adopters include large-scale industries (e.g, Amazon [353], BBC [145], Netflix [336], Uber [273], and SoundCloud [63]) with millions of end users and inevitably a large number of QoS attributes demanded by them. Therefore, the likelihood of *unbounded number of QoS attributes of concern* and *unbounded number of assessed microservices* is very high in the context of large-scale industries adopting microservices. On the other hand, it is seldom the case that there is an *unbounded number of runtime evidence variables per QoS*. In the architectural evaluation, goal-oriented approaches map high-level goals (i.e. quality goals) to their measurable metrics (i.e. runtime evidence variables) [343, 88, 22, 23, 175]. It is seldom the case that a runtime evidence variable maps to all the QoSs of concern. Therefore, it is seldom the case that q QoS attributes and r runtime evidence variables actually require q*r claims to reason about at runtime.

Therefore, it is only the scalability obstacles related to *unbounded number of QoS attributes of concern* and *unbounded number of assessed microservices* that are worth resolving since they are both highly likely and critical.

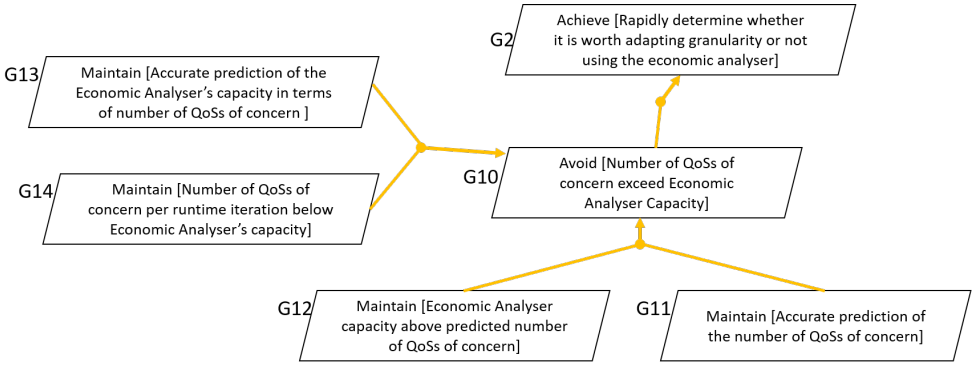


Fig. 6.12 Resolving the *number of QoS attributes of concern exceed economic analyser ability to achieve acceptable performance* obstacle by introducing and refining the *Avoid [number of QoS attributes of concern exceed Economic Analyser Capacity]* scalability goal

6.5.4.2.3 Resolving Scalability Obstacles We propose resolving the high risk scalability goal identified by preventing the *number of QoS attributes of concern exceed economic analyser ability to achieve acceptable performance* obstacle from occurring by introducing and refining a scalability obstacle prevention goal:*Avoid [number of QoS attributes of concern exceed economic analyser capacity]*.

This goal can be achieved via two routes, illustrated as an OR-refinement of G10 in Figure 6.12. On one hand, it can be achieved by ensuring the analyser's capacity tolerates a dynamically predicted number of QoSs (G11 and G12 in Figure 6.12). On the other hand, the architects can ensure that the number of QoS of concern are less than a dynamic prediction of the analyser's tolerance capacity (G13 and G14 in Figure 6.12). The ability of the planning engine to withstand an increase in the number of QoS attributes of concern can at least partially guarantee provided the scalability goal in Figure 6.12 is achieved.

Goal	Scalability metric	Scalability dimension	Influenced by
Achieve[Efficient interactive decision support in dynamic microservice environments]	constituent component performance	num. of QoS attributes of concern, num. of runtime evidence variables per QoS, *C bank* size, *G bank* size, num. of assessed microservices	

Goal	Scalability metric	Scalability dimension	Influenced by
Achieve [Rapidly determine whether it is worth adapting granularity or not using the economic analyser]	economic analyser performance	num. of QoS attributes of concern, num. of runtime evidence variables per QoS, num. of assessed microservices	
Achieve[Rapidly suggest a granularity adaptation strategy]	constituent component performance	*C bank* size, *G bank* size,num. of runtime evidence variables per QoS, num. of assessed microservices	*C bank* size <depends on> [num. of assessed microservices]
Achieve[Rapidly suggest an architectural configuration]	architectural configuration chooser performance	*C bank* size, num. of assessed microservices	*C bank* size <depends on> [num. of assessed microservices]
Achieve[Rapidly suggest a granularity adaptation aspect(s)]	granularity adaptation aspect chooser performance	*G bank* size,num. of runtime evidence variables per QoS, num. of QoS attributes of concern	*G bank* size <depends on> [num. of runtime evidence variables per QoS, num. of QoS attributes of concern]
Achieve[Suggest revisiting unrealistic strategies]	granularity adaptation aspect chooser performance	num. of QoS attributes of concern, num. of runtime evidence variables per QoS	
Maintain [Monitoring runtime evidence variables at runtime to assess claim(s)]		num. of QoS attributes of concern, num. of runtime evidence variables per QoS, num. of assessed microservices	
Achieve [Compare the meta-data of the C bank candidates against the monitored evidence variables]	architectural configuration chooser performance	*C bank* size, num. of assessed microservices	*C bank* size <depends on> [num. of assessed microservices]

Goal	Scalability metric	Scalability dimension	Influenced by
Achieve [Compare the triggers in the G bank candidates against the monitored evidence variables]	granularity adaptation aspect chooser performance	*G bank* size,num. of runtime evidence variables per QoS,num. of QoS attributes of concern	*G bank* size <depends on> [num. of runtime evidence variables per QoS,number of QoS attributes of concern]
Achieve [Output tuples resembling emergent behaviour scenarios]	granularity adaptation aspect chooser performance	num. of QoS attributes of concern, num. of runtime evidence variables per QoS	

Table 6.8 Identifying relevant scalability dimensions and metrics for the modelled goals of the planning engine; this table is used to identify the scalability obstacles in Table 6.9

Scalability Obstacle	Criticality	Likelihood	Rationale
Number of QoS attributes of concern exceed economic analyser ability to achieve acceptable performance	High	High	Each QoS needs to be tracked and assessed at runtime in the form of a claim using the economic analyser so the criticality of this obstacle is high. Moreover, microservice adopters include large-scale industries with millions of end users and inevitably a similar order of QoSs demanded by them; the likelihood of the obstacle is high in this context.
Number of runtime evidence variables per QoS exceed economic analyser ability to achieve acceptable performance	High	Low	The number of runtime evidence variables per QoS attribute determines the number of claims to be assessed related to that QoS, so this obstacle is highly critical to achieving acceptable performance of the economic analyser. However, assuming that goal-oriented approaches are used to map high-level goals (i.e. QoSs) to their measurable metrics (i.e. runtime evidence variables), this likelihood of this obstacle is low.

Scalability Obstacle	Criticality	Likelihood	Rationale
Number of assessed microservices exceeds economic analyser ability to achieve acceptable performance	High	High	The number of assessed microservices is one of the factros which determine the number of claims to be assessed by the economic analyser, so this obstacle is highly critical to achieving acceptable performance of the economic analyser. Given that some microservice adopters are large-scale industries, it is highly likely that there would be large number of microservices to be assessed for a given claim.
C bank size exceeds Architectural Configuration Chooser ability to achieve acceptable performance	High	Low	For c configurations input to the planning engine, the chooser always needs to examine c instances of *metadata* to produce the *C shortlist*. This obstacle is therefore highly critical. However, utilising the hierarchical property pf ambients reduces the likelihood of this obstacle. Abstracting the target scope of the planning engine using ambients allows reducing the possible variations on this scope. Each variation is a potential member of the *C bank*.

Scalability Obstacle	Criticality	Likelihood	Rationale
Number of runtime evidence variables per QoS exceeds Granularity Adaptation Aspect Chooser ability to achieve acceptable performance	High	Low	The number of granularity adaptation aspects in a *G bank* directly impacts the granularity adaptation aspect chooser's process. For g aspects input to the planning engine, the chooser must examine all of them in the first filtering stage. However, only a fraction of them needs to be examined in the second filtering stage. This fraction (which has g' aspects) is partially dependent on the number of runtime evidence variables per QoS. The triggers within g' are examined in the second stage. In the worst-case scenario, g=g' and therefore a single extra aspect input to the planning engine corresponds to 2 extra operations in this chooser (1 for each filtering stage), making this obstacle highly critical. However, the likelihood of this scenario (and thereby obstacle) is low. Using goal-oriented approaches, it is possible to elaborate on the runtime evidence variables considered in the triggers of each aspect in the *G bank*. Therefore, it is seldom the case that a runtime evidence variable is constrained in all g aspects within a *G bank*. In turn, it is often the case that g' is less than g.
Number of QoS attributes of concern exceed Granularity Adaptation Aspect Chooser ability to achieve acceptable performance	High	High	As explained above, the *G bank* size has a direct impact on the number of operations that need to be carried out in this chooser so this obstacle is highly critical. As explained in Table 6.8, the *G bank* size partially depends on the number of QoS attributes of concern. In the microservice industry, it is highly likely that there is a large number of QoSs to match the large variety of end users.

Table 6.9 Assessing scalability obstacles of the planning engine, which were identified through Table 6.8

Reflecting on Table 6.9, there are two high risk obstacles that need to be resolved in order to at least partially guarantee the scalability of the planning engine. Resolving these obstacles can be done by achieving the following scalability obstacle prevention goals:

- As the number of QoS attributes of concern increases, the economic analyser should have the capacity to achieve acceptable performance (refined in Section 6.5.4.2.3).
- As the number of microservices assessed for a given claim increases, the economic analyser should have the capacity to achieve acceptable performance. This goal can be refined using a similar rationale as Section 6.5.4.2.3.
- As the number of QoS attributes of concern increases, the granularity adaptation aspect chooser should have the capacity to achieve acceptable performance. This goal can be refined using a similar rationale as Section 6.5.4.2.3.

In Section 6.5.2.2, the results of Filmflix scalability goal-obstacle analysis showed that performance is the only scalability metric relevant to the critical scalability obstacles in Filmflix. In other words, even if all three claims suggested in Section 6.5.2.2 are assessed, they will all be related to a single QoS attribute of concern — performance. There are two runtime evidence variables related to that attribute (number of Filmflix end users and number of Filmflix architects) and one of the claims in assessed for two microservices (MovieReview and ReviewRegulation). Overall, there are three claims which the planning engine should assess to render a scalability-aware granularity adaptation strategy for Filmflix.

On the other hand, the critical scalability obstacles for the planning engine can be triggered if the number of the QoS attributes of concern or the number of assessed microservices increase beyond the respective constituent components' capacity. Unless this capacity is a single QoS attribute and two assessed microservices at time, we envision that the planning engine assessed in this section would be able to scale to assess all three claims related to Filmflix performance suggested in Section 6.5.2.2.

6.5.5 Reflection

6.5.5.1 Catalogue Comprehensiveness

Reflecting on the scalability dimensions in Table 6.7, we observe that the most critical scalability dimensions for Filmflix (i.e. number of Filmflix end users and number of

Filmflix architects) are present in Table 6.4. The number of end users is in essence the end user base size under the QoS provision category of Table 6.4. According to Section 6.5.2, Filmflix architects is a generic term referring sources which include but are not limited to data privacy laws, historical data, and compliance regulations. Therefore, the number of Filmflix architects in Table 6.7 potentially maps to the number of countries the application is serving and audit compliance considerations in Table 6.4. One of the scalability dimensions which have not been deemed critical — volume of received reviews — can be mapped to the end user base size under the architectural category in Table 6.4. The mapping is based on the relationship between volume of received reviews and number of Filmflix end users captured in Table 6.7. The same can be said about number of "foul" terms in blacklist and number of user input fields since they both depend on number of Filmflix architects which is a dimension implicitly present in Table 6.7. All the scalability metrics considered 6.6 are related to performance; it is present under the QoS provision category of Table 6.5.

It is worth noting however that while the application to Filmflix can act as evidence for the comprehensiveness of our catalogues, they can only be as good as the model of the analysed case study. Moreover, the comprehensiveness of our catalogue relies heavily on the completeness we strived for in Chapter 2. In particular, we used all the relevant publications from Chapter 2 to compile the catalogues due to our confidence in this chapter's coverage of the relevant literature.

It is also worth noting that although some of dimensions and metrics in our catalogue have not been deemed potential obstacles in Section 6.5, this does not mean those metrics/dimensions need not be considered for other microservice architectures. The aim of our catalogue is to provide comprehensive guidance for microservice architects about the possible scalability dimensions and metrics. Therefore, it is through systematic scalability goal-obstacle analysis of a particular microservice architecture that the dimension/metric significance to it can be justified.

6.5.5.2 Scalability Goal-Obstacle Analysis Significance

Comparing Filmflix scalability assessment results in Sections 6.5.1 and 6.5.2, we observe that goal-obstacle analysis delved to the actual dimensions which impact the ones identified by ad-hoc scalability assessment. Moreover, ad-hoc scalability assessment failed to systematically identify how obstacle resolution tactics can inform input to microservice granularity adaptation decision support; this is possible through goal-obstacle analysis.

Comparing planning engine scalability assessment results in Sections 6.5.3 and 6.5.4, we observe major differences. The main difference is in the scalability dimensions; ad-hoc analysis followed a myopic intuition thereby considering the inputs to the planning engine as the only dimensions that lead to scalability problems. On the other hand, systematic scalability goal-obstacle analysis analysed each input further to justify the critical scalability obstacles which are actually worth resolving to ensure it can scale to handle the necessary input parameters. Moreover, ad-hoc scalability assessment failed to systematically identify how to resolve the potential scalability obstacles. Goal-obstacle analysis on the other hand provides obstacle resolution tactics to aid in taking the appropriate action to for ensuring the planning engine can scale to with respect to its input parameters.

Although we apply scalability goal-obstacle analysis to one case for each usage scenario (a microservice architecture and a decision support system), the same experience can be copied to other systems that support reasoning about microservice granularity adaptation and other microservice architectures. We envision only one caveat regarding the decision support systems which can be targeted by our contributions: how systematic they are. In other words, they must have clear inputs, processing and outputs. Identifying scalability obstacles entails looking at the loads which each agent in the system receives; this is only possible if it does receive inputs. Assessing scalability goals entails identifying the criticality and likelihood of each obstacle; this is only possible if the analysed system has a process where an obstacle is critical and/or likely. Assessing scalability obstacles entails applying tactics that ensure the

overall goals of the system are satisfied; this is only possible if it has outputs which the goals aim to deliver.

It is worth noting that another possible usage of our contributions is applying them directly on members of the *C bank* and *G bank* to check their relative scalability. In essence, this usage would be an alternative to using our planning engine but with focus on choosing a scalable architectural configuration. In the planning engine however a more objective approach is taken to choose a granularity adaptation strategy; the added value which can be introduced by the granularity adaptation is quantified by our dynamic evaluation process. Moreover, the planning engine can choose granularity adaptation strategies that can introduce added value along any driver of interest (e.g., scalability, performance, availability). In this chapter, our usage of the catalogue and scalability goal-obstacle analysis complements rather than replaces the planning engine.

6.6 CONCLUSION

In this chapter, we contribute to a working catalogue of microservice-specific scalability dimensions and metrics. Our catalogue helps identify dimensions and metrics which are important for the scalability of a given microservice architecture; they need to be considered in order to render scalability-aware granularity adaptation decisions for it. We compile our catalogue by: 1) digesting the state-of-the-practice in decision support systems for microservice granularity adaptation and, 2) reflecting on the sample strategies formulated in Appendix B to infer additional potentially important scalability dimensions and/or metrics. Secondly, we apply scalability goal-obstacle analysis [104, 91] in the new context of reasoning about microservice granularity adaptation. Applying scalability goal-obstacle analysis to a given microservice architecture helps identify obstacles along each dimension of importance from our catalogue. Applying scalability goal-obstacle analysis to a given decision support system helps analyse the extent to which it can scale to consider all these dimensions and metrics.

We evaluate and discuss our contributions by comparing their usage to both initial Filmflix architecture and our planning engine in Section 1.1 against ad-hoc scalability assessment of both cases. Comparing both assessment approaches, we show how our contributions lead to more informed results than ad-hoc scalability assessment (Section 6.5.5). Finally, we discuss how scalability goal-obstacle analysis can be applied to other microservice architectures and other decision support systems for microservice granularity adaptation.

Chapter 7

Conclusions, Reflections and Future Work

In this chapter, we conclude the thesis by summarising how the research questions from Section 1.3 have been addressed by our contributions. We reflect on each contribution and its evaluation using different qualitative criteria (e.g., flexibility, computational overhead, and practical deployment). We consider flexibility as the ability of the contributions to cope with various microservice application domains. We consider practical deployment as the ability of the contributions to cope with industrial microservice settings.

We suggest research directions that can be addressed to enrich each contribution in the future.

7.1 RESEARCH QUESTION 1

Providing a better understanding of the transition to microservices — how can the state-of-the-art and practice in microservices be probed to advance the understanding of microservices and whether the transition to them has an impact on development tools, deployment management and/or personnel organisation etc.?

7.1.1 Contribution

In Chapter 2, we contribute a systematic mapping study that consolidates various views, approaches and activities that commonly assist in the transition to microservices. The study aims to provide a better understanding of the transition; we use the study to introduce a working definition of the transition and technical activities underlying it. We term the transition and technical activities leading to microservice architectures as microservitization:

Microservitization it is a form of servitization where services/components are transformed into microservices — a more fine-grained and autonomic form of services — to introduce added value to the architecture [148]. Microservitization is also an example of a paradigm shift since it involves a dramatic change to the following activities are done: architectural design, managing the organisational hierarchy, operation, deployment, development, monitoring and logging.

This systematic mapping study can advance the understanding of microservices and make the design challenges of the transition more explicit to microservice adopters and researchers in the software architecture community, thereby addressing Research Question 1.

Among the crucial design challenges introduced by microservitization is finalising the granularity level of a microservice (i.e. whether a microservice should be decomposed/merged further or not). We use the results of our systematic mapping study to identify the following gaps in the current state of the art and practice of defining the granularity of a microservice to motivate the need for our contributions in the rest of the thesis:

- Microservice-specific modelling support potentially using an architecture definition language (ADL) that "treats microservice boundaries as adaptable first-class entities [150, p.2]" thereby facilitating runtime analysis of microservice granularity in a systematic architecture-oriented manner [150]. We contribute this modelling approach in Chapter 3.

- A dynamic architectural evaluation process that captures two dimensions under uncertainty at runtime: added value to be introduced and cost to be incurred

if the granularity of microservice architectures is adapted. We contribute this evaluation process in Chapter 4.

- Effective decision support that should systematically guide the software architects towards suitable granularity adaptation strategies at runtime or suggest re-visiting their expectations of the architecture's runtime environment. We contribute this decision support in Chapter 5.

- A systematic approach for analysing the scalability obstacles of contributions that support reasoning about runtime microservice granularity adaptation. We apply such approach in the context of microservice granularity decision support in Chapter 6.

7.1.2 Reflection

We made our best effort to probe for a better understanding of the transition to microservices by using a systematic mapping study process [261, 116, 181, 262] and acknowledging the threats to validity in Section 2.5. We appreciate that a better understanding of the transition could have been arrived using other evidence-based approaches which extract evidence from multiple sources (e.g., interviewing microservice practitioners, industrial case studies). Evidence-based approaches aim to "to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding [182, p.2]" a contribution in the software engineering field. Such research approaches would ensure that the evaluated contribution is directed to the requirements of industry and other stakeholder groups. However, we argue that the systematic mapping study approach we followed is more suited to our thesis for the following reasons:

- "The human skill factor in evidence-based approaches means software engineering experiments are vulnerable to subject and experimenter bias [182, p.1]". We aim to reduce this bias by considering published blog articles, presentations and videos as arguably more trusted since they present a more responsible and objective view than interviews. Though they can enrich the study with diverse

opinions, interviews tend to suffer from bias, subjectivity and irresponsible answers [182].

- Software engineering contributions are hard to evaluate in isolation of other existing trends and the immediate outcomes of the contribution are not necessarily strongly related with the overall architecture's wellbeing [182, p.1]. Including microservice adoption case studies among the publications and adapting the definition from a microservice-specific systematic mapping study [17] aims to address this issue. In particular, reflecting on the scope of activities in the case studies strives for more comprehensive coverage in the systematic mapping study.

Directly consulting practitioners through interviews, industrial case studies, and/or surveys can nevertheless complement our systematic mapping study to evaluate it for acceptance among microservice practitioners. Moreover, we appreciate that such consultations can evolve our definition of microservitization. In particular, it could help determine the scope of "sub-activities" that comprise each microservitization activity listed in our definition. Consulting practitioners can help answer questions such as: Does microservitization entail changing the technology personnel allocations, re-structuring the organisational hierarchy throughout the firm or neither? Nevertheless, our definition Chapter 2 captures the essence of microservitization by looking at the fundamental activities underlying the transition.

7.1.3 Future Work

Our systematic mapping study can be further strengthened by investigating whether there are domain-specific rooms for improvement underlying the transition to microservices and reasoning about their granularity. For example, an interesting research dimension is to conduct domain-specific systematic mapping studies covering health or entertainment microservice applications. Conducting such studies can require several cases to understand the scope and variety of the domain-specific microservice challenges. The challenge we envision in this direction is gaining access to information regarding safety-critical microservice applications.

7.2 RESEARCH QUESTION 2

Formalising the microservice granularity adaptation problem — how can microservices be modelled using flexible, expressive support that aids defining different strategies for runtime microservice granularity adaptation?

7.2.1 Contribution

In Chapter 3, we call for a systematic, flexible and expressive architectural modelling and analysis support for representing microservices and managing their granularities to model this decision problem. A systematic architecture-centric approach for modelling microservice granularity provides the appropriate level of abstraction for managing this runtime decision problem. In particular, this approach has the promise to scale the analysis of this problem.

Thereafter, we contribute an architecture-centric modelling concept for microservices which extends ambients [66, 13] — microservice ambients. Microservice ambients provide the primitives for modelling microservices and and treat microservice boundaries as adaptable first-class entities. Microservice ambients use "aspects" to define the adaptation behaviour needed to support changes in granularity at runtime. Aspects flexibly separate cross-cutting concerns (and thereby scope) of each boundary. We introduce the *granularity adaptation aspect* to help define the runtime triggers for granularity adaptation to be used as the microservice(s) are monitored at runtime. When any of these triggers is invoked, the graphical notation of microservice ambients provides architectural modelling support to express the candidate solution for granularity adaptation. The transition (by merging or decomposing a microservice ambient(s)) from an initial architecture to a chosen candidate is captured in the granularity adaptation aspect of the microservice ambient.

7.2.2 Reflection

We make our best effort in Chapter 3 to qualitatively evaluate the expressiveness and effectiveness of microservice ambients and granularity adaptation aspects using Filmflix as an illustration example as well as criteria from Architecture Description

Language (ADL) classification framework. The evaluation focuses on the fundamental modelling constructs of microservice ambients and how they support microservitization properties such as utility-driven design, tool heterogeneity and decentralised governance. The evaluation highlights how microservice ambients support analysis, evolution and mobility/location awareness which are significant to quality-driven microservice granularity adaptation.

The evaluation is general and irrespective of the particular application domain and the business competencies in that domain. This generic evaluation aims to ensure the flexibility of microservice ambients and granularity adaptation aspects to be applied in any microservice application. Furthermore, the ambient language by definition supports instantiation of ambient and aspect templates according to the specific microservice application, microservitization utility, granularity adaptation strategy and thresholds of concern. Furthermore, we do appreciate that microservice ambients might be applicable to any architectural style where reasoning about granularity is a design challenge. However, this does not deem microservice ambients any less significant. The state-of-the-practice and -art among microservice adopters is short of a systematic architecture-centric approach to model microservice granularity; this inadequacy is the main gap our contribution targets.

However, we appreciate that our evaluation is in a controlled case study rather than industrial setting. Therefore, further development and research will be required to apply and evaluate our modelling approach in an industrial setting. In [13] a middleware is contributed which supports the ambient language (despite in a proprietary setting). This middleware can accelerate applying our modelling approach in an industrial setting. It is worth noting however such a practical deployment of our microservice ambients and granularity adaptation aspects would turn our contributions into a modelling tool for microservice industry rather than a conceptual modelling approach. In that context, further human-centred evaluation criteria [80] would be necessary to evaluate such a tool (e.g., ease of learning, multiple use, ease of use). Furthermore, only in such a setting can the computational overhead of the microservice ambients and granularity adaptation aspects be assessed quantitatively.

Moreover, although we made our best effort to provide granularity adaptation strategy examples corresponding to the most common microservitization utilities in Section B,we appreciate that further investigation is required to assess the impact of more microservitization drivers on microservice granularity adaptation strategies.

7.2.3 Future Work

The inclusion of explicit triggers in our granularity adaptation aspects paves the way to an interesting research dimension: studying the measurable variables which represent each utility that can be enhanced by microservitization. For example, attachment number is used as a variable if microservitization is aimed to reduce logical dependencies across microservices. The total number of architectural elements, the ratio of functional elements to microservice ambients and the number of attachments between functional elements are further examples of variables that can measure logical dependency reduction.Another important future research direction is assessing the causal relationship between the triggering a granularity adaptation aspect and subsequent negative or positive impacts on the runtime variables monitored by the microservice ambients. This is significant because we appreciate that ultimately the implementation choices made when enforcing a granularity adaptation strategy can have an impact on the resulting behaviour of architecture.

Furthermore, we envision granularity adaptation aspects can be extended to capture conflicts among them. For each possible conflict,transactions can be defined to resolve it. One direction of future research therefore is to cross-reference granularity adaptation strategies into a directed graph that the research community can use to form a reference catalogue for microservice adopters. There are already promising attempts at this in the industrial community [275].

7.3 RESEARCH QUESTION 3

Objective reasoning about the added value of architectural design decisions regarding microservice granularity — can the trade-off between predicted benefit

and cost of pursuing microservice granularity adaptation (or not) be objectively reasoned about and tracked under uncertainty?

7.3.1 Contribution

In Chapter 4, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem. We further propose a novel evaluation process for dynamically evaluating the granularity adaptation design decisions under uncertainty. Our process is based on a novel combination of real options and the concept of Bayesian surprises. Our evaluation process dynamically determines at runtime if it would be worth adapting granularity from the perspective of introducing added value. The argument is that granularity adaptation is only worthwhile if it will introduce added value to the architecture. Moreover, this added value of granularity adaptation is a moving target and can be best tracked and analysed at runtime.

Unlike classical design time usage of binomial options analysis in software architectures [330, 20, 248, 329] which statically reasons about added value predictions at design time, we apply the analysis at runtime. We extend the formulation to cater for runtime changes in the added value which can be enabled by granularity adaptation. The extension uses Bayesian surprises [44] to enable binomial real options analysis to update such added value calculations iteratively at runtime in response to runtime evidence variable values (e.g. workload variations). Our contribution is the first to link granularity adaptation to its added value under uncertainty at runtime. Furthermore, it is the first to apply Bayesian surprises in the context of microservice granularity adaptation and binomial real options analysis.

7.3.2 Reflection

In Chapter 4, we show the benefits of our evaluation process by comparing it to four representative industrial microservice runtime monitoring tools. The results show the unique benefit of our contribution in linking the technical decision of granularity adaptation to its added value under uncertainty at runtime. We implement Filmflix using Amazon Web Service (AWS) Lambda and use this implementation to highlight how our process can go beyond traditional application of real options analysis in

revealing dynamic added value trends of granularity adaptation in response to runtime evidence. We further show the feasibility of our process to assist microservice adopters in industry by presenting a Java implementation of it.

Despite the evaluation being in a controlled Filmflix setting, we envision that our contribution is flexible to be applied in any microservice application domain due to the following reasons:

- All the input parameters of the evaluation process (including the form of assessed claims) can be instantiated with different values in accordance with the microservice application of concern.
- Our process is flexible so that it can utilise any underlying value, cost and probability estimation technique to provide inputs to our evaluation process.
- Our process is flexible so that it can be applied regardless the tools using to develop, deploy and monitor the microservice application of concern.
- Our process is not specific to a critical or non-critical microservice applications. Instead, the length of each iteration of our evaluation process and the Bayesian surprise tolerance threshold can be adjusted according to the criticality of the microservitization utility of concern.

In Chapter 4, we further show the feasibility of our approach to assist microservice adopters in industry by reporting on a Java implementation of it. This shows the a proof of concept for the practical deployment of our proposed evaluation process. However, we do acknowledge that if this proof of concept was applied in an industrial setting, it needs to be further evaluated using human-centred criteria [80] (e.g., ease of use, evolutionary development). This evaluation would ideally be done through questionnaires and/or surveys targeted at microservice practitioners. Unfortunately, we did not have access to practitioners during our research to conduct this evaluation.

When we tested our Java implementation on the Filmflix controlled experiment, we observed that its performance was acceptable as an interactive tool where no restructuring is done to the architecture on-the-fly. We use this observation to infer that the computational overhead is acceptable for the microservice granularity problem.

We acknowledge that for the planning engine to support re-structuring the architecture on-the-fly, potential computational bottlenecks in our evaluation process need to be resolved. We systematically identify, assess and suggest how these bottlenecks can be resolved in Chapter 6.

7.3.3 Future Work

In the short term future, our evaluation can be further strengthened by observing rather than controlling the workload, setting the claims as alarms within AWS Cloudwatch, and systematically evaluating the scalability of our contribution.

Furthermore, we appreciate the significance of extending our process by using runtime evidence variables to update uncertainty regarding the maintenance and switching cost estimations. Determining the form of measurable runtime evidence required to update these estimations and the formulation of claims in this context are among the interesting challenges to this extension.

Another broad future research direction is to extend our evaluation process with a means to capture the architects' attitude towards the uncertainty of adding value through granularity adaptation. Depending on whether the architects are risk-seekers, risk-neutral or risk-averse, their usage of our contribution would differ. For example, a risk-averse architect might only want our evaluation process to suggest adaptation is the probability of introducing added value is above a certain minimum threshold. We envision that extending the process for such consideration can enhance its flexibility.

7.4 RESEARCH QUESTION 4

Realising microservice granularity adaptation — how can microservice granularity adaptation can be effectively engineered at runtime, combining systematic modelling of granularity adaptation strategies with objective reasoning about added value?

7.4.1 Contribution

In Chapter 5, we contribute an interactive, iterative, runtime planning engine which orchestrates our evaluation process from Chapter 4, microservice ambients and granularity adaptation aspects from 3. The input of the planning engine are candidate granularity adaptation strategies that have been defined by the architect(s) at design time and reasoned about in a simulated environment. We utilise our work on modelling microservices from Chapter 3 using ambients and aspects in defining and simulating these strategies [150]. At runtime, the engine iteratively solicits runtime evidence variable values regarding the deployed architecture. It then reasons about the solicited values using our evaluation process from Chapter 4. If it calls for pursuing adaptation within the current iteration of the planning engine, it provides software architect(s) with recommendations on suitable adaptation strategies or on revisiting unrealistic strategies defined at design time.

7.4.2 Reflection

In Chapter 5, we use Filmflix to evaluate our planning engine in multiple controlled settings to show that there is at least partial guarantee that the suggested configuration or aspect(s) can introduce more added value than the initial microservice architecture. We consider that agile, iterative development is the state-of-the-practice in microservice adoption. Therefore, the Filmflix scale is comparable to the scope considered in a single microservice development iteration. Our controlled experiment setting therefore gives strong insight into the effectiveness of the decision support provided by planning engine. The invocation duration values used in Section 4.5 and reflected on in Section 5.5 are observed directly from the Filmflix implementation rather than using publicly available benchmarks; this allows making our controlled setting more realistic.

We acknowledge that ultimately our planning engine needs to be evaluated in a larger-scale industrial setting (where the workload is observed rather controlled) to accelerate the acceptance of our planning engine in industry. Evaluating the planning engine in an industrial setting can highlight possible side effects of the

suggested granularity adaptation strategies on the architecture (e.g., impacts on testing, configuration settings and data consistency). Examples of larger (yet not industrial) settings where the planning engine could be evaluated include [357, 2, 5].

Despite the evaluation being in a controlled Filmflix setting, we envision our planning engine is flexible enough to be applicable across microservice application domains for the following reasons:

- The planning engine orchestrates the evaluation process with our modelling concepts, both of which are flexible contributions in their own right. Therefore, the planning engine inherits this flexibility property.
- Our planning engine supports reasoning about the microservice granularity problem assuming that the functional requirements of the microservice application are guaranteed. Therefore, our planning engine is flexible such that it can support microservices in application domain.

Since our planning engine incorporates our proposed evaluation process, our Java implementation of this contribution can be seen as a potential for proving the feasibility for the practical deployment of our planning engine.

7.4.3 Future Work

Our contributions in this thesis are in essence all incorporated within the last contribution — an interactive decision support system that addresses the target problem of this thesis: reasoning about the microservice granularity design decision problem. We envision that the following usage contexts can benefit from our decision support in the future:

- *Self-Adaptive microservice architectures:* Our contributions can be used in self-adaptive microservice architectures where candidate strategies are already encoded in the running architecture. A self-adaptive solution would be made up of an adaptation system which is responsible for planning any adaptations when need be, and a managed system which autonomously executes the suitable strategy [92]. The planning engine's role in this context would be similar to

symbiotic feedback loops except the suitable granularity adaptation strategy would be autonomously realised. Realisation only involves switching from one encoded strategy to another without having to take the microservices of concern offline.

- *DevOps:* Our contributions can also be used in a DevOps environment, where there is a feedback loop between the development and operation departments. Architects develop a microservice architecture and hand it to the operational staff who provide them with feedback regarding its potential runtime behaviour. This feedback can be used by the planning engine to suggest suitable granularity adaptation strategies to the architects in the development environment. This feedback can be further enriched if the developers and/or operational staff have previously adopted microservices so they can rely on this previous experience to define more informed candidate adaptation strategies. Realising granularity adaptation in this context is up to the developers rather than the planning engine; traditionally DevOps allow easily adapting granularity of microservices and re-building them. Hence, the decision support is interactive rather than autonomous in this context.

- *Models at runtime:* Another possible usage context is using the planning engine's output to operate on models at runtime. A model at runtime "is a causally connected self-representation of the associated system that emphasizes the structure, behaviour, or goals of the system from a problem space perspective [50, p.23]". Crucially, this model "should provide up-to-date and exact information about the system to drive subsequent adaptation decisions [50, p.23]." This entails reflecting changes in the running system on the model and vice versa. The planning engine's suggestions can allow these changes to give more informed with value-driven insights. The decision support in this case is also interactive such that realising granularity adaptation depends on the architects final decision given insights given by both the model and the planning engine.

In the short term future, we envision the planning engine can be decentralised by replicating its components across microservice ambients to make them autonomic

elements [11]. However, this will raise challenges related to consistency across the autonomic microservice ambients. The MAPE-K loop defines the stages for an autonomous element to plan the adaptation of a running system: monitoring the running system, analysing monitored data, planning the adaptation actions, executing the plan on the running system, and finally updating the knowledge base of the autonomous element to improve the planning in future iterations. This might require revisiting the definition of granularity adaptation aspects or resorting to centralised implementation (although that can potentially introduce a computation bottleneck).

In the long term future, the planning engine can be optimised to automate determination of its target scope, generation of the G and C banks, and enforcing adaptation. Enriching the planning engine with a knowledge base over time accelerates this transition to automation but introduces computational complexity. Building this base can be done by recording results of deploying each output, inspired by reinforcement learning approaches [290]. The planning engine's automation can be evaluated for effectiveness in terms of the output's accuracy and/or the time required to converge to an output. The automatic mode is also useful for cases where the architect has little experience in granularity adaptation anyway.

For any aforementioned usage context and/or optimisation of the planning engine, we acknowledge that an important dimension to consider is its overall cost. There is a trade-off between the planning engine's cost and its benefit to a microservice application as opposed to using ad-hoc reasoning about granularity adaptation. We envision that managing this trade-off involves answering three open research questions: 1) how can the planning engine's cost be quantified, 2) what is the minimum size of a microservice application that can tolerate this cost and, 3) how can this tolerance be objectively justified?

7.5 RESEARCH QUESTION 5

Analysing the extent to which it is possible to render scalability-aware granularity adaptation decisions — how is it possible to systematically analyse the

extent to which a decision support system can render scalability-aware granularity adaptation decisions?

7.5.1 Contribution

In Chapter 6, we contribute to a working catalogue of microservice-specific scalability dimensions and metrics. Our catalogue helps identify dimensions and metrics which are important for the scalability of a given microservice architecture; they need to be considered in order to render scalability-aware granularity adaptation decisions for it. We compile our catalogue by: 1) digesting the state-of-the-practice in decision support systems for microservice granularity adaptation and, 2) reflecting on the sample strategies formulated in Section B to infer additional potentially important scalability dimensions and/or metrics. Secondly, we apply scalability goal-obstacle analysis [104, 91] in the new context of reasoning about microservice granularity adaptation. Applying scalability goal-obstacle analysis to a given microservice architecture helps identify obstacles along each dimension of importance from our catalogue. Applying scalability goal-obstacle analysis to a given decision support system helps analyse the extent to which it can scale to consider all these dimensions and metrics.

Both contributions together aim to address Research Question 5. Our catalogue and application of scalability goal-obstacle analysis provide coherent guidance for: 1) identifying and analysing the dimensions and metrics which are important for the scalability of a microservice architecture and, 2) systematically analysing the extent to which a decision support system can scale to consider all these dimensions and metrics.

We evaluate and discuss our contributions by comparing their usage to both our planning engine and initial Filmflix architecture in Section 1.1 against ad-hoc scalability assessment of the planning engine and Filmflix. This aims to highlight how both contributions can aid rendering scalability-aware granularity adaptation decisions. Finally, we discuss how this analysis can be applied to other microservice architectures and other decision support systems for microservice granularity adaptation.

7.5.2 Reflection

We made our best effort using two cases to clearly illustrate the significance of our contributions to aid rendering scalability-aware granularity adaptation decisions. As discussed in Section 6.5.5, we appreciate that further investigation is required to assess the comprehensiveness of our catalogue and the practicality of transferring goal-obstacle analysis to other microservice applications and decision support for reasoning about microservice granularity adaptation.

7.5.3 Future Work

In the short term future, scalability goal-obstacle analysis can be used to extend our catalogues with the most common scalability dimensions and metrics that can affect reasoning about microservice granularity adaptation at a large-scale.

In the long term future, scalability goal-obstacle analysis can itself be developed into a tool which can automatically or interactively assess the impact of scalability on reasoning about microservice granularity adaptation. We envision such a tool would accelerate and assist systematic (and perhaps autonomous) identification of scalability bottlenecks related to reasoning about microservice granularity adaptation at a large-scale. Another interesting research direction would be to develop guidance for rendering granularity adaptation decisions that are aware of other dimensions of interest (e.g., availability-aware, maintainability-aware, and/or reliability-aware).

7.6 CONCLUDING REMARKS

In this thesis we target a critical, main problem related to the transition towards microservices: ***reasoning about the suitable granularity level of a microservice (i.e. when and how to merge or decompose microservices).*** Initially, we contribute a systematic mapping study to provides a relatively disciplined understanding of the transition to microservices and technical activities underlying it. The results of the study help identify inadequacies in the literature regarding the target problem of the thesis. To address the identified inadequacies, we make a number of contributions in this thesis to provide an interactive decision support system that addresses the target

problem of this thesis. We use a microservice application — Filmflix — as a case study for evaluating each aforementioned contribution using a mixture of qualitative and quantitative evaluation (through controlled experiments). The results of our evaluation provide strong insights regarding the effectiveness of our contributions in providing decision support for the target problem of our thesis. Finally, this thesis contributes to microservice-specific guidance aiming to render scalability-aware granularity adaptation decisions. We evaluate this contribution by comparing its usage against ad-hoc scalability analysis. We hope that our contributions can pave the way for future work along with the mentioned research directions thereby further disciplining the transition to microservices.

Appendix A

Systematic Mapping Study Keywords and Classification Detailed Description

Here we classify the included publications according to two classification frameworks. Initially, we classify them according to the following research approaches, elicited from [360]:

- Evaluation research: these investigate the significance of a problem and/or investigate the feasibility of a contribution in practice.
- Opinion papers: "these contain the author's opinion about what is wrong or good about something, how we should do something, etc [360, p.105]".
- Solution proposals: these "propose a solution technique and argue for its relevance, without a full-blown validation. The technique must be novel, or at least a significant improvement of an existing technique. A proof-of-concept may be offered by means of a small example, a sound argument, or by some other means [250, p.86]."
- Experience chapter: in these publications "the emphasis is on what and not on why. The experience may concern one project or more, but it must be the author's personal experience [360, p.105]." Such papers should "contain a list

of lessons learned by the author from his or her experience. Publications in this category will often come from industry practitioners or from researchers who have used their tools in practice, and the experience will be reported without a discussion of research methods. The evidence presented in the chapter can be anecdotal [360, p.106]."

- Validation research: such publications validate solution proposals which "may have been proposed elsewhere, by the author or by someone else. The investigation uses a thorough, methodologically sound research setup. Possible research methods are experiments, simulation, prototyping, mathematical analysis, mathematical proof of properties, etc [360, p.105]."

- Philosophical chapter: "these papers sketch a new way of looking at things, a new conceptual framework, etc [360, p.105]."

The second classification framework entails categorising the publications under categories derived from our research questions of concern. A publication belongs in a category if it contains any of the corresponding keywords identified in Table A.1.

Research Question	Category	Keywords
What activities are undertaken to adopt microservices?	Architectural design	architectural style, communication mechanism, boundaries, orchestration, service choreography, service registration, service discovery, design patterns, proxy, bulkhead, circuit breaker, router, routing
	Organisation	Conway's law, decentralised governance, cross-functional teams, hierarchical teams
	Operation	Devops, NoOps, configuration settings, operation
	Deployment	Continuous integration, CICD, continuous deployment, deployment pipeline, automated deployment, virtualisation, hypervisors, containerisation, configuration provider
	Development	Heterogenous tools, agile development, extreme programming
	Monitoring	Regression unit testing, health monitoring, cluster monitoring, troubleshooting, debugging, failure

Research Question	Category	Keywords
	Logging	central logging, decentralised logging, profiling, tracing
What modelling approaches are used to define the granularity of a microservice?	Structural	Domain-driven, classes, instances, resources, components, message format, data item, topology, service dependency, nodes, type definition
	Behavioural	Event flow, message stream, activity flow, communication flow, event triggers, execution timeline, use cases
Which quality attributes are considered when reasoning about microservice granularity and how are they captured?	Performance	QoS, efficiency, service contracts, SLA, performance, response time, throughput, performance bottleneck, invocation duration, transaction duration, customer value
	Reliability	Fault tolerance, disaster recovery, single point of failure, resilience, robustness, failure rate, error rate
	Scalability	Auto-scale, scalable, scaling, load balancing, load distribution, completed transactions per second
	Maintainability	Maintainable, changeable, maintenance cost, maintenance overhead, adaptability, changeability, effort cost, expandability, dynamicity
	Complexity	Communication overhead, complexity cost, development cost, tight coupling, low cohesion
How is reasoning about microservice granularity described?	Guidelines	Two-pizza team, lines of code, half-life, agile manifesto, one task, single responsibility, fine-grained functionality, separation of business concerns, high cohesion, loose coupling
	Processes	Iterative, strangler pattern

Table A.1 Inferred classification framework used to classify the included publications in our study

The identification of the keywords is inspired by a microservice-specific systematic mapping study [17] and refined iteratively as more publications were examined. It is worth noting that for all the included publications, we manually categorised them

to ensure that synonyms or partial matches of these keywords are accurately handled. We justify the keywords (italicised below) under each category as follows: *What activities were undertaken to adopt microservices?*

- Architectural design: publications in this category are concerned with all the technical activities which comprise adopting microservices. We use this category to verify whether or not microservice adopters call microservices an *architectural style* and/or *design pattern*. Choosing the generic *communication paradigm* of the architecture (e.g., *orchestration, choreography*) and the more concrete message exchange pattern (e.g., using a *router/proxy*) are also among the technical activities of microservice adoption. In addition, the *boundaries* of the system and individual components of the system is a technical design decision when moving to microservices. The fault tolerance mechanisms of the system also need to be identified (e.g. *bulkhead* and *circuit breaker* patterns). Because microservice applications are highly distributed and scalable, 2 additional technical decisions are critical in microservitization: *service registration* and *service discovery*.

- Organisation: in this category we are concerned with the organisational impact of adopting microservices. The state-of-the-practice in the microservice industry is to motivate *decentralised governance* (i.e. holding the responsibility of building and running [41]) through microservitization [130]. The three most common means of achieving that is through following *Conway's law [130], cross-functional teams, or hierarchical teams [159].* Conway's law states "organisations which design systems ... are constrained to produce designs which are copies of the communication structures of these organisations [84]."

- Operation: publications in this category are concerned with identifying how the system will be governed post-deployment. This includes defining who is responsible for this governance to begin with. The state-of-the-practice is in the microservice industry is 2 alternative approaches: *DevOps* where the governance is shared across a development team and an operations team; and *NoOps*, where the governance is fully the responsibility of the development

team. In addition, operational activities include defining the *configuration settings* and *configuration provider* which the governor of the microservice application needs to follow or in different runtime situations.

- Deployment: this category includes publications that define the activities constituting the *deployment pipeline* of the microservice application. The state-of-the-practice in the microservice industry is 2 alternative approaches [355, 79]: *continuous integration* and *continuous delivery* (abbreviated as CICD), and *automated deployment*. "Continuous integration is a coding philosophy and set of practices that drive development teams to implement small changes and check in code to version control repositories frequently [296]." "Continuous delivery picks up where continuous integration ends. CD automates the delivery of applications to selected infrastructure environments [296]." "Deployment automation allows applications to be deployed across the various environments used in the development process, as well as the final production environments [108]." Regardless of the deployment pipeline, the host on which this pipeline is enforced is another critical decision in this category. The 3 most common approaches for deploying microservices are *virtualisation*, using *hypervisors* and/or *containerisation*.

- Development: publications in this category describe how the transition to microservices impacts software development practices. The state-of-the-practice in microservice development is adopting *extreme programming and agile* practices using *heterogenous tools* (e.g., Kubernetes, Istio, Springboot, fabric8).

- Monitoring: publications in this category are concerned with identifying the alternative rationales of runtime monitoring (e.g. *health, cluster*) of microservice application and the alternative approaches which support monitoring (e.g. *regression unit testing, troubleshooting, debugging* and *failure* identification).

- Logging: in this category we are concerned with where the monitoring results are to be stored (alternatively called *profiling* and *tracing*). The 2 alternative

approaches to this in the microservice industry as we have examined are *central* or *decentralised logging*.

What modelling approaches are used to define the granularity of a microservice?

- Structural: publications in this category are concerned about contributions that capture the structure (alternatively called *topology*) of the microservice architecture. Depending on the nature of the contribution (e.g., *domain-driven* [112]), the units of this structure differ (e.g. *classes, instances, resources, components, data items, messages*, and/or *nodes*). Modelling these units includes capturing the *dependencies* across those units and their *types*.
- Behavioural: alternative to the approach above, publications in this category capture the sequence of actions in the microservice application rather the structure of the units constituting the application. This sequence can be in the form of *events, messages, activities, communication, execution steps* and/or *use cases*.

Which quality attributes are considered when reasoning about microservice granularity and how are they captured?

- Performance: wherever the main driving force of reasoning about granularity is performance (alternatively called *QoS, long-term value, efficiency or customer value*), the publication is put under this category. There means of capturing this objectively include *response time, throughput, invocation duration*, identifying the *performance bottlenecks* and/or *transaction duration*. Thresholds on these metrics are captured in *service contracts, service level agreements* (abbreviated as SLAs). We subsume service contracts as "an agreement between the a consumer and provider service about the format of data that they transfer between each other. Normally, the format of the contract is defined by the consumer and shared with the corresponding provider. Afterwards, tests are being implemented in order to verify that the contract is being kept [237]."
- Reliability: publications that imply or explicitly focus on enhancing reliability as the primary driving force when reasoning about granularity are included in

this category. Reliability entails exhibiting *fault tolerance, disaster recovery, resilience*, eliminating *single points of failure* and/or *robustness*. Reliability is captured objectively in terms of *failure/error rate*.

- Scalability: publications that reason about granularity in terms of how it enhances the scalability of the architecture are included in this category. Exhibiting scalability entails employing strategies such as *auto-scaling, load balancing* and/or *load distribution*. Measuring scalability objectively can be done by looking at the *number of completed transactions per second* (or per unit of time more generally).

- Maintainability: exhibiting maintainability entails exhibiting *adaptability, changeability, expandability* and/or *dynamicity*. These properties are objectively captured in terms of *maintenance cost, maintenance overhead*, and/or *effort cost*. Publications concerned with reasoning about granularity in terms of enhancing maintainability are included in this category.

- Complexity: minimising complexity entails following 2 crucial design principles: *tight coupling* and/or *low cohesion*. This is measured in terms of *communication overhead, complexity cost*, and/or *development cost*. Publications where reasoning about granularity considers and/or measures its complexity are included in this category.

How is reasoning about microservice granularity described?

- Guidelines: publications under this category provide decision-making strategies for reasoning about granularity regardless of the steps of applying these strategies. The keywords under this category capture the state-of-the-practice guidelines.

- Processes: publications under this category are more elaborate in the sense that they enrich the granularity adaptation strategies with a sequence for applying them. The keywords under this category capture the alternative state-of-the-practice processes encountered for reasoning about microservice granularity.

Appendix B

Sample Granularity Adaptation Strategies

In this section, we outline a number of granularity adaptation strategies that can be formulated in the *GranAdapt* aspect of a microservice ambient. The strategies are inspired by practices from other utility-driven fields (e.g., SOAs [289, 341], enterprise integration [157], lean architecture development [85] and agile software development [238]). We leverage on these practices by assessing their impact on microservice granularity and present this impact as granularity adaptation strategies with different driving forces and rationale. We present each strategy using the following structure inspired by [10]:

- ***Rationale***, describing intuition behind the strategy.
- ***A problem context***, describing the situation where the strategy could be suitable.
- ***A problem***, describing what exactly the strategy aims for.
- ***Driving forces***, describing the utilities that could be enhanced by adopting this strategy (e.g., data integrity, minimum response time)
- ***Solution***, outlining the actions comprising the strategy.
- ***Resulting context***, in terms of the residual issues which adopting the strategy would introduce.

- ***Related strategies and practices*** that can address these residual issues which can be alternatives or supplements to applied granularity adaptation strategy.

This structure is inspired from the architectural design patterns domain. Patterns convey knowledge in a flexible manner without claiming to be definitive step-by-step specifications [289]. Different patterns have different motivations and therefore represent different design trade-offs [85]. The problem context where the pattern is applied dictates the level of detail at which the pattern should be defined. The more detailed the definition of the pattern, the more likely it is to cross the boundary into a specification rather than a flexible pattern. In essence, granularity adaptation aspects can be likened to architectural design patterns since they facilitate capturing granularity adaptation strategies in a definitive yet flexible manner such that an aspect can be utilised multiple times.

B.1 CROSS-FUNCTIONAL TEAMS

Rationale: This strategy is inspired by Conway's law, which states that organisations usually design systems to replicate their communication structure [130]. This law has been put in action in the microservices context to enforce a DevOps culture, where teams are aligned with business capabilities rather than silos [130]. Here we apply the same law but from a different angle, revising the granularity of the business capabilities around which the teams can be aligned.

Problem Context: Where a DevOps organisational structure is to be adopted in a firm, cross-functional teams will be responsible for each microservice(s). Referring to the running example, we consider a problem scenario to illustrate the context. We consider the functional element regulating the reviews uploaded by system users is encapsulated by the same microservice ambient (i.e. assumed by the same silo) as another functional element regulating the age restrictions on displayed media content.

Problem: Aligning the microservice architecture with the firm's organisational structure.

Driving Forces:

- Increasing the productivity within the firm, thereby facilitating shorter release cycles.
- Focussing the ownership of the microservice within a specific team, thereby enhancing accountability in the firm's culture.
- Facilitating more freedom in microservice development choices.

Solution: The granularity adaptation aspects within a team's microservice ambients need to evaluate the functional requirements encapsulated by each ambient. In case certain functional requirements define low level constraints specific to a certain microservice, they need to be encapsulated by a separate ambient from other unrelated functional requirements.

Resulting Context: A residual issue of this strategy is defining the knowledge sharing requirements across these teams.

Related strategies and practices: Behavioural practices such as "shallow silos" across the decomposed ambients [236] can be introduced as a result of applying this strategy. With shallow silos, frequent meetings are arranged within the team aligned to an ambient. In addition however much less frequent, shorter meetings are organised across those teams to exchange decisions rather discuss them.

B.2 POLYGLOT PERSISTENCE:

Rationale: Polyglot persistence has been proposed by microservice proponents [127] motivated also by enhancing autonomy of the microservices. We leverage on that with the implication which the separation of data would have on the data format translation requirements and the locality of this data translation functionality.

Problem Context: Where a prohibitive amount of data format translation occurs within a microservice ambient, computation resources and maintenance costs can throttled for data format translation purposes. We assume here that a translation functional component handles the data translation requirements within a microservice ambient. For example, system user personal information can be initially encapsulated by the same microservice ambient as sales record information. Different record

lengths and data formats for each type of information can lead to too much data translation overhead.

Problem: Optimising the microservice architecture for data format transparency.

Driving Forces:

- Reducing the data dependencies across microservices, thereby enhancing the autonomy of each individual microservice; the business functionality is coupled only with the data it requires.
- Standardising message formats across microservices boundaries, thereby enhancing the replaceability of the individual microservices and the evolvability of the overall architecture.

Solution: The granularity adaptation aspect monitors the invocation rate of the data translation component it encapsulates. In case the invocation rate is too high, the granularity adaptation aspect of such a microservice ambient needs to create separate microservice sub-ambients each encapsulating compatible data formats. The newly formed microservice ambients do not need the data format translation functionality. In case data exchange is required across the newly formed sub-ambients, this is handled by services published through the ports of these sub-ambients if need be.

Resulting Context: A residual issue to be addressed here however is ensuring the consistency of data in cases where transaction need to span multiple services (in this case multiple microservice ambients).

Related Strategies and Practices: The residual issue can be addressed by the event collaboration [131] communicational mechanism. Here the service providers publish events onto a queue for other service consumers to intercept events they are interested in. The consumers of events can further be organised serially or hierarchically to avoid overloading the event bus [284].

B.3 EXTENSIBILITY POINT

Rationale: Proponents of microservices have proposed the concept of extensibility points in the context of requirement definition [283]. In particular, extensibility points have been proposed with the motivation of letting the end user drive the service

provisioning through contracts where they define their requirements (as outlined below). We leverage on this application by studying the boundary separating the “stable" and “frequently changing" functionalities more explicitly.

Problem Context: Similar to change clustering, this strategy is also triggered by change events. However, this strategy is concerned with the differences in change rates across functional rather than simultaneous change of functional elements. For example, support for different client browsers causes changes in FRs related to media rendering. However, FRs related to fetching the media content from content distribution networks remains unchanged.

Problem: Simplifying new feature introduction through granularity adaptation.

Driving forces:

- Facilitating shorter release cycles by reducing new feature introduction effort.
- Standardising the interfaces providing the core functionality thereby optimising for the evolvability of the architecture.

Solution: For a microservice ambient encapsulating several functional elements, the granularity adaptation aspect evaluates the relative change rate across these functional elements. In cases where a functional element changes at a significantly higher rate than the others, the granularity adaptation aspect triggers creating new nested microservice ambients. One microservice ambient encapsulates the frequently changing functional elements while the other encapsulates the more stable functional elements. The exact number of nested microservice ambients created can be elaborated by further iterations of applying this pattern, taking a subset of microservice ambients as input each time. Microservice ambients encapsulating frequently changing FRs are extensibility points in the system. In case there are several extensibility points related to a single “more stable" microservice ambient, this strategy can be leveraged with a scheduling aspect. In particular, when the granularity adaptation aspect triggers the formation of the “stable" microservice ambient, it can that dictates this microservice ambient contains a scheduling aspect to coordinate the interaction between the connected extensibility point microservice ambients.

Resulting Context: A residual issue here is how the stable FRs are published transparently to the extensibility points.

Related Practices and Strategies: Communication mechanisms such as “dumb pipes and smart filters" can be used for the residual issue here. Messages between the service consumer and provider pass through a “dump pipe" unaware of the payload it is carrying. However, this pipe contains a sequence of “smart filters" (the stable FRs) which apply the processing required by the messages in transit. The filters expose their services through standardised interfaces.

B.4 BULKHEAD

Rationale: The bulkhead strategy has its roots in agile development practices [238]. Therefore, outside the context of microservices, this strategy is motivated by agility and (indirectly) enhanced productivity. By applying this strategy in the context of granularity adaptation, we explicitly locate and isolate more vulnerable functionalities and embrace this likelihood of failure in the design of microservices.

Problem context: In contrast to change rate, this strategy is triggered by the failure rate of the encapsulated functional components. Consider a functional element implementing the regulation of movie reviews and another implementing the upload of reviews to the client browser. Both elements reside in the same microservice ambient. However, the failure rate of the latter is much higher than the former.

Problem: Isolating vulnerabilities in the microservice architecture.

Driving forces:

- Enhancing the evolvability of the architecture through isolating vulnerable business functionalities.
- Facilitating a more robust architecture by localising failure.

Solution: The granularity adaptation aspect of a microservice ambient encapsulating several connected functional elements monitors their failure rates. In cases where the failure rate of a functional element is too high, the granularity adaptation aspect triggers the encapsulation of a new microservice sub-ambient (a new “bulkhead") that encapsulates the vulnerable functional element.

Resulting Context: A residual issue here is increasing the fault tolerance of the “bulkhead" microservice ambient.

Related Practices and Strategies: The residual issue can be addressed using fault tolerance development practices such as the circuit breaker[132]. Here a wrapper circuit breaker object protects a business functionality from being overloaded with requests beyond a certain threshold. Other practices such as concurrency thread pools and the semaphores optimise for the availability of a service [78].

References

[1] Service-oriented architecture ontology, version 2.0. URL https://publications.opengroup.org/c144.

[2] The spring petclinic community. URL https://spring-petclinic.github.io/. Accessed: 05 November 2015.

[3] W. Schroeder A. Rajasekar, M. Wan. *Micro-Services: A Service-Oriented Paradigm for Scalable, Distributed Data Management*. Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management, 2012.

[4] C. A. J. Acevedo, J. P. Gómez y Jorge, and I. R. Patiño. Methodology to transform a monolithic software into a microservice architecture. In *2017 6th International Conference on Software Process Improvement (CIMPS)*, pages 1–6, Oct 2017. doi: 10.1109/CIMPS.2017.8169955.

[5] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi. Benchmark requirements for microservices architecture research. In *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering*, ECASE '17, pages 8–13, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-0417-5. doi: 10.1109/ECASE.2017..4. URL https://doi.org/10.1109/ECASE.2017..4.

[6] A. Agrawala, J. Krause, and S. Vestal. Domain-specific software architectures for intelligent guidance, navigation and control. In *IEEE Symposium on Computer-Aided Control System Design*, pages 110–116, March 1992. doi: 10.1109/CACSD.1992.274442.

[7] A. Agrawala, J. Krause, and S. Vestal. Domain-specific software architectures for intelligent guidance, navigation and control. In *Computer-Aided Control System Design, 1992. (CACSD), 1992 IEEE Symposium on*, pages 110–116, Mar 1992. doi: 10.1109/CACSD.1992.274442.

[8] M. Ahmadvand and A. Ibrahim. Requirements reconciliation for scalable and secure microservice (de)composition. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 68–73, Sept 2016. doi: 10.1109/REW.2016.026.

[9] F. Al-Rebiesh. *Adaptively Improving Performance Stability of Cloud Based Application Using the Modern Portfolio Theory*. PhD thesis, School of Computer Science, 2016.

[10] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. OUP USA, 1977. ISBN 9780195019193. URL https://books.google.co.uk/books?id=hwAHmktpk5IC.

[11] N. Ali and C. Solis. Self-adaptation to mobile resources in service oriented architecture. In *2015 IEEE International Conference on Mobile Services*, pages 407–414. IEEE, 2015.

[12] N. Ali, C. Solís, and I. Ramos. Comparing architecture description languages for mobile software systems. In *Proceedings of the 1st International Workshop on Software Architectures and Mobility*, SAM '08, pages 33–38, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-022-7. doi: 10.1145/1370888.1370897. URL http://doi.acm.org/10.1145/1370888.1370897.

[13] Nour Ali, Isidro Ramos, and Carlos Solís. Ambient-prisma: Ambients in mobile aspect-oriented software architecture. *Journal of Systems and Software*, 83(6):937 – 958, 2010. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2009.12.009. URL http://www.sciencedirect.com/science/article/pii/S0164121209003161. Software Architecture and Mobility.

[14] H. Alipour and Y. Liu. Online machine learning for cloud resource provisioning of microservice backend systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2433–2441, Dec 2017. doi: 10.1109/BigData.2017.8258201.

[15] Robert John Allen. *A Formal Approach to Software Architecture*. PhD thesis, Pittsburgh, PA, USA, 1997. AAI9813815.

[16] W. H. C. Almeida, L. de Aguiar Monteiro, R. R. Hazin, A. C. de Lima, and F. S. Ferraz. Survey on microservice architecture-security, privacy and standardization on cloud computing environment. In *The Twelfth International Conference on Software Engineering Advances (ICSEA 2017)*, page 210, 2017.

[17] N. Alshuqayran, N. Ali, and R. Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications*, 2016.

[18] N. Alshuqayran, N. Ali, and R. Evans. Towards micro service architecture recovery: An empirical study. In *IEEE International Conference on Software Architecture 2018*, Apr 2018.

[19] *AWS Lambda Metrics*. Amazon Web Services, 2017. http://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions-metrics.html.

[20] M. Amram and N. Kulatilaka. *Real Options: Managing Strategic Investment in an Uncertain World*. Harvard Business School Press, Cambridge, Massachusetts, Dec 1999.

[21] M. Andrawos and M. Helmich. *Cloud Native programming with Golang: Develop microservice-based high performance web apps for the cloud with Go*. Packt Publishing, 2017. ISBN 9781787127968. URL https://books.google.co.uk/books?id=NvNFDwAAQBAJ.

[22] A. I. Anton. Goal-based requirements analysis. In *Proceedings of the 2Nd International Conference on Requirements Engineering (ICRE '96)*, 1996. ISBN 0-8186-7252-8. URL http://dl.acm.org/citation.cfm?id=850944.853130.

[23] A. I. Anton. *Goal identification and refinement in the specification of software-based information systems*. PhD thesis, Georgia Institute of Technology, 1997.

[24] N. Ashikhmin, G. Radchenko, and A. Tchernykh. Raml-based mock service generator for microservice applications testing. In V. Voevodin and S. Sobolev, editors, *Supercomputing*, pages 456–467, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71255-0.

[25] T. Asik and Y. E. Selcuk. Policy enforcement upon software based on microservice architecture. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 283–287, June 2017. doi: 10.1109/SERA.2017.7965739.

[26] U. Aßmann, S. Götz, J. Jézéquel, B. Morin, and M. Trapp. *A Reference Architecture and Roadmap for Models@run.time Systems*, pages 1–18. Springer International Publishing, Cham, 2014. ISBN 978-3-319-08915-7. doi: 10.1007/978-3-319-08915-7_1. URL https://doi.org/10.1007/978-3-319-08915-7_1.

[27] P. Assouline and V. Grazi. Perspective on architectural fitness of microservices. https://www.infoq.com/articles/Microservices-Architectural-Fitness, aug 2017. Accessed: 19 September 2017.

[28] J. Asundi, R. Kazman, and M. Klein. Using economic considerations to choose among architecture design alternatives. Technical Report CMU/SEI-2001-TR-035, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2001. URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5785.

[29] Istio Authors. Istio, Aug 2018. URL https://istio.io/. Accessed: 17 July 2018.

[30] *What is AWS Lambda?* AWS, 2017. http://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[31] Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, and Hans van der Vliet. *Software Architecture Knowledge Management - Theory and Practice*. Springer-Verlag Berlin Heidelberg, 2009.

[32] T. Baines. Digitalisation and servitization: the competitive advantage? https://www.advancedservicesgroup.co.uk/single-post/2018/04/04/Digitalisation-and-servitization-the-competitive-advantage. Accessed: 13 October 2015.

[33] T.S. Baines, H.W. Lightfoot, O. Benedettini, and J.M. Kay. The servitization of manufacturing: A review of literature and reflection on future challenges. *Journal of Manufacturing Technology Management*, 20(5):547–567, 2009. doi: 10.1108/17410380910960984. URL http://dx.doi.org/10.1108/17410380910960984.

[34] K. Bakshi. Microservices-based software architecture and approaches. In *2017 IEEE Aerospace Conference*, pages 1–8, March 2017. doi: 10.1109/AERO.2017.7943959.

[35] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33 (3):42–52, May 2016. ISSN 0740-7459. doi: 10.1109/MS.2016.64.

[36] C. Y. Baldwin and K. B. Clark. *Design Rules, Volume 1 The Power of Modularity*. The MIT Press, 2000.

[37] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufman Publishers Inc., 1998. ISBN 9781493303571. URL https://books.google.co.uk/books?id=hNCrcQAACAAJ.

[38] V. Baraiya and V. Singh. Netflix conductor: A microservices orchestrator. https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40, Dec 2016. Accessed: 19 January 2017.

[39] F. Barbier. Mde-based design and implementation of autonomic software components. In *2006 5th IEEE International Conference on Cognitive Informatics*, volume 1, pages 163–169, July 2006. doi: 10.1109/COGINF.2006.365692.

[40] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. SEI Series in Software Engineering. Pearson Education, 2015. ISBN 9780134049878. URL https://books.google.com.sa/books?id=fcwkCQAAQBAJ.

[41] Dr. Gopala Krishna Behara. Microservices governance: A detailed guide. https://blog.leanix.net/en/microservices-governance, Jul 2018. Accessed: 23 April 2018.

[42] B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H.H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 297–308, 2002. doi: 10.1109/ICDE.2002.994738.

[43] N. Bencomo and A. Belaggoun. Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, volume 7830 of *Lecture Notes in Computer Science*, pages 221–236. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-37421-0. doi: 10.1007/978-3-642-37422-7_16. URL http://dx.doi.org/10.1007/978-3-642-37422-7_16.

[44] N. Bencomo and A. Belaggoun. A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 460–463, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591118.

[45] N. Bencomo, A. Belaggoun, and V. Issarny. Dynamic decision networks for decision-making in self-adaptive systems: A case study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 113–122, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-4401-2. URL http://dl.acm.org/citation.cfm?id=2663546.2663565.

[46] C. Berger, B. Nguyen, and O. Benderius. Containerized development and microservices for self-driving vehicles: Experiences best practices. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 7–12, April 2017. doi: 10.1109/ICSAW.2017.56.

[47] T. Betts. Q&a with susan fowler on production-ready microservices. https://www.infoq.com/news/2017/01/production-ready-microservices, Jan 2017. Accessed: 14 May 2017.

[48] P. Bianco, R. Kotermanski, and P. F. Merson. Evaluating a service-oriented architecture. Technical Report CMU/SEI-2007-TR-015, Carnegie Mellon University, 2007.

[49] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher. *Value-Based Software Engineering*. Springer Berlin Heidelberg, 2006. ISBN 9783540292630. URL https://books.google.co.uk/books?id=CAlM6nNPcsgC.

[50] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. *Computer*, 42 (10):22–27, October 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.326. URL https://doi.org/10.1109/MC.2009.326.

[51] B. W. Boehm and K. J. Sullivan. Software economics: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 319–343, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336584. URL http://doi.acm.org/10.1145/336512.336584.

[52] L. C. Briand, S. Morasca, and V. R. Basili. Measuring and assessing maintainability at the end of high level design. In *Proceedings of the Conference on Software Maintenance*, ICSM '93, pages 88–97, Washington, DC, USA, 1993. IEEE Computer Society. ISBN 0-8186-4600-4. URL http://dl.acm.org/citation.cfm?id=645542.658018.

[53] J. Brilhante, R. Costa, and T. Maritan. Asynchronous queue based approach for building reactive microservices. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, WebMedia '17, pages 373–380, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5096-9. doi: 10.1145/3126858.3126873. URL http://doi.acm.org/10.1145/3126858.3126873.

[54] A. Brogi, A. Canciani, D. Neri, L. Rinaldi, and J. Soldani. Towards a reference dataset of microservice-based applications. In Antonio Cerone and Marco Roveri, editors, *Software Engineering and Formal Methods*, pages 219–229, Cham, 2018. Springer International Publishing.

[55] D. Bryant. The seven (more) deadly sins of microservices. https://www.infoq.com/presentations/microservices-antipatterns-2016, Jul 2016. Accessed: 25 August 2016.

[56] D. Bryant. The economics of microservices: Phil calçado recommends avoiding 'microliths' at craftconf. https://www.infoq.com/news/2017/05/economics-microservices, may 2017. Accessed: 18 February 2018.

[57] D. Bryant. Observability and avoiding alert overload from microservices at the financial times. https://www.infoq.com/articles/observability-financial-times, nov 2017. Accessed: 21 January 2018.

[58] D. Bryant. Microservices: The organisational and people impact. https://gotocph.com/2017/sessions/295, oct 2017. Accessed: 10 November 2017.

[59] D. Bryant. Shrinking microservices to functions: Adrian cockcroft discusses serverless at microxchg. https://www.infoq.com/news/2017/02/microxchg-microservice-functions, feb 2017. Accessed: 04 October 2017.

[60] R. Burns. *Advanced Control Engineering*. Butterworth-Heinemann, 2001. ISBN 9780080498782. URL https://books.google.co.uk/books?id=DovwVQu6ImsC.

[61] S. Butler M. Shaw C. Scaffidi, A. Arora. A value-based approach to predicting system properties from design. In *5th EDSER*, 2005.

[62] P. Calçado. Layering microservices. http://philcalcado.com/2018/09/24/services_layers.html, Sept 2018. Accessed: 12 December 2018.

[63] P. Calcado. No free lunch, indeed: Three years of micro-services at soundcloud. http://www.infoq.com/presentations/soundcloud-microservices, Jan . Accessed: 05 October 2017.

[64] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9): 69–77, September 2012. ISSN 0001-0782. doi: 10.1145/2330667.2330686. URL http://doi.acm.org/10.1145/2330667.2330686.

[65] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir. Adaptive model learning for continual verification of non-functional properties. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 87–98, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2733-6. doi: 10.1145/2568088.2568094. URL http://doi.acm.org/10.1145/2568088.2568094.

[66] Luca Cardelli. *Abstractions for Mobile Computation*, pages 51–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-48749-4. doi: 10.1007/3-540-48749-2_4. URL http://dx.doi.org/10.1007/3-540-48749-2_4.

[67] L. Carlson. What are microservices? lightweight software development explained. https://www.infoworld.com/article/3237697/application-development/what-are-microservices-lightweight-software-development-explained.html, nov 2017. Accessed: 13 December 2017.

[68] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in eflow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, CAiSE '00, pages 13–31, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67630-9.

[69] O. Celiku, D. Garlan, and B. Schmerl. Augmenting architectural modeling to cope with uncertainty. international workshop on living with uncertainty. In *DCC+ 02] [GH94] [GMW00] [Hil96] Daniel*, 2007.

[70] T. Cerny. Aspect-oriented challenges in system integration with microservices, soa and iot. *Enterprise Information Systems*, 2018. doi: 10.1080/17517575.2018.1462406. URL https://doi.org/10.1080/17517575.2018.1462406.

[71] T. Cerny, M. J. Donahoo, and J. Pechanec. Disambiguation and comparison of soa, microservices and self-contained systems. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '17, pages 228–235, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5027-3. doi: 10.1145/3129676.3129682. URL http://doi.acm.org/10.1145/3129676.3129682.

[72] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. Contextual understanding of microservice architecture: Current and future directions. *SIGAPP Appl. Comput. Rev.*, 17(4):29–45, January 2018. ISSN 1559-6915. doi: 10.1145/3183628.3183631. URL http://doi.acm.org/10.1145/3183628.3183631.

[73] K. Channabasavaiah, K. Holley, and E. Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16:727–728, 2003.

[74] L. Chen. Microservices: Architecting for continuous delivery and devops. In *IEEE International Conference on Software Architecture (ICSA 2018)*, 03 2018.

[75] BettyH.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller,

S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In BettyH.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02160-2. doi: 10.1007/978-3-642-02161-9_1. URL http://dx.doi.org/10.1007/978-3-642-02161-9_1.

[76] S. Cheng, D. Garlan, and B. Schmerl. Self-star properties in complex information systems. In Ozalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, and Stefano Leonardi, editors, *Making Self-adaptation an Engineering Reality*, pages 158–173. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 3-540-26009-9, 978-3-540-26009-7. URL http://dl.acm.org/citation.cfm?id=2167575.2167589.

[77] N. A. Chriss. *The Black-Scholes and Beyond and the Black-Scholes and Beyond Interactive Toolkit: A Step-By-Step Guide to In-Depth Option Pricing Models*. McGraw-Hill Trade, 1997. ISBN 9780786311408. URL https://books.google.co.uk/books?id=MLjFPQAACAAJ.

[78] Ben Christensen. Fault tolerance in a high volume, distributed system. http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html, feb 2012.

[79] A. Ciuffoletti. Automated deployment of a microservice-based monitoring infrastructure. *Procedia Computer Science*, 68:163 – 172, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.09.232. URL http://www.sciencedirect.com/science/article/pii/S187705091503077X. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[80] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28:626–643, 1996.

[81] J. Cámara, D. Garlan, W. Gu Kang, W. Peng, and B. Schmerl. Uncertainty in self-adaptive systems categories, management, and perspectives. Technical Report CMU-ISR-17-110, Carnegie Mellon University, jul 2017.

[82] A. Cockroft. State of the art in microservices. https://www.infoq.com/presentations/microservices-comparison-evolution, Jun 2015. Accessed: 30 September 2015.

[83] A. Cockroft. Gluecon monitoring microservices and containers: A challenge. http://www.slideshare.net/adriancockcroft/gluecon-monitoring-microservices-and-containers-a-challenge, May 2015. Accessed: 19 July 2015.

[84] M. E. Conway. How do committees invent. 1968.

[85] J.O. Coplien and G. Bjørnvig. *Lean Architecture: for Agile Software Development*. Wiley, 2011. ISBN 9780470970133.

[86] C. Curlett. Tame microservices complexity with apis. https://www.infoworld.com/article/3111349/application-development/tame-microservices-complexity-with-apis.html, aug 2016. Accessed: 23 August 2017.

[87] M. Danilovic and T. R. Browning. Managing complex product development projects with design structure matrices and domain mapping matrices. *International Journal of Project Management*, 25(3):300 – 314, 2007. ISSN

0263-7863. doi: http://dx.doi.org/10.1016/j.ijproman.2006.11.003. URL http://www.sciencedirect.com/science/article/pii/S0263786306001645.

[88] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1):3 – 50, 1993. ISSN 0167-6423. doi: https://doi.org/10.1016/0167-6423(93)90021-G. URL http://www.sciencedirect.com/science/article/pii/016764239390021G.

[89] A. Davies. Moving base into high-value integrated solutions: a value stream approach. *Industrial and Corporate Change*, 13(5):727–756, 2004. doi: 10.1093/icc/dth029. URL http://icc.oxfordjournals.org/content/13/5/727.abstract.

[90] S. Daya, N. Van Duy, K. Eati, C.M. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins, et al. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2015. ISBN 9780738440811. URL https://books.google.co.uk/books?id=eOZyCgAAQBAJ.

[91] L. D. de Cerqueira. *A framework for the characterization and analysis of software systems scalability*. PhD thesis, University College London (University of London), 2010.

[92] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzé, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In Rogério de Lemos, Holger Giese, HausiA. Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 1–32. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-35812-8. doi: 10.1007/978-3-642-35813-5_1. URL http://dx.doi.org/10.1007/978-3-642-35813-5_1.

[93] Zhamak Dehghani. Zhamak dehghani real world microservices: Lessons from the frontline. https://youtu.be/hsoovFbpAoE, feb 2015. Accessed: 17 March 2016.

[94] S. del Amo Caballero. Micronaut tutorial: How to build microservices with this jvm-based framework. https://www.infoq.com/articles/micronaut-tutorial-microservices-jvm, Oct 2018. Accessed: 12 November 2018.

[95] M. Derakhshanmanesh and M. Grieger. Model-integrating microservices: A vision paper. In *Software Engineering Workshops*, 2016.

[96] Jens Dietrich, Catherine McCartin, Ewan Tempero, and SyedM.Ali Shah. Barriers to modularity - an empirical study to assess the potential for modularisation of java programs. In GeorgeT. Heineman, Jan Kofron, and Frantisek Plasil, editors, *Research into Practice – Reality and Gaps*, volume 6093 of *Lecture Notes in Computer Science*, pages 135–150. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13820-1. doi: 10.1007/978-3-642-13821-8_11.

[97] R.K. Dixit and R.S. Pindyck. *Investment Under Uncertainty*. Princeton University Press, 2012. URL https://books.google.co.uk/books?id=korbtwEACAAJ.

[98] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 28(7):638–653, Jul 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.1019479.

[99] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, December 2006. ISSN 1556-4665. doi: 10.1145/1186778.1186782. URL http://doi.acm.org/10.1145/1186778.1186782.

[100] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. *CoRR*, abs/1606.04036, 2016. URL http://arxiv.org/abs/1606.04036.

[101] N. Dragoni, S. Dustdar, S. Thordal Larsen, and M. Mazzara. Microservices: Migration of a mission critical system. *CoRR*, abs/1704.04173, 2017. URL http://arxiv.org/abs/1704.04173.

[102] N. Dragoni, I. Lanese, S. Thordal Larsen, M. Mazzara, R. Mustafin, and L. Safina. Microservices: How to make your application scale. *CoRR*, abs/1702.07149, 2017. URL http://arxiv.org/abs/1702.07149.

[103] L. Duboc, D. Rosenblum, and T. Wicks. A framework for characterization and analysis of software system scalability. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 375–384, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi: 10.1145/1287624.1287679. URL http://doi.acm.org/10.1145/1287624.1287679.

[104] L. Duboc, E. Letier, and D. S. Rosenblum. Systematic elaboration of scalability requirements through goal-obstacle analysis. *IEEE Transactions on Software Engineering*, 39(1):119–140, Jan 2013. ISSN 0098-5589. doi: 10.1109/TSE.2012.12.

[105] Jr. Edmund M. C., O. Grumberg, and D. Peleg. *Model Checking*. The MIT Press, 1999.

[106] A. Ekárt and S.Z. Németh. Stability analysis of tree structured decision functions. *European Journal of Operational Research*, 160(3):676 – 695, 2005. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2003.10.007. URL http://www.sciencedirect.com/science/article/pii/S0377221703006647. Decision Analysis and Artificial Intelligence.

[107] *Monitoring Logstash*. ElasticSearch, 2017. https://www.elastic.co/guide/en/x-pack/5.5/monitoring-logstash.html.

[108] ElectricCloud. Deployment automation. http://electric-cloud.com/wiki/display/releasemanagement/Deployment+Automation. Accessed: 16 June 2018.

[109] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. The Prentice Hall Service Technology Series from Thomas Erl. Pearson Education, 2005. ISBN 9780132715829. URL https://books.google.co.uk/books?id=y2MALc9HOF8C.

[110] T. Erl. *SOA Principles of Service Design*. The Prentice Hall Service Technology Series from Thomas Erl. Pearson Education, 2007. ISBN 9780132715836. URL https://books.google.co.uk/books?id=mkQJvjR2sX0C.

[111] A. M. Del Esposte, F. Kon, F. M. Costa, and N. Lago. Interscity: A scalable microservice-based open source platform for smart cities. In *The 6th International Conference on Smart Cities and Green ICT*, 2017.

[112] E. J. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, first edition, Aug 2003. ISBN 0321125215.

[113] D. S. Thrane F. Montesi. Packaging microservices (work in progress). In *IFIP International Federation for Information Processing 2017*, pages 131–137. Springer, 2017.

[114] G. Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010. ISBN 9780984618101. URL https://books.google.co.uk/books?id=5UZ-AQAAQBAJ.

[115] V. Farcic. *The DevOps 2.0 Toolkit Automating the Continuous Deployment Pipeline with Containerized Microservices*. Leanpub, 2018.

[116] A. Fernandez, E. Insfran, and S. Abrahão. Usability evaluation methods for the web: A systematic mapping study. *Inf. Softw. Technol.*, 53(8):789–817, August 2011. ISSN 0950-5849. doi: 10.1016/j.infsof.2011.02.007. URL http://dx.doi.org/10.1016/j.infsof.2011.02.007.

[117] A. Filieri, C. Ghezzi, and G. Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2011. ISSN 1433-299X. doi: 10.1007/s00165-011-0207-2. URL http://dx.doi.org/10.1007/s00165-011-0207-2.

[118] T. Fisher. Digital transformation is underpinned by the enterprise it landscape. https://www.capgemini.com/resources/digital-transformation-is-underpinned-by-the-enterprise-it-landscape/, jun 2014. Accessed: 20 September 2015.

[119] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J. Jézéquel. Models in software engineering. pages 97–108. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-01647-9. doi: 10.1007/978-3-642-01648-6_11. URL http://dx.doi.org/10.1007/978-3-642-01648-6_11.

[120] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *IEEE Softw.*, 23(2):62–70, March 2006. ISSN 0740-7459. doi: 10.1109/MS.2006.61. URL https://doi.org/10.1109/MS.2006.61.

[121] L. Florio and E. D. Nitto. Gru: An approach to introduce decentralized autonomic behavior in microservices architectures. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 357–362, July 2016. doi: 10.1109/ICAC.2016.25.

[122] R. Flygare and A. Holmqvist. Performance characteristics between monolithic and microservice-based systems. Master's thesis, , Department of Software Engineering, 2017.

[123] N. Ford. Building microservice architectures. http://nealford.com/downloads/Building_Microservice_Architectures_Neal_Ford.pdf, April 2018. Accessed: 24 June 2018.

[124] The Linux Foundation. Production-grade container orchestration, 2018. URL https://kubernetes.io/. Accessed: 14 June 2017.

[125] M. Fowler. Event interception. http://www.martinfowler.com/bliki/EventInterception.html, Jun 2004. Accessed: 03 September 2015.

[126] M. Fowler. Strangler application. https://www.martinfowler.com/bliki/StranglerApplication.h Jun 2004. Accessed: 05 February 2016.

[127] M. Fowler. Polyglot persistence. http://martinfowler.com/bliki/PolyglotPersistence.html, Nov 2011. Accessed: 19 August 2015.

[128] M. Fowler. Bounded context. https://martinfowler.com/bliki/BoundedContext.html, Jan 2014. Accessed: 13 October 2015.

[129] M. Fowler. Microservices — martin fowler. https://www.youtube.com/watch?v=wgdBVIX9ifA, Jan 2015. Accessed: 14 March 2016.

[130] M. Fowler and J. Lewis. Microservices a definition of this new architectural term. http://martinfowler.com/articles/microservices.html, March 2014. Accessed: 20 December 2015.

[131] Martin Fowler. Event collaboration. https://martinfowler.com/eaaDev/EventCollaboration.htr jun 2006. Accessed: 14 October 2016.

[132] Martin Fowler. Circuitbreaker. http://martinfowler.com/bliki/CircuitBreaker.html, mar 2014. Accessed: 15 October 2015.

[133] Martin Fowler. Microservicepremium. http://martinfowler.com/bliki/MicroservicePremium.html, may 2015. Accessed: 13 August 2015.

[134] P. D. Francesco, I. Malavolta, and P. Lago. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30, April 2017. doi: 10.1109/ICSA.2017.24.

[135] J. Fritzsch. From monolithic applications to microservices guidance on refactoring techniques and result evaluation. Master's thesis, Reutlingen University, 2018.

[136] A. Gamba. Real options valuation: A monte carlo approach. 2003.

[137] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. ISBN 9780321700698. URL https://books.google.co.uk/books?id=6oHuKQe3TjQC.

[138] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37 (10):46–54, October 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.175. URL http://dx.doi.org/10.1109/MC.2004.175.

[139] David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. *ACM SIGSOFT Software Engineering Notes*, 19 (5):175–188, 1994.

[140] M. Garriga. Towards a taxonomy of microservices architectures. In A. Cerone and M. Roveri, editors, *Software Engineering and Formal Methods*, pages 203–218, Cham, 2018. Springer International Publishing. ISBN 978-3-319-74781-1.

[141] X Geerinck. An architecture for resource analysis, prediction and visualization in microservice deployments. Master's thesis, University of Ghent. URL http://lib.ugent.be/fulltxt/RUG01/002/367/136/RUG01-002367136_2017_0001_AC.pdf.

[142] F. George. Challenges in implementing microservices by fred george. https://youtu.be/yPf5MfOZPY0, Aug 2015. Accessed: 17 June 2016.

[143] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the First Workshop on Self-healing Systems*, WOSS '02, pages 33–38, New York, NY, USA, 2002. ACM. ISBN 1-58113-609-9. doi: 10.1145/582128.582135. URL http://doi.acm.org/10.1145/582128.582135.

[144] S. Giallorenzo, I. Lanese, J. Mauro, and M. Gabbrielli. Programming adaptive microservice applications: An aiocj tutorial. *Behavioural Types: from Theory to Tools*, page 147, 2017.

[145] S. Godwin. Cloud-based microservices powering bbc iplayer, jun 2016. URL https://www.infoq.com/presentations/bbc-microservices-aws?utm_campaign=infoq_content&utm_source=infoq&utm_medium=feed&utm_term=Microservices.

[146] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. Towards recovering the software architecture of microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 46–53, April 2017. doi: 10.1109/ICSAW.2017.48.

[147] S. Haselböck, R. Weinreich, and G. Buchgeher. Decision models for microservices: Design areas, stakeholders, use cases, and requirements. In A. Lopes and R.**ogério** de Lemos, editors, *Software Architecture*, pages 155–170, Cham, 2017. Springer International Publishing. ISBN 978-3-319-65831-5.

[148] S. Hassan and R. Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. In *13th IEEE International Conference on Services Computing (SCC)*, San Francisco, USA, jun 2016.

[149] S. Hassan, R. Bahsoon, and N. Bencomo. Minimizing nasty surprises with better informed decision-making in self-adaptive systems. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 134–145, May 2015. doi: 10.1109/SEAMS.2015.13.

[150] S. Hassan, N. Ali, and R. Bahsoon. Microservice ambients: An architectural meta-modelling approach for microservice granularity. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 1–10, 4 2017. doi: 10.1109/ICSA.2017.32.

[151] S. Hassan, R. Bahsoon, and R. Kazman. Microservice transition and its granularity problem: A systematic mapping study. submitted to ACM Computing Surveys (CSUR), 2018.

[152] W. Hasselbring. Microservices for scalability: Keynote talk abstract. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 133–134, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4080-9. doi: 10.1145/2851553.2858659. URL http://doi.acm.org/10.1145/2851553.2858659.

[153] F. Haupt, F. Leymann, A. Scherer, and K. Vukojevic-Haupt. A framework for the structural analysis of rest apis. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 55–58, April 2017. doi: 10.1109/ICSA.2017.40.

[154] D. Haywood. In defence of the monolith, part 1. https://www.infoq.com/articles/monolith-defense-part-1, Mar 2017. Accessed: 19 October 2017.

[155] R. Heinrich, C. Zirkelbach, and R. Jung. Architectural runtime modeling and visualization for quality-aware devops in cloud applications. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 199–201, April 2017. doi: 10.1109/ICSAW.2017.33.

[156] N. Herzberg, C. Hochreiner, O. Kopp, and J. Lenhard. Challenges of microservices architecture: A survey on the state of the practice. In *10th ZEUS Workshop, ZEUS 2018*, 2018.

[157] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012. ISBN 9780133065107. URL https://books.google.co.uk/books?id=qqB7nrrna_sC.

[158] Charles Humble. How buzzfeed migrated from a perl monolith to go and python microservices. https://www.infoq.com/articles/buzzfeed-microservices-migration, jun 2018. Accessed: 25 September 2018.

[159] T. Huston. What is microservice architecture? https://smartbear.com/learn/api-design/what-are-microservices/, Apr 2018. Accessed: 16 June 2018.

[160] J Huttunen. Micro service testing practices in public sector software projects. Master's thesis, Aalto University, 2017. URL https://aaltodoc.aalto.fi/handle/123456789/26673.

[161] IBM. An architectural blueprint for autonomic computing. Technical report, IBM, jun 2006.

[162] D. Iffland. Q&a with intuit's alex balazs. https://www.infoq.com/articles/intuit-alex-balazs-node-services, Jun 2016. Accessed: 13 July 2017.

[163] Investopedia. Compound option. https://www.investopedia.com/terms/c/compoundoption.asp

[164] U. Ismail, D. Rolnick, D. Fabijan, and A. Wallgren. Challenges of micro-service deployments. http://techtraits.com/microservice.html, F 2016. Accessed: 26 April 2016.

[165] Respect IT. A kaos tutorial, oct 2007. URL http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf.

[166] I. Ivkovic and K. Kontogiannis. A framework for software architecture refactoring using model transformations and semantic annotations. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 10 pp.–144, March 2006. doi: 10.1109/CSMR.2006.3.

[167] P. Jamshidi, C. Pahl, S. Chinenyeze, and X. Liu. Cloud migration patterns: A multi-cloud service architecture perspective. In F. Toumani, B. Pernici, D. Grigori, D. Benslimane, J. Mendling, N. Ben Hadj-Alouane, B. Blake, O. Perrin, I. Saleh Moustafa, and S. Bhiri, editors, *Service-Oriented Computing - ICSOC 2014 Workshops*, pages 6–19, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22885-3.

[168] ZL Ji and Y Liu. A dynamic deployment method of micro service oriented to sla. *International Journal of Computer Science Issues ...*, 2016.

[169] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, June 2000. ISSN 1045-9219. doi: 10.1109/71.862209.

[170] P.P. Jogalekar and C.M. Woodside. A scalability metric for distributed computing applications in telecommunications. In V. Ramaswami and P.E. Wirth, editors, *Teletraffic Contributions for the Information Age*, volume 2 of *Teletraffic Science and Engineering*, pages 101 – 110. Elsevier, 1997. doi: https://doi.org/10.1016/S1388-3437(97)80016-9. URL http://www.sciencedirect.com/science/article/pii/S1388343797800169.

[171] M. I. Joselyne, B. Kanagwa, and J. Balikuddembe. A framework to modernize sme application in emerging economies: Microservice architecture pattern approach. 2017.

[172] K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting. Automatic model-driven recovery in distributed systems. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 25–36, Oct 2005. doi: 10.1109/RELDIS.2005.11.

[173] S. Kaiser, G. Podjarny, I. Levine, M. Burgess, and J. Stenberg. The future of microservices and distributed systems: Qcon london microservices panel discussion. https://www.infoq.com/news/2018/03/microservices-future, Mar 2018. Accessed: 12 December 2018.

[174] M. Kalske, N. Mäkitalo, and T. Mikkonen. Challenges when moving from monolith to microservice architecture. In I. Garrigós and M. Wimmer, editors, *Current Trends in Web Engineering*, pages 32–47, Cham, 2018. Springer International Publishing.

[175] R. Kazman, M. Klein, and P. Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000. URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5177.

[176] G. Kecskemeti, A. C. Marosi, and A. Kertesz. The entice approach to decompose monolithic services into microservices. In *2016 International Conference on High Performance Computing Simulation (HPCS)*, pages 591–596, July 2016. doi: 10.1109/HPCSim.2016.7568389.

[177] G. Kecskemeti, A. Kertesz, and A. C. Marosi. Towards a methodology to form microservices from monolithic ones. In *Euro-Par 2016: Parallel Processing Workshops*, pages 284–295, Cham, 2017. Springer International Publishing. ISBN 978-3-319-58943-5.

[178] F. B. Kessler. Astro-captevo. Online, 2014. URL http://das.fbk.eu/astro-captevo.

[179] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu. Efficiency analysis of provisioning microservices. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 261–268, Dec 2016. doi: 10.1109/CloudCom.2016.0051.

[180] T. Killalea. The hidden dividends of microservices. *Commun. ACM*, 59 (8):42–45, July 2016. ISSN 0001-0782. doi: 10.1145/2948985. URL http://doi.acm.org/10.1145/2948985.

[181] B. Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and University of Durham, 2007.

[182] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 273–281, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0. URL http://dl.acm.org/citation.cfm?id=998675.999432.

[183] P. Kleindienst. Implementation and evaluation of a hybrid microservice infrastructure. Master's thesis, Hochschule der Medien, Department 1: Printing and Media, 2017.

[184] F. Klinaku, M. Frank, and S. Becker. Caus: An elasticity controller for a containerized microservice. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 93–98, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5629-9. doi: 10.1145/3185768.3186296. URL http://doi.acm.org/10.1145/3185768.3186296.

[185] S. Klock, J. M. E. M. V. D. Werf, J. P. Guelen, and S. Jansen. Workload-based clustering of coherent feature sets in microservice architectures. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 11–20, April 2017. doi: 10.1109/ICSA.2017.38.

[186] H. Knoche and W. Hasselbring. Using microservices for legacy software modernization. *IEEE Software*, 35(3):44–49, May 2018. ISSN 0740-7459. doi: 10.1109/MS.2018.2141035.

[187] A. Koutsouras, D. Kougioumoutzakis, and I. Kantzavelou. Assessment and security issues in cloud computing services. In *Proceedings of the 19th Panhellenic Conference on Informatics*, PCI '15, pages 165–166, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3551-5. doi: 10.1145/2801948.2802030. URL http://doi.acm.org/10.1145/2801948.2802030.

[188] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, Jun 1994. ISSN 1573-1375. doi: 10.1007/BF00175355. URL https://doi.org/10.1007/BF00175355.

[189] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-oriented Architecture Best Practices*. The Coad series. Prentice Hall Professional Technical Reference, 2005. ISBN 9780131465756. URL https://books.google.co.uk/books?id=R7oGhITYUuUC.

[190] O. Krajicek. Micro-services or solid services. http://www.infoq.com/presentations/microservices-solid, Jul 2015. Accessed: 23 January 2016.

[191] L. Krause. *Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns*. Lucas Krause, 2015. ISBN 9780692424278. URL https://books.google.co.uk/books?id=dd5-rgEACAAJ.

[192] A. Krylovskiy, M. Jahn, and E. Patti. Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30, Aug 2015. doi: 10.1109/FiCloud.2015.55.

[193] P. P. Kukade and Prof. Geetanjali Kale. Auto-scaling of micro-services using containerization. *International Journal of Science and Research (IJSR)*, 4(9), sep 2015.

[194] N. Kulkarni and V. Dwivedi. The role of service granularity in a successful soa realization a case study. In *2008 IEEE Congress on Services - Part I*, pages 423–430, July 2008. doi: 10.1109/SERVICES-1.2008.86.

[195] Y. Kwon and E. Tilevich. Cloud refactoring: automated transitioning to cloud-based services. *Automated Software Engineering*, 21(3):345–372, 2013. ISSN 1573-7535. doi: 10.1007/s10515-013-0136-9. URL http://dx.doi.org/10.1007/s10515-013-0136-9.

[196] P. Leitner, J. Cito, and E. Stöckli. Modelling and managing deployment costs of microservice-based cloud applications. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 165–174, Dec 2016.

[197] A. Levcovitz, R. Terra, and M. T. Valente. Towards a technique for extracting microservices from monolithic enterprise systems. *CoRR*, abs/1605.03175, 2016. URL http://arxiv.org/abs/1605.03175.

[198] S. Li. Understanding quality attributes in microservice architecture. In *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, pages 9–10, Dec 2017. doi: 10.1109/APSECW.2017.33.

[199] B. Limthanmaphon and Y. Zhang. Web service composition with case-based reasoning. In *Proceedings of the 14th Australasian Database Conference - Volume 17*, ADC '03, pages 201–208, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc. ISBN 0-909-92595-X. URL http://dl.acm.org/citation.cfm?id=820085.820122.

[200] D. S. Linthicum. Practical use of microservices in moving workloads to the cloud. *IEEE Cloud Computing*, 3(5):6–9, Sept 2016. ISSN 2325-6095. doi: 10.1109/MCC.2016.114.

[201] M. Little. The difference between soa and microservices? https://www.infoq.com/news/2017/07/soaandmicroservices, Jul 2017. Accessed: 23 September 2017.

[202] H. Loftis. Why microservices matter. https://blog.heroku.com/why_microservices_matter, jan 2015. Accessed: 19 December 2015.

[203] F. Losavio, O. Ordaz, and V. Esteller. Refactoring-based design of reference architecture. *Revista Antioqueña de las Ciencias Computacionales*, 5(1), 2015.

[204] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. Softw. Eng.*, 21(4):336–355, April 1995. ISSN 0098-5589. doi: 10.1109/32.385971. URL http://dx.doi.org/10.1109/32.385971.

[205] L. Maciaszek and Mr B. L. Liong. *Practical Software Engineering: A Case-Study Approach*. Addison Wesley, 2004.

[206] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton. Reference model for service oriented architecture 1.0. *OASIS standard*, 12:18, 2006.

[207] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, London, UK, UK, 1995. Springer-Verlag. ISBN 3-540-60406-5. URL http://dl.acm.org/citation.cfm?id=645385.651497.

[208] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A decentralized redeployment algorithm for improving the availability of distributed systems. In Alan Dearle and Susan Eisenbach, editors, *Component Deployment*, volume 3798 of *Lecture Notes in Computer Science*, pages 99–114. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-30517-0. doi: 10.1007/11590712_8. URL http://dx.doi.org/10.1007/11590712_8.

[209] E. M. Maximilien and M. P. Singh. Toward autonomic web services trust and selection. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 212–221, New York, NY, USA, 2004. ACM. ISBN 1-58113-871-7. doi: 10.1145/1035167.1035198. URL http://doi.acm.org/10.1145/1035167.1035198.

[210] J. D. McGregor, D. P. Gluch, and P. H. Feiler. Analysis and design of safety-critical, cyber-physical systems. *Ada Lett.*, 36(2):31–38, May 2017. ISSN 1094-3641. doi: 10.1145/3092893.3092899. URL http://doi.acm.org/10.1145/3092893.3092899.

[211] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, Toulouse, France, April 22-25 2002.

[212] N. Medvidovic. Moving architectural description from under the technology lamppost. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 2–3, Aug 2006. doi: 10.1109/EUROMICRO.2006.47.

[213] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26 (1):70–93, January 2000. ISSN 0098-5589. doi: 10.1109/32.825767. URL http://dx.doi.org/10.1109/32.825767.

[214] N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor. Using object-oriented typing to support architectural design in the c2 style. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 24–32, New York, NY, USA, 1996. ACM. ISBN 0-89791-797-9. doi: 10.1145/239098.239106. URL http://doi.acm.org/10.1145/239098.239106.

[215] K. Meinke and P. Nycander. Learning-based testing of distributed microservice architectures: Correctness and fault injection. In D. Bianculli, R. Calinescu, and B. Rumpe, editors, *Software Engineering and Formal Methods*, pages 3–10, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-49224-6.

[216] R. Miles. *Antifragile Software Building Adaptable Software with Microservices*. Leanpub, 2016.

[217] Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, and Qiong Feng. Decoupling level: A new metric for architectural maintenance complexity. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 499–510, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1. doi: 10.1145/2884781.2884825. URL http://doi.acm.org/10.1145/2884781.2884825.

[218] M. Mongiello, S. Colucci, E. Vogli, L. A. Grieco, and M. Sciancalepore. Runtime architectural modeling for future internet applications. *Complex and Intelligent Systems*, 2(2):111–124, Jun 2016. ISSN 2198-6053. doi: 10.1007/s40747-016-0020-x. URL https://doi.org/10.1007/s40747-016-0020-x.

[219] O. Mont. Introducing and developing a product-service system (pss) concept in sweden, 2001. ISSN 1650-1675.

[220] F. Montesi and J. Weber. Circuit breakers, discovery, and API gateways in microservices. *CoRR*, abs/1609.05830, 2016. URL http://arxiv.org/abs/1609.05830.

[221] B. Morin, O. Barais, G. Nain, and J. M. Jezequel. Taming dynamically adaptive systems using models and aspects. In *2009 IEEE 31st International Conference on Software Engineering*, pages 122–132, May 2009. doi: 10.1109/ICSE.2009.5070514.

[222] B. Mozaffari. *Microservice Architecture Building microservices with JBoss EAP 7*. RedHat, 1.0 edition, Jun 2016.

[223] D. Müssig, R. Stricker, J. Lässig, and J. Heider. Highly scalable microservice-based enterprise architecture for smart ecosystems in hybrid cloud environments. In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 3: ICEIS,*, pages 454–459. INSTICC, SciTePress, 2017. ISBN 978-989-758-249-3. doi: 10.5220/0006373304540459.

[224] O. Mustafa and J. M. Gómez. Using fuzzy clustering techniques to improve the design of microservices web applications. In *Eureka International Virtual Meeting Eureka 2016 OPTISAD*, dec 2016.

[225] O. Mustafa and J. M. Gómez. Sustainable approach for improving microservices based web application. In *Sustainability Dialogue: International Conference on Sustainability and Environmental Management*, feb 2017.

[226] O. Mustafa, J. Marx Gómez, M. Hamed, and H. Pargmann. Granmicro: A black-box based approach for optimizing microservices based applications. In *From Science to Society*, pages 283–294, Cham, 2018. Springer International Publishing. ISBN 978-3-319-65687-8.

[227] S.C. Myers. *Modern Developments in Financial Management*. Praeger, 1976. ISBN 9780275341008. URL https://books.google.co.uk/books?id=ElkoAQAAIAAJ.

[228] O. Kopp J. Lenhard N. Herzberg, C. Hochreiner. Analyzing the relevance of soa patterns for microservice-based systems. In *10th ZEUS Workshop, ZEUS 2018*, may 2018.

[229] D. Namiot and M. sneps sneppe. On micro-services architecture. 2:24–27, 09 2014.

[230] S. Newman. Practical considerations for microservice architectures. part 1. (sam newman, uk). https://www.youtube.com/watch?v=12n9E5L6qgs, Dec 2014. Accessed: 23 March 2015.

[231] S. Newman. Practical considerations for microservice architectures. part 2. (sam newman, uk). https://www.youtube.com/watch?v=cU0J0w6sFoA, Dec 2014. Accessed: 18 May 2015.

[232] S. Newman. Microservices talk with sam newman. https://www.youtube.com/watch?v=GDVcUM5wbxU, Jun 2015. Accessed: 07 October 2015.

[233] Sam Newman. *Building Microservices*. O'Reilly Media, first edition, feb 2015. ISBN 491950358.

[234] P. Nguyen and K. Nahrstedt. Monad: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 187–196, July 2017. doi: 10.1109/ICAC.2017.38.

[235] Y. Niu, F. Liu, and Z. Li. Load balancing across microservices. In *IEEE International Conference on Computer Communications*, 2018.

[236] Dan North. Dan north - patterns of effective delivery, jun 2012. URL https://vimeo.com/43659070. Accessed: 05 September 2015.

[237] Novatec. Introduction to microservices testing and consumer driven contract testing with pact. https://blog.novatec-gmbh.de/introduction-microservices-testing-consumer-driven-contract-testing-pact/, may 2017. Accessed: 30 July 2018.

[238] M. Nygard. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

[239] OASIS. Amqp is the internet protocol for business messaging. https://www.amqp.org/about/what, 2018. Accessed: 15 December 2018.

[240] R. V. O'Connor, P. Elger, and P. M. Clarke. Continuous software engineering—a microservices architecture perspective. *Journal of Software: Evolution and Process*, 29(11):e1866. doi: 10.1002/smr.1866. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1866.

[241] United States Department of Defense. *RISK MANAGEMENT GUIDE FOR DOD ACQUISITION*. Unitest States Department of Defence, 6th edition, aug 2006.

[242] S. O'Grady. The difference between soa and microservices isn't size. https://redmonk.com/sograd y/2017/07/20/soa-microservices/, jul 2017. Accessed: 25 April 2018.

[243] M Oksa. Web api development and integration for microservice functionality in web applications. Master's thesis, University of Jyväskylä, 2016. URL http://urn.fi/URN:NBN:fi:jyu-201612215220.

[244] I Oksanych, I Shevchenko, I Shcherbak, and S. Shcherbak. Development of specialized services for predicting the business activity indicators based on micro-service architecture. *Information Technology*, 2017.

[245] G. Olliffe. Microservices : Building services with the guts on the outside. https://blogs.gartner.com/gary-olliffe/2015/01/30/microservices-guts-on-the-outside/, Jan 2015. Accessed: 18 March 2016.

[246] P. Oreizy. On the role of software architectures in runtime system reconfiguration. *IEE Proceedings - Software*, 145:137–145, October 1998. ISSN 1462-5970. URL http://digital-library.theiet.org/content/journals/10.1049/ip-sen_19982296.

[247] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, ICSE '98, pages 177–186, 1998. ISBN 0-8186-8368-6. URL http://dl.acm.org/citation.cfm?id=302163.302181.

[248] I. Ozkaya, R. Kazman, and M. Klein. Quality-attribute-based economic valuation of architectural patterns. Technical Report CMU/SEI-2007-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2007.

[249] C. Pahl and P. Jamshidi. Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2*, CLOSER 2016, pages 137–146, Portugal, 2016. SCITEPRESS - Science and Technology Publications, Lda. ISBN 978-989-758-182-3. doi: 10.5220/0005785501370146. URL https://doi.org/10.5220/0005785501370146.

[250] E. Papatheocharous, J. Andersson, and J. Axelsson. Ecosystems and open innovation for embedded systems: A systematic mapping study. In *Software Business*, pages 81–95, Cham, 2015. Springer International Publishing. ISBN 978-3-319-19593-3.

[251] M. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer. Service-oriented computing research roadmap. 2006.

[252] M. P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12, Dec 2003. doi: 10.1109/WISE.2003.1254461.

[253] M. P. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Commun. ACM*, 46(10):24–28, October 2003. ISSN 0001-0782. doi: 10.1145/944217.944233. URL http://doi.acm.org/10.1145/944217.944233.

[254] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. Microservices in practice, part 1: Reality check and service design. *IEEE Software*, 34(1):91–98, Jan 2017. ISSN 0740-7459. doi: 10.1109/MS.2017.24.

[255] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *J. Manage. Inf. Syst.*, 24(3):45–77, December 2007. ISSN 0742-1222. doi: 10.2753/MIS0742-1222240302. URL http://dx.doi.org/10.2753/MIS0742-1222240302.

[256] S. Penchikala. Susanne kaiser on microservices journey from a startup perspective. https://www.infoq.com/news/2017/07/kaiser-microservices-journey, jul 2017. Accessed: 18 August 2017.

[257] S. Penchikala. Managing data in microservices. https://www.infoq.com/news/2017/06/managing-data-in-microservices, jun 2017. Accessed: 15 April 2018.

[258] Jennifer Pérez, Nour Ali, Jose A. Carsı, Isidro Ramos, Bárbara Álvarez, Pedro Sanchez, and Juan A. Pastor. Integrating aspects in software architectures: Prisma applied to robotic tele-operated systems. *Inf. Softw. Technol.*, 50(9-10):969–990, August 2008. ISSN 0950-5849. doi: 10.1016/j.infsof.2007.08.007. URL http://dx.doi.org/10.1016/j.infsof.2007.08.007.

[259] Perish. Publish or perish. https://harzing.com/resources/publish-or-perish, 1999. Accessed: 30 June 2018.

[260] R. Perrey and M. Lycett. Service-oriented architecture. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119, Jan 2003. doi: 10.1109/SAINTW.2003.1210138.

[261] D. E. Perry, A. A. Porter, and L. G. Votta. Empirical studies of software engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 345–355, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336586. URL http://doi.acm.org/10.1145/336512.336586.

[262] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swindon, UK, 2008. BCS Learning & Development Ltd. URL http://dl.acm.org/citation.cfm?id=2227115.2227123.

[263] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1 – 18, 2015. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2015.03.007. URL http://www.sciencedirect.com/science/article/pii/S0950584915000646.

[264] T. Porrmann, R. Essmann, and A. W. Colombo. Development of an event-oriented, cloud-based scada system using a microservice architecture under the rami4.0 specification: Lessons learned. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 3441–3448, Oct 2017. doi: 10.1109/IECON.2017.8216583.

[265] C. Posta. The hardest part of microservices: Calling your services. http://blog.christianposta.com/microservices/the-hardest-part-of-microservices-calling-your-services/, apr 2017. Accessed: 23 July 2017.

[266] C. Posta. Low-risk monolith to microservice evolution part ii. http://blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution-part-ii/, oct 2017. Accessed: 14 December 2017.

[267] Christian Posta. Low-risk monolith to microservice evolution part i. http://blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution/, jun 2015. Accessed: 17 July 2015.

[268] K. Probst and J. Becker. Engineering trade-offs and the netflix api re-architecture. https://medium.com/netflix-techblog/engineering-trade-offs-and-the-netflix-api-re-architecture-64f122b277dd, aug 2016. Accessed: 19 September 2017.

[269] F. Rademacher, S. Sachweh, and A. Zündorf. Differences between model-driven development of service-oriented and microservice architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 38–45, April 2017. doi: 10.1109/ICSAW.2017.32.

[270] F. Rademacher, S. Sachweh, and A. Zündorf. Towards a uml profile for domain-driven design of microservice architectures. In *Software Engineering and Formal Methods*, pages 230–245, Cham, 2018. Springer International Publishing. ISBN 978-3-319-74781-1.

[271] S. Rajagopalan and H. Jamjoom. App–bisect: Autonomous healing for microservice-based apps. In *7th USENIX Workshop on Hot Topics in*

Cloud Computing (HotCloud 15), Santa Clara, CA, 2015. USENIX Association. URL https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/rajagopalan.

[272] A. Rajasekar, J. Coposky T. Russell and, A. de Torcy, H. Xu, M. Wan, R. W. Moore, W. Schroeder, S. Chen, M. Conway, and J. H. Ward. *The integrated Rule-Oriented Data System (iRODS 4.0) Microservice Workbook*. CreateSpace Independent Publishing Platform, first edition, 2015.

[273] E. Reinhold. Lessons learned on uber's journey into microservices. https://www.infoq.com/presentations/uber-darwin, Jul 2016. Accessed: 24 September 2016.

[274] M. Richards. *Microservices vs. Service-Oriented Architecture*. O Reilly Media, 2016.

[275] C. Richardson. A pattern language for microservices. http://microservices.io/patterns/index.html, 2014. Accessed: 15 January 2016.

[276] C. Richardson. *Microservice Patterns*. Manning Publications Company, 2018. ISBN 9781617294549. URL https://books.google.co.uk/books?id=UeK1swEACAAJ.

[277] C. Richardson and F. Smith. Microservices: From design to deployment, May 2016.

[278] C. Richardson, E. Wolff, R. Miles, S. Kaiser, S. Newman, and S. Tilkov. microxchg 2016 - c. richardson, e. wolff, r. miles, s. kaiser, s. newman, s. tilkov, feb 2016. URL https://www.youtube.com/watch?v=wHuI7C3-Eis&list=PLx2By31njbhrs8caX08BEusyD_fBDu-XG&feature=player_detailpage. Accessed: 23 March 2016.

[279] D. Richter, M. Konrad, K. Utecht, and A. Polze. Highly-available applications on unreliable infrastructure: Microservice architectures in practice. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 130–137, July 2017. doi: 10.1109/QRS-C.2017.28.

[280] S. Riegg Cellini and J. Edwin Kee. *Cost-Effectiveness and Cost-Benefit Analysis*, pages 636–672. JohnWileySons, 2015. ISBN 9781119171386. doi: 10.1002/9781119171386.ch24. URL http://dx.doi.org/10.1002/9781119171386.ch24.

[281] *Riemann - Concepts*. Riemann, 2017. http://riemann.io/concepts.html.

[282] Robert Riemenschneider and Mark Moriconi. Introduction to sadl 1.0 a language for specifying software architecture hierarchies. http://www.csl.sri.com/papers/sadl-intro/, 2019. Accessed: 15 April 2018.

[283] Ian Robinson. Consumer driven contracts. http://martinfowler.com/articles/consumerDrivenContracts.html, jun 2006. Accessed: 11 May 2016.

[284] R. Rodger. Measuring microservices. http://www.infoq.com/presentations/measuring-microservices, april 2015. Accessed: 28 February 2016.

[285] R. Rodger. *The Tao of Microservices*. Manning Publications Company, 2017. ISBN 9781617293146. URL https://books.google.co.uk/books?id=uosOkAEACAAJ.

[286] G. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 241–258, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336567. URL http://doi.acm.org/10.1145/336512.336567.

[287] H.D. Rombach. A controlled expeniment on the impact of software structure on maintainability. *Software Engineering, IEEE Transactions on*, SE-13(3): 344–354, March 1987. ISSN 0098-5589. doi: 10.1109/TSE.1987.233165.

[288] M. Rosen, B. Lublinsky, K.T. Smith, and M.J. Balcer. *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley, 2012. ISBN 9781118079799. URL https://books.google.co.uk/books?id=GFL9lWKojFYC.

[289] A. Rotem-Gal-Oz. *SOA Patterns*. Running Series. Manning, 2012. ISBN 9781933988269. URL https://books.google.co.uk/books?id=n3-5PgAACAAJ.

[290] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, Essex,England, dec 2009.

[291] K. Eati C. M. Ferreira D.n Glozic V. Gucer M. Gupta S. Joshi V. Lampkin M. Martins S. Narain R.n Vennam S. Daya, N. Van Duy. Microservices from theory to practice: Creating applications in ibm bluemix using the microservices approach. http://www.redbooks.ibm.com/abstracts/sg248275.html?Open, Aug 2016.

[292] L. Minku N. Ali S. Hassan, R. Bahsoon. Dynamic evaluation of microservice granularity adaptation. submitted to IEEE Transactions on Software Engineering (TSE), 2019.

[293] B. Coyne S. Shareema. Devops for dummies - 3rd ibm limited edition. https://www.ibm.com/ibm/devops/us/en/resources/dummiesbooks/.

[294] R.W. Saaty. The analytic hierarchy process—what it is and how it is used. *Mathematical Modelling*, 9(3):161 – 176, 1987. ISSN 0270-0255. doi: https://doi.org/10.1016/0270-0255(87)90473-8. URL http://www.sciencedirect.com/science/article/pii/0270025587904738.

[295] T.L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Analytic hierarchy process series. RWS, 1990. ISBN 9780962031724. URL https://books.google.co.uk/books?id=lfDXAAAAMAAJ.

[296] I. Sacolick. What is ci/cd? continuous integration and continuous delivery explained. https://www.infoworld.com/article/3271126/ci-cd/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html, may 2018. Accessed: 1 May 2017.

[297] Y. Sader. A microservices reference architecture. https://www.youtube.com/watch?v=KHqMPRA6jVI, 2015. Accessed: 11 June 2016.

[298] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 318–325, Dec 2016. doi: 10.1109/ICITST.2016.7856721.

[299] I. Salvadori, B. C. N. Oliveira, A. Huf, E. C. Inacio, and F. Siqueira. An ontology alignment framework for data-driven microservices. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '17, pages 425–433, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5299-4. doi: 10.1145/3151759.3151793. URL http://doi.acm.org/10.1145/3151759.3151793.

[300] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang. Context-aware adaptive applications: Fault patterns and their automated identification, September 2010. ISSN 0098-5589. URL http://dx.doi.org/10.1109/TSE.2010.35.

[301] Michele Sama, David S. Rosenblum, Zhimin Wang, and Sebastian Elbaum. Multi-layer faults in the architectures of mobile, context-aware adaptive applications: A position paper. In *Proceedings of the 1st International Workshop on Software Architectures and Mobility*, SAM '08, pages 47–49, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-022-7. doi: 10.1145/1370888.1370901. URL http://doi.acm.org/10.1145/1370888.1370901.

[302] A. R. Sampaio, H. Kadiyala, Bo Hu, J. Steinbacher, T. Erwin, N. Rosa, I. Beschastnikh, and J. Rubin. Supporting microservice evolution. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 539–543, 2017.

[303] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0. doi: 10.1145/1094811.1094824. URL http://doi.acm.org/10.1145/1094811.1094824.

[304] D. I. Savchenko, G. I. Radchenko, and O. Taipale. Microservices validation: Mjolnirr platform case study. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 235–240, May 2015. doi: 10.1109/MIPRO.2015.7160271.

[305] V. Sazawal and N. Sudan. Modeling software evolution with game theory. In *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*, ICSP '09, pages 354–365, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01679-0. doi: 10.1007/978-3-642-01680-6_32. URL http://dx.doi.org/10.1007/978-3-642-01680-6_32.

[306] R. Schaefer. Goto 2016 - from monolith to microservices at zalando - rodrigue schaefer. https://youtu.be/gEeHZwjwehs, aug 2016. Accessed: 24 September 2016.

[307] S. Schaevitz. Deploying changes to production in the age of the microservice. https://www.usenix.org/conference/srecon17europe/program/presentation/schaevitz, 2017.

[308] F. Schmidt, S. G. MacDonell, and A. M. Connor. *An Automatic Architecture Reconstruction and Refactoring Framework*, pages 95–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-23202-2. doi: 10.1007/978-3-642-23202-2_7. URL https://doi.org/10.1007/978-3-642-23202-2_7.

[309] Aston Business School. The aston centre for servitization research and practice. http://www.aston.ac.uk/aston-business-school/research/centres/aston-centre-for-servitization-research-and-practice/. Accessed: 16 May 2016.

[310] E. S. Schwartz. The valuation of warrants: Implementing a new approach. *Journal of Financial Economics*, 4(1):79 – 93, 1977. ISSN 0304-405X. doi: https://doi.org/10.1016/0304-405X(77)90037-X. URL http://www.sciencedirect.com/science/article/pii/0304405X7790037X.

[311] S. Sharma. *Mastering Microservices with Java 9: Build domain-driven microservice-based applications with Spring, Spring Cloud, and Angular*. Packt Publishing, 2017. ISBN 9781787282414. URL https://books.google.co.uk/books?id=WsxPDwAAQBAJ.

[312] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, April 1995. ISSN 0098-5589. doi: 10.1109/32.385970.

[313] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Trans. Softw. Eng.*, 21(4):314–335, apr 1995. ISSN 0098-5589. doi: 10.1109/32.385970. URL http://dx.doi.org/10.1109/32.385970.

[314] M. Sheppard. Design metrics: an empirical analysis. *Software Engineering Journal*, 5(1):3–10, Jan 1990. ISSN 0268-6961.

[315] F. S. Shoumik, M. I. M. M. Talukder, A. I. Jami, N. W. Protik, and M. M. Hoque. Scalable micro-service based approach to fhir server with golang and no-sql. In *2017 20th International Conference of Computer and Information Technology (ICCIT)*, pages 1–6, Dec 2017. doi: 10.1109/ICCITECHN.2017.8281846.

[316] R. Shoup. Goto 2016 - pragmatic microservices - randy shoup. https://youtu.be/9vS7TbgirgY, jul 2016. Accessed: 23 November 2016.

[317] D. Simons. Decoupled apis through microservices. https://www.infoq.com/presentations/api-microservices-tools, aug 2016. Accessed: 15 March 2017.

[318] N. Slack. Operations strategy: will it ever realize its potential? *Gestao & Producao*, 12:323 – 332, 12 2005. ISSN 0104-530X. URL http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-530X2005000300004&nrm=iso.

[319] T. Soenen, W. Tavernier, D. Colle, and M. Pickavet. Optimising microservice-based reliable nfv management orchestration architectures. In *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 1–7, Sept 2017. doi: 10.1109/RNDM.2017.8093034.

[320] J. Soldani, D. Andrew Tamburri, and W. V. D. Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215 – 232, 2018. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2018.09.082. URL http://www.sciencedirect.com/science/article/pii/S0164121218302139.

[321] Y. Song. Adaptation hiding modularity for self-adaptive systems. In *Companion to the Proceedings of the 29th International Conference on Software Engineering*, ICSE COMPANION '07, pages 87–88, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2892-9. doi: 10.1109/ICSECOMPANION.2007.15. URL http://dx.doi.org/10.1109/ICSECOMPANION.2007.15.

[322] J Sorgalla. Ajil: A graphical modeling language for the development of microservice architectures. In *Microservices*.

[323] Michael Stal. Chapter 3 - refactoring software architectures. In M. A. Babar, A. W. Brown, and I. Mistrik, editors, *Agile Software Architecture*, pages 63–82. Morgan Kaufmann, Boston, 2014. ISBN 978-0-12-407772-0. doi: https://doi.org/10.1016/B978-0-12-407772-0.00003-4. URL https://www.sciencedirect.com/science/article/pii/B9780124077720000034.

[324] G. Steinacker. On monoliths and microservices. https://dev.otto.de/2015/09/30/on-monoliths-and-microservices/, sep 2015. Accessed: 20 March 2016.

[325] J. Stenberg. Exploring the hexagonal architecture. https://www.infoq.com/news/2014/10/exploring-hexagonal-architecture, oct 2014. Accessed: 19 January 2016.

[326] J Stenberg. Microservices and teams at amazon. https://www.infoq.com/news/2015/12/microservices-amazon, dec 2015. Accessed: 05 June 2017.

[327] J. Stenberg. Strategies for decomposing a system into microservices. https://www.infoq.com/news/2018/06/decomposing-system-microservices, jun 2018. Accessed: 11 August 2018.

[328] Jan Stenberg. About the soa heritage impact on microservices. https://www.infoq.com/news/2017/11/soa-impact-microservices, nov 2017. Accessed: 03 March 2018.

[329] K. J. Sullivan, P. Chalasani, S. Jha, and V. Sazawal. Software design as an investment activity: A real options perspective. *Real options and business strategy: Applications to decision making*, (10):215 — 262, 1999.

[330] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. The structure and value of modularity in software design. In *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, pages 99–108, New York, NY, USA, 2001. ACM. ISBN 1-58113-390-1. doi: 10.1145/503209.503224. URL http://doi.acm.org/10.1145/503209.503224.

[331] D. Taibi and V. Lenarduzzi. On the definition of microservice bad smells. *IEEE Software*, 35(3):56–62, May/June 2018. ISSN 0740-7459. doi: 10.1109/MS.2018.2141031. URL doi.ieeecomputersociety.org/10.1109/MS.2018.2141031.

[332] B. Terzić, V. Dimitrieski, Slavica Kordić, A., G. Milosavljevic, and I. Luković. Microbuilder: A model-driven tool for the specification of rest microservice architectures. Mar 2017.

[333] ThoughtWorks. Otto | from legacy systems to fast and flexible platforms. https://youtu.be/bSvjZi3WKZQ, May 2016. Accessed: 24 June 2016.

[334] S. Tilkov. One size does not fit all. https://gotocon.com/dl/goto-london-2016/slides/Stefan_Tilkov-OneSizeDoesNotFitAllGOTOLondon16.pdf, Oct 2016. Accessed: 10 January 2017.

[335] Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Florian Dudouet, and Andrew Edmonds. An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, AIMC '15, pages 19–24, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3476-1. doi: 10.1145/2747470.2747474. URL http://doi.acm.org/10.1145/2747470.2747474.

[336] Sudhir Tonse. Scalable microservices at netflix. challenges and tools of the trade. http://www.infoq.com/presentations/netflix-ipc, mar 2015. Accessed: 18 April 2016.

[337] Google Trends. Microservices - explore. https://trends.google.com/trends/explore?date=2013-01-01 Accessed: 04 September 2017.

[338] L. Trigeorgis and P.F.L. Trigeorgis. *Real Options: Managerial Flexibility and Strategy in Resource Allocation*. Mit Press. PAPERBACKSHOP UK IMPORT, 1996. ISBN 9780262201025. URL https://books.google.com.sa/books?id=Z8o20TmBiLcC.

[339] J. Uhle. On dependability modeling in a deployed microservice architecture. Master's thesis, Universitat Potsdam, jun 2014.

[340] D Ulander. Software architectural metrics for the scania internet of things platform: From a microservice perspectiv. Master's thesis, Uppsala University, 2017. URL http://www.diva-portal.org/smash/record.jsf?pid=diva2:1115342.

[341] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003. ISSN 0926-8782. doi: 10.1023/A:1022883727209.

[342] R. van Engelen. Code generation techniques for developing light-weight xml web services for embedded devices. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 854–861, New York, NY, USA, 2004. ACM. ISBN 1-58113-812-1. doi: 10.1145/967900.968075.

[343] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 5–19, 2000. ISBN 1-58113-206-9. doi: 10.1145/337180.337184. URL http://doi.acm.org/10.1145/337180.337184.

[344] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 249–262, 2001. doi: 10.1109/ISRE.2001.948567.

[345] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10): 978–1005, Oct 2000. ISSN 0098-5589. doi: 10.1109/32.879820.

[346] Sandra Vandermerwe and Juan Rada. Servitization of business: Adding value by adding services. *European Management Journal*, 6(4):314 – 324, 1988. ISSN 0263-2373. doi: http://dx.doi.org/10.1016/0263-2373(88)90033-3. URL http://www.sciencedirect.com/science/article/pii/0263237388900333.

[347] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182, May 2016. doi: 10.1109/CCGrid.2016.37.

[348] S. Vlaovic, R. Pilani, S. Parulekar, and S. Handa. Netflix billing migration to aws. https://medium.com/netflix-techblog/netflix-billing-migration-to-aws-451fba085a4, Jan 2016. Accessed: 16 July 2016.

[349] Thomas Vogel and Holger Giese. Model-driven engineering of self-adaptive software with eurema. *ACM Trans. Auton. Adapt. Syst.*, 8(4):18:1–18:33, January 2014. ISSN 1556-4665. doi: 10.1145/2555612. URL http://doi.acm.org/10.1145/2555612.

[350] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 202–207, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0575-4. doi: 10.1145/1988008.1988037. URL http://doi.acm.org/10.1145/1988008.1988037.

[351] M. Vukovic and P. Robinson. Shop2 and tlplan for proactive service composition. In *UK-Russia Workshop on Proactive Computing*, feb 2005.

[352] H. Vural, M. Koyuncu, and S. Guney. A systematic literature review on microservices. In *Computational Science and Its Applications – ICCSA 2017*, pages 203–217, Cham, 2017. Springer International Publishing. ISBN 978-3-319-62407-5.

[353] T. Wagner. Microservices without the servers, Sep 2015. URL https://aws.amazon.com/blogs/compute/microservices-without-the-servers/.

[354] P. Wang, K. Chao, C. Lo, R. Farmer, and P. Kuo. A reputation-based service selection scheme. In *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, pages 501–506, Oct 2009. doi: 10.1109/ICEBE.2009.80.

[355] M. Wasson and B. Tam. Designing microservices: Continuous integration. https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd, Oct 2018. Accessed: 15 November 2018.

[356] C. Watson, S. Emmons, and B. Gregg. A microscope on microservices. https://medium.com/netflix-techblog/a-microscope-on-microservices-923b906103f4, feb 2015. Accessed: 20 March 2016.

[357] Weaveworks. Microservices demo: Sock shop, 2017. URL https://microservices-demo.github.io/. Accessed: 05 October 2016.

[358] D. Weyns, S. Malek, and J. Andersson. On decentralized self-adaptation: Lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 84–93, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-971-8. doi: 10.1145/1808984.1808994. URL http://doi.acm.org/10.1145/1808984.1808994.

[359] J. White, H. D. Strowd, and D. C. Schmidt. Creating self-healing service compositions with feature modeling and microrebooting. In *The International Journal of Business Process Integration and Management (IJBPIM), Special issue on Model-Driven Service-Oriented Architectures*, 2008.

[360] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requir. Eng.*, 11(1):102–107, December 2005. ISSN 0947-3602. doi: 10.1007/s00766-005-0021-6. URL http://dx.doi.org/10.1007/s00766-005-0021-6.

[361] A. Wiggins. *The Twelve-Factor App*, 2017.

[362] Philip Wizenty, Jonas Sorgalla, Florian Rademacher, and Sabine Sachweh. Magma: Build management-based generation of microservice infrastructures. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ECSA '17, pages 61–65, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5217-8. doi: 10.1145/3129790.3129821. URL http://doi.acm.org/10.1145/3129790.3129821.

[363] E. Wolff. *Microservices: Flexible Software Architecture*. Pearson Education, 2016. ISBN 9780134650401. URL https://books.google.co.uk/books?id=zucwDQAAQBAJ.

[364] T. Xie J. Sun C. Xu C. Ji W. Zhao X. Zhou, X. Peng. Benchmarking microservice systems for software engineering research. In *ICSE '18 Companion*, May 2018.

[365] L. Yu and S. Ramaswamy. Improving modularity by refactoring code clones: A feasibility study on linux. *SIGSOFT Softw. Eng. Notes*, 33(2):9:1–9:5, March 2008. ISSN 0163-5948. doi: 10.1145/1350802.1350816. URL http://doi.acm.org/10.1145/1350802.1350816.

[366] Y. Yu, H. Silveira, and M. Sundaram. A microservice based reference architecture model in the context of enterprise architecture. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 1856–1860, Oct 2016. doi: 10.1109/IMCEC.2016.7867539.

[367] J. A. Zachman. *The Zachman Framework For Enterprise Architecture: Primer for Enterprise Engineering and Manufacturing*. Zachman International, 2003.

[368] H. Zeiner, M. Goller, V. J. Expósito Jiménez, F. Salmhofer, and W. Haas. Secos: Web of things platform based on a microservices architecture and support of time-awareness. *e & i Elektrotechnik und Informationstechnik*, 133(3):158–162, Jun 2016. ISSN 1613-7620. doi: 10.1007/s00502-016-0404-z. URL https://doi.org/10.1007/s00502-016-0404-z.

[369] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 371–380, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134337. URL http://doi.acm.org/10.1145/1134285.1134337.

[370] T. Zheng, Y. Zhang, X. Zheng, M. Fu, and X. Liu. Bigvm: A multi-layer-microservice-based platform for deploying saas. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 45–50, Aug 2017. doi: 10.1109/CBD.2017.16.

[371] O. Zimmermann. Architectural decision identification in architectural patterns. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, WICSA/ECSA '12, pages 96–103, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1568-5. doi: 10.1145/2361999.2362021. URL http://doi.acm.org/10.1145/2361999.2362021.

[372] O. Zimmermann. Microservices tenets. *Computer Science - Research and Development*, 32(3):301–310, Jul 2017. ISSN 1865-2042. doi: 10.1007/s00450-016-0337-0. URL https://doi.org/10.1007/s00450-016-0337-0.

[373] O. Zimmermann. Architectural refactoring for the cloud: a decision-centric view on cloud migration. *Computing*, 99(2):129–145, Feb 2017. ISSN

1436-5057. doi: 10.1007/s00607-016-0520-y. URL https://doi.org/10.1007/s00607-016-0520-y.

[374] O. Zimmermann, V. Doubrovski, J. Grundler, and K. Hogg. Service-oriented architecture and business process choreography in an order management scenario: Rationale, concepts, lessons learned. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 301–312, New York, NY, USA, 2005. ACM. ISBN 1-59593-193-7. doi: 10.1145/1094855.1094965. URL http://doi.acm.org/10.1145/1094855.1094965.