A Concern-Oriented Software Engineering Methodology for Micro-Service Architectures

Maximilian Schiedermeier
max.schiedermeier@mcgill.ca
McGill University
Montréal, Canada

ABSTRACT

Component-Based Systems (CBS) allow for the construction of modular, highly scalable software. Decomposing a system into individually maintainable and deployable components enables a targeted replication of performance bottlenecks, and promotes code modularity. Over the last years, the Micro-Service Architecture (MSA) style has become a popular approach to maximize the benefits of CBS. However, MSA introduces new challenges, by imposing a conceptual and technological stack on adherent projects, which require new critical design choices. Throughout my PhD I want to investigate to which extent a systematic reuse of MSA solutions of various granularity can streamline MSA application development by guiding design decisions.

CCS CONCEPTS

• **Software and its engineering** → **Domain specific languages**; **Reusability**; *Interface definition languages*; *Model-driven software engineering*; **Abstraction, modeling and modularity**; • **Computer systems organization** → *Client-server architectures*; • **Networks** → Network performance modeling.

KEYWORDS

Micro-Service Architectures, Model-Driven Engineering, Concern-Oriented Reuse, Representational State Transfer

ACM Reference Format:
Maximilian Schiedermeier. 2020. A Concern-Oriented Software Engineering Methodology for Micro-Service Architectures. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3419488

1 INTRODUCTION

The modular character of component-based systems (CBS) leads to two advantages over monolithic software: the code is more coherent and loosely coupled, and performance bottlenecks can be replicated more easily. The Micro-Service Architecture (MSA) style, has positioned itself as a popular CBS representative that maximizes these advantages. The most outstanding difference to its predecessor, "*Service-Oriented Architecture*" (SOA), is finer grained services that expose their functionality exclusively over REST interfaces [16]. The latter eliminates the need for a protocol translating bus, which is considered a potential bottleneck. However, while finer grained services increase the advantages of modularity, they can also constitute a trade-off against performance [9] [13], because more services also lead to increased traffic.

Although MSA promotes desirable characteristics, the style also introduces new challenges. Engineers are confronted with a complex stack of additional technological and conceptual choices that require thorough design decisions. Still, many of these choices showcase a recurring character and have standard solutions. Potential alternatives can be accurately captured by feature models, to promote techniques from Model-Driven Engineering (MDE). However, the combinatoric outcome can be overwhelming to the user. Furthermore, the consequences of different choices are often hard to predict, which is why engineers seek tools to assists MDE's implicit decision-making, and enable a more streamlined exploration process, with direct feedback on the impact of different choices.

The development of such a tool is challenging, because MSA related solutions cannot always readily be encapsulated as a service. Solutions often cross-cut the internals of different services. Therefore, a purely component-oriented reuse, where off-the-shelf solutions are selected from a reusable component library, is too course grained to be unrestrictedly applicable to an MSA context. However, a concern-oriented approach tailors the granularity of reuse, as it considers solutions that cross-cut services.

Throughout my PhD I seek to investigate to which extent concern-oriented reuse can promote the vision of streamlined design decision making for MSA applications. This goal is challenging in two aspects: The MSA style is still relatively young and keeps evolving. Therefore identified solutions must regularly be put into question, and uprising trends must be followed. Secondly, many MSA related solutions are too context-dependent to allow an off-the-shelf integration. Additional models, that accurately describe further customization are required, prior to an effective integration of adapted solutions into an applicative base context. Streamlined MSA design will thus require additional domain-specific languages to express reusable solutions at a meaningful level of abstraction.

2 RELATED WORK

The general idea of leveraging MSA design with a streamlined decision and configuration process builds on top of several building blocks, which are presented in this section.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3419488

The following list compiles articles that either deal with, or indirectly support an identification of essential MSA challenges and solution strategies:

- In [10], *Zündorf et al.*, identify and capture common technological MSA choices, based on an *Internet of Things* (IoT) case study. The authors acknowledge the benefits of a streamlined guiding through meaningful design choices. However, it is unclear to which extent their work is applicable to our context. The IoT domain renders the case study unrepresentative for an MSA context. Furthermore their discussion is limited to technological variability. Many reusable MSA concepts are not bound to a specific technology.
- In [7], *Kakivaya et al.* describe *Service Fabric*, Microsoft's Cloud Platform for the development of Micro-Service applications. Although the paper is specifically scoped to a single product, the authors accurately describe common MSA challenges and solutions. This provides us with valuable insights on common MSA reuse candidates.
- In[1], *Dragoni et al.* discuss the backgrounds that lead to the emergence of MSA. The authors discuss recurring style concepts and argue how those address common MSA challenges. The study also covers a section of challenges that are not yet satisfyingly answered by MSA. Unfortunately the article does not consider models that would allow to express the observed variability, for a later systematic reuse.
- In [9], *Rademacher et al.* investigate how the evolution of SOA to MSA affected model-driven engineering. The article contains a compilation of typical conceptual MSA best practices and solutions. The authors put these observation in relation to the predecessor SOA. They also share our understanding that the matter of reuse should not be bound to a specific level of abstraction or granularity. Yet the study provides no insight on how variability could be accurately captured through models.
- In [4], *Halin et al.* present a novel product line approach based on the open-source software JHipster [11]. While JHipster allows sampling web-related configurations and code, the software did at the moment of writing not yet target an MSA context. The authors present a feature model that organizes JHipster's reusable concepts and provides valuable insights on how to express variability. However, the feature model barely covers concepts for non-monolithic architectures, such as different load-balancing strategies.
- A de-facto standard for MSA is inter-service communication through REST interfaces. In [3], *Roy Fielding*, the author of the REST style, summarizes fundamental style paradigms and clarifies common misunderstandings. This specification is highly relevant for the crafting of models that express service interfaces at a meaningful level of abstraction.

There currently is no concern-oriented approach for the streamlined development of MSA applications. However the venture can benefit of concern-oriented best practices, gained on monolithic systems.

- In [15], *Schöttle et al.* present *TouchCORE*. This software exposes the *Concern-Oriented Reuse* (CORE) paradigm though a multi-touch enabled user interface. TouchCORE is a reasonable starting point for our venture, because its included *Concern-Oriented Reuse* implementation targets the systematic modularization and packaging of tested and proven design solutions. Those concerns are made available in a reusable *concern library* to support the streamlined selection and integration into a base applications. However, TouchCORE was developed for monolithic applications and therefore neither includes MSA-related concerns, nor does its integrated code generator support non-monolithic software.
- Concern-oriented reuse highly relies on integrating selected models into a application base context model. This process is called *weaving*. In [5], *Horcas et al.* present a concept to express variability, user selections and weaving strategies. Although their approach is related to our envisioned methodology, the authors do not discuss whether their concept is applicable for non-monolithic software.
- A potential starting point to our venture is [8], where *Kienzle et al.* discuss the general possibility of combining concern-oriented reuse with component-oriented software engineering. Since MSA applications are component-based systems, and some solutions can be modelled at component granularity, the discussion is highly relevant for our purpose. However, the study is limited to a general discussion of potential benefits and expected challenges.

An essential advantage of a concern-oriented MSA design process is the possibility to receive direct feedback on the impact of specific design choices on software qualities. Given the MSA context, we will prioritize feedback on expected *performance scaling* over other relevant software qualities, such as *resource efficiency*, *maintainability* and *reliability*. We want to benefit from existing simulation based predictions that operate on component-based models. We believe that the required input models can be generated from user-provided high level design choices.

- In [12], *Reussner et al.* describe *Palladio*, a performance simulation tool for component-based systems. Based on input models that define components, deployment on hardware, usage profiles, and functional dependencies, Palladio runs a response time simulation for queries entering the system. This provides insight on a deployment configuration's overall scaling behaviour. A potential downside of the approach is that the simulations can be resource intensive. This could hinder timely user feedback throughout the exploration of configurations.
- In [2], *Eismann et al.* discuss the advantages of hybrid performance simulations. Those are a combination of component-based simulations and statistical estimations. The authors argue that this approach can provide faster simulations, while preserving general applicability. We could imagine to accelerate predictions using such a hybrid mechanism if responsiveness is an issue. It is however currently unclear to which extent a hybrid approach is compatible with concern-oriented reuse. Hybrid approaches require statistical benchmark data at the component level. In a concern-oriented methodology such data does not necessarily exist.

3 PROPOSED SOLUTION: A CONCERN-ORIENTED APPROACH TO MSA DEVELOPMENT

We suggest to streamline design decisions based on a concern-oriented reuse approach. This means that for any recurring MSA problem, the user is guided through her different solution alternatives and provided with minimal but valuable information on the impact of different choices. This enables fast but informed decision-making. Thanks to CORE [15], the most suitable solution can be integrated into the application base. Novel to this approach is that the offered solutions are decoupled from service-level granularity. Compared to component-oriented reuse, where every solution has to be modularized and encapsulated as a specific service, concern-oriented reuse targets solutions that cross-cut components or affect their interplay.

The key parts to our proposal relate the existing approaches as follows:

- To enable reuse, the user must have access to a meaningful compilation of articles on MSA challenges and the corresponding solutions. We suggest to group technical solutions, mentioned in MSA specific contritions by the problem they address. Specifically we want to consider the overlaps of the following studies, to compile an initial MSA specific concern library: [7], [9], [10], [6]
- In existing concern-oriented frameworks, user selected solutions require further customization before they can be integrated into a base context. It is expectable that this requirement also holds for the MSA context. We suggest to analyse individual concerns of our initial library for meaningful customization abstractions. We consider the integration of concern-provided Domain-Specific-Languages (DSLs) for a more concern-tailored configuration process.
- As we propose an informed decision making, the user has to be provided with insightful feedback on the impact of her choices. We suggest to focus initially on performance estimations, as those are the primary motivation for MSA in the first place. We propose to gain reliable performance estimation through existing component-based simulation software [2] ,[12]. Model transformations can be used to generate the required simulation input data from the user's design choices.
- We suggest an approach based on model weaving for the integration of selected solutions into an application base context. TouchCORE [15] already supports model-weaving for monolithic applications. We suggest to translate these algorithms to newly introduced MSA-specific models. If the current weaving patterns are not applicable, we want to rely on more elaborate patterns as described in [5].

4 PLANNED EVALUATION

We consider the following criteria to evaluate whether the proposed methodology is beneficial to a more streamlined MSA application design:

- *Concern coverage:* A critical factor is whether the provided concerns are relevant and sufficient for an effective MSA design. This can be validated empirically, by realizing a representative case study, entirely relying on the provided concern library. We suggest to perform this validation based on a common MSA test application, the *Teastore* [16].
- *Correctness:* We intend to support model-based code generation. Therefore, a correct functioning of the produced application code can be easily verified by unit testing the modeled service interfaces.
- *Usability:* By confronting inexperienced users with a tool that implements our proposed methodology, we can evaluate how well requested MSA strategies can be effectively integrated in a given amount of time.
- *Feedback accuracy:* The purposefulness of predicting scaling behaviour based on generated simulation input data can be validated by comparing the performance predictions to benchmarks, obtained from the generated application.

5 EXPECTED CONTRIBUTIONS

A key factor to our research, is a successive elicitation of representative MSA concerns and solutions into a concern library. As our first contribution, we will elaborate a collection of reusable solutions to recurring MSA challenges. This can be done based on their occurrence in related work and representative case studies. A criteria for the structuring of this library will be a classification into functional and non-functional concerns, as well as an arrangement by the scope of the contained solutions: service cross-cutting, targeting the interplay of services, or specifically at single-service granularity.

TouchCORE supports the customization of concern solutions. Yet, so far there has not been a need for additional domain-specific languages (DSLs) to further guide a solution's customization. From our initial experiments with REST technology it seems, that MSA-related concerns often showcase the need for an extra level of abstraction, throughout the customization process. We want to demonstrate the usefulness of a concern-provided DSL for the purpose of further customization. We could also imagine to target a secondary contribution that concludes which general concern properties legitimate the integration of a concern-internal DSL. An aspect of this study would be to investigate why MSA concerns often require this extra level of abstraction.

Weaving is the process of composing models based on mappings. It is a key concept for the integration of solutions from our concern library into an application context. However, weaving requires homogeneous input models types. In the presence of concern-specific DSLs, weaving is therefore not directly applicable. We believe that this gap can be bridged with additional model transformations. We want to perform a case study on an illustrative MSA concern, to demonstrate the feasibility of model weaving, despite initial model incompatibility due to a concern's customization with a custom DSL. As our concern library grows we would also like to re-validate if the described technique is applicable to all DSL-charged concerns in our library.

We expect another contribution resulting from a support of performance predictions. Since we want to gain insight on expected

scaling behaviour through component-based simulations, we must at some point generate the required input models. To some extent these inputs change, in correlation to the reuses selected for integration. Selecting a server-side load-balancer must, for instance, lead to a different prediction than choosing a client-side load balancer. Yet we believe that it is possible to synthesize the required simulation inputs based on the user-provided base context in combination with the user's reuse selection. Such model conversions would not only enable performance predictions, they would notably contribute to bridging the semantic gap between concern and component-oriented models.

6 CURRENT STATUS

In this last section, I summarize the results we have obtained since the PhD-start in January 2019. Precisely, I present a list of concerns that we extracted from related work, and that we consider candidates for a MSA-specific concern library. Afterwards, I summarize insights that we have gained while working on a proof of concept of our methodology on a first representative concern. The article concludes with a short outline on the upcoming steps of our research.

6.1 Concern Library Candidates

Based on related work, we identified the following four MSA challenges as reasonable candidates for an initial concern library:

- *Restification*, as the process of exposing functionality through a REST interface. After choosing the desired framework technology, this concern will allow a user to map from a custom REST interface on existing method signatures. The target functionality is then wrapped up as a REST service, and can from there on be included in a Micro-Service application. The granularity of this functional concern is *component cross-cutting*, for the concern targets service internals.
- *Load balancing* deals with strategies that optimize the performance by routing requests alternately to distinct service representatives. The user can chose between different load-balancing strategies, such as *client-sided* or *proxy load balancing*. Based on the available resources and the selected strategy, she is then provided with a performance estimation, produced by an internal, component-based simulation. The scope of this *non-functional* concern lies at inter-service level, because it affects the interplay of services.
- The *Consistency* concern gathers strategies that ensure a consistent service state for consecutive queries. Consistency is relevant whenever multiple instances of a stateful service are deployed. There are various strategies to ensure consistency , such as *outsourcing state to a database*, *synchronization over asynchronous side channels*, and *master-slave replication*. All options have different impacts on high level goals, notably performance. Depending on the selected reuse, the concern will target all levels of reuse granularity.
- *Access Control* deals with concepts that restrict specific service functionality to certain users or user groups. There are again various strategies to ensure access control, ranging from *cryptographic session tokens issued by single-sign-ons*, to *static client certificates* and repeated *credential-based authentication*. An access control concern offers the most common solutions and provides feedback on the expected impacts on security and performance. The concern operates at inter-component granularity.

6.2 Methodology, Proof of Concept

Among the concerns listed above, we consider a realization of the *restification* concern a straightforward starting point for a general validation of the proposed methodology. The exposure of internal functionality through a REST interface is a particularity common to all Micro-Services. Furthermore, this concern exemplifies the challenge of customizing a concern at a meaningful level of abstraction. We want to guide the user toward a pertinent interface design, while concealing technical implementation details.

We realized this abstraction with a novel DSL that describes a REST-Interface's (RIF) layout in form a tree structure and additionally indicates which CRUD methods are enabled per resource. Furthermore, we integrated an intuitive RIF editor into TouchCORE that allows a developer to efficiently create such models. A capture of this editor in action is displayed in figure 1. RIF models differ from existing REST-specifications, because they are designed to exclude some API information, specifically all information that can be derived from a mapping to existing method signatures. This decision was made, because the purpose of RIF models is the re-exposure of existing functionality, not the description of an entire API.

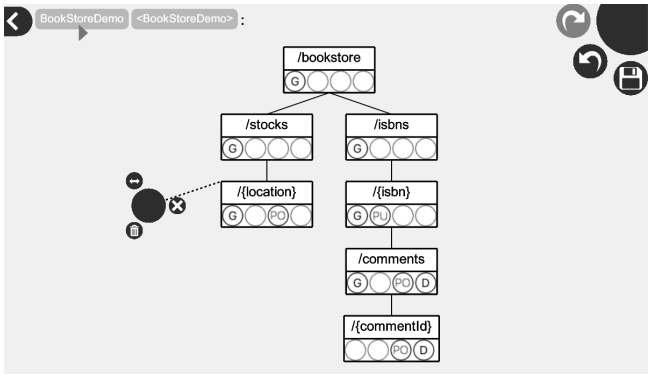

Figure 1: TouchCORE's new RIF editor in action. RIF models enable a re-exposure of existing functionality through a REST interface. The displayed model originates the Bookstore case-study [14]. Nodes represent REST-resources, while round buttons stand for access methods. The abbreviations Get, Put, Post, Delete indicate whether a method has been activated.

Our concept of weaving RIF models into a base application relies on model transformations. Although this part has not yet been integrated into TouchCORE, we confirmed the feasiability of translating RIF models to functionality-poor class diagrams. The latter are class diagrams that reflect all information contained in a RIF model through dedicated annotations. Still, these class diagrams do

not yet contain any logic from the base application. This transformation is an intermediary step that enables weaving. Based on a user provided mapping, the generated class diagrams can be woven with the class diagrams of a base application. We were able to manually apply this approach to multiple case studies.

Although at the moment of writing, the restification concern still requires these manual model operations, we were able to confirm a general viability of our approach. We used our RIF-editor to design an interface for a case study application [14], and manually combined the models into a woven, restified service description. Using TouchCORE's integreated code generator we were then able to generate and operate the corresponding REST-service. All unit tests, that covered the complete REST interface of our generated service, passed without errors.

6.3 Outlook

The promising results obtained with restfication leave us optimistic that there is no general incompatibility between concern-oriented reuse and MSA. Notably, it seems that concern-specific DSLs are an elegant way to guide the concern customization process at a meaningful level of abstraction.

In the near future we plan to automatize the weaving-related steps that were still performed manually during our previous restification experiments. Once this functionality has been integrated into TouchCORE, nothing speaks against an embedded, dedicated "restification" concern, that can then be tested on more complex case studies.

In the long run, we want to achieve the same for concerns that target the interplay of services. Notably the *Load Balancing* concern seems to be an appropriate candidate for this venture, also because it inherently motivates the integration of performance predictions. In order to achieve this, the most important stages will be:

- Enable TouchCORE support for non monolithic base-contexts. This is an essential precondition to any modeling of service interplays.
- Support for the conversion of modular TouchCORE projects to existing component models. This will enable the integration of performance predictions. Precisely we target the *Palladio Component Model* (PCM), which can be used as input for Palladio's performance simulator [12].
- Critical re-validation of the concern library as it grows. We want to assess the relevance of included concerns and solutions, by modelling a generally accepted reference MSA application, the *Teastore* [16]. This step might also reveal other essential MSA concerns, that are yet missing in our preliminary concern library.
- A user study, where inexperienced users are confronted with specific concerns and asked to apply them to their own base-applications. The outcome of this study will be a honest feedback on the usability of our approach, as well as a confrontation with contexts outside of our control or consideration.

The above list indicates that many open challenges remain to be adressed, before MSA development can practically benefit from concern-oriented reuse. Still we are looking forward to the upcoming experiments and insights, and remain confident that our work will advance the state of the art toward a streamlined reuse-oriented engineering methodology for the design on MSA applications.

REFERENCES

[1] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*. Springer, 195–216.

[2] Simon Eismann, Johannes Grohmann, Jürgen Walter, Jóakim Von Kistowski, and Samuel Kounev. 2019. Integrating Statistical Response Time Models in Architectural Performance Models. In *2019 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 71–80.

[3] Roy T Fielding, Richard N Taylor, Justin R Erenkrantz, Michael M Gorlick, Jim Whitehead, Rohit Khare, and Peyman Oreizy. 2017. Reflections on the REST architectural style and principled design of the modern web architecture (impact paper award). In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 4–14.

[4] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Patrick Heymans. 2017. Yo variability! JHipster: a playground for web-apps analyses. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*. 44–51.

[5] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2016. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software* 112 (2016), 78–95.

[6] David Jaramillo, Duy V Nguyen, and Robert Smart. 2016. Leveraging microservices architecture by using Docker technology. In *SoutheastCon 2016*. IEEE, 1–5.

[7] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, et al. 2018. Service fabric: a distributed platform for building microservices in the cloud. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 33.

[8] Jörg Kienzle, Anne Koziolek, Axel Busch, and Ralf H Reussner. 2016. Towards Concern-Oriented Design of Component-Based Systems.. In *ModComp@ MoDELS*. 31–36.

[9] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2017. Differences between model-driven development of service-oriented and microservice architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 38–45.

[10] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2019. Aspect-Oriented Modeling of Technology Heterogeneity in Microservice Architecture. In *2019 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 21–30.

[11] Matt Raible. 2016. *The JHipster mini-book*. Lulu. com.

[12] Ralf Reussner, Steffen Becker, Jens Happe, Anne Koziolek, Heiko Koziolek, Klaus Krogmann, Max Kramer, and Robert Heinrich. 2016. *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press.

[13] Nick Rozanski. 2011. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition*. Addison-Wesley Professional.

[14] Maximilian Schiedermeier. 2020. Book-Store Internals Sources. https://github.com/kartoffelquadrat/BookStoreInternals/.

[15] Matthias Schöttle, Nishanth Thimmegowda, Omar Alam, Jörg Kienzle, and Gunter Mussbacher. 2015. Feature modelling and traceability for concern-driven software development with TouchCORE. In *Companion Proceedings of the 14th International Conference on Modularity*. 11–14.

[16] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 223–236.