Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices

Holger Knoche*
Software Engineering Group
Kiel University
24118 Kiel, Germany
hkn@informatik.uni-kiel.de

ABSTRACT

Microservices are a promising target architecture for the modernization of monolithic software. However, breaking up a monolith into services can have a severe impact on performance, especially transactions. Therefore, careful planning of such modernizations with respect to performance is required. This is particularly true for incremental modernizations, which release partially modernized states of the application into production. In this paper, we present a simulation-based approach for sustaining runtime performance during incremental modernizations towards Microservices.

1. INTRODUCTION

Microservices [8] have recently emerged as a promising architectural style for enterprise software. Unlike traditional service-oriented approaches, which aim at business-centric and coarsely-grained services, Microservices explicitly focus the internal structure of an application. This focus makes them a promising option for the modernization of monolithic software systems by splitting them into a set of interacting services [11].

Most monolithic software systems are too large to modernize them in a single step. As a consequence, such modernizations are carried out gradually in several steps, or increments, along a modernization path [17].[1] Since each increment is released into production, it is imperative that every single one meets the requirements of productive use, of which performance constraints are of particular importance. To ensure continuous production-readiness throughout the modernization, these constraints have to be considered already during the design of the modernization path.

Migrating an existing software system to services may affect runtime performance in several ways. During the migration, immediate accesses, such as native function calls or database joins, have to be substituted by service invocations. Such substitutions are known to potentially reduce performance, e.g. due to serialization and network communication [9]. Another major issue are transactions. The introduction of service invocations to existing transaction contexts increases transaction and thus lock duration, potentially leading to a reduction in transaction throughput.

There are several measures that can be taken to mitigate the performance effects of service invocations. Less rigorous transaction concepts such as explicit compensation and Try-Cancel-Confirm (TCC) [13] may help avoiding performance degradation due to transactions, and patterns like the ones presented in [4] can be applied to increase invocation performance.

Applying these measures may require significant changes on the client side, i.e. changes to the monolith. As such changes are non-trivial, they also have to be planned in advance as part of the modernization path. According to our experience, however, performance problems are tackled as they appear. Since this usually happens in late testing stages or even in production, fixes are implemented under high pressure, often jeopardizing the modernization schedule. Sometimes, modernization steps even have to be undone or postponed, because no quick fix can be devised.

In this paper, we present an approach to improve the planning of incremental modernizations by predicting the performance impact of modernization paths on use cases. The remainder of this paper is structured as follows. In Section 2, related work to our approach is discussed. The approach itself is described in detail in Section 3, and the plan for our intended research is presented in Section 4.

2. RELATED WORK

In the following section, existing work related to our proposed research is presented and shortcomings with respect to our approach are discussed.

The migration of existing software to services has been extensively researched, and a large body of literature is available on the subject. Although much of this work is concerned with traditional services, large parts can also be applied to Microservices, since both rely on similar technolo-

*Holger Knoche is also affiliated with b+m Informatik AG, Rotenhofer Weg 20, 24109 Melsdorf, Germany.

[1]Seacord et al. use the term *modernization strategy*; however, we prefer the term modernization path to denote plans for concrete measures as strategies commonly refer to more abstract patterns.

ICPE'16 March 12-18, 2016, Delft, Netherlands

ACM ISBN 978-1-4503-4080-9/16/03.

DOI: http://dx.doi.org/10.1145/2851553.2892039

gies. Surveys of the existing research can be found, for instance, in [1] and [6]. In [15], a generic conceptual framework for such migrations, called SOA-MF, is presented. Based on this conceptual framework, several existing approaches are categorized into eight so-called migration families in [14]. A major shortcoming of these existing approaches is described in [16]. The authors conclude from expert interviews that migrations in industrial practice employ a similar approach, which, however, is fundamentally different from existing research approaches. While research approaches aim at fully comprehending and transforming the as-is state of the application, practical approaches focus the to-be state of the application and tend to design from scratch based on expert knowledge. Research challenges resulting from these findings are presented in [5]. Specific challenges in migrating to Microservices are discussed in [11].

Performance properties of migrations to service-oriented systems have been investigated by several authors. Performance pitfalls when migrating to Web Services are discussed in [9]. Furthermore, the author presents a performance model based on Layered Queueing Networks. Quality-of-Service issues of the migration are also discussed in [12]. A tool for performance modeling of service-oriented systems is presented in [3]. Furthermore, architectural performance models like Palladio [2] can be used for simulating performance properties of service-oriented systems. However, these approaches are created for analyzing a system at a specific point in time, and do not provide means for structural model evolution. Furthermore, the approaches do not allow to simulate the effects of transactional behavior.

3. APPROACH

In order to improve increment planning with respect to runtime performance and to address the shortcomings discussed above, we propose the following approach. Our approach assumes that the following pattern is used to separate a service from the monolith, which is based our experience from industrial projects and similar to the Branch-by-Abstraction pattern.[2] First, an existing service provided by the monolith is identified, e.g., from expert knowledge, and its to-be signature is specified. Then, implementation elements (e.g., modules, methods, or database tables) currently providing this service are localized inside the monolith, and a service implementation is provided. Note that this may be a temporary implementation wrapping or adapting the existing monolith's functionality. Once the implementation is available, accesses to the existing implementation are changed to use the new service. According to our experience, there is often a significant mismatch of the as-is and to-be signature, even to the extent that multiple services have to be invoked to substitute an existing access site. If a temporary implementation was provided earlier, it may be replaced during the substitution process. As soon as the substitution is completed and the final implementation is available, the original implementation can be retired.

Our approach targets development teams who wish to incrementally migrate a transactional, monolithic software application towards a Microservice architecture. It aims to guide and support increment planning by predicting the performance impact of the future increments. As the prediction is performed during the design phase, the findings can be anticipated in the current increment. We propose an iterative approach, where each iteration consists of the eight steps described below. The process is also depicted in Figure 1.

In Step **S1**, the expected workload and performance constraints are specified based on usage scenarios. Usage scenarios are instantiations of use cases that are immediately executable using the existing application. For each usage scenario, performance constraints (response times, transaction duration, throughput) and a workload intensity distribution are defined.

Step **S2** consists of a hybrid analysis of the current evolution state of the monolith. Using dynamic analysis techniques, the usage scenarios are profiled to investigate which parts of the application take part in each scenario, and to determine current timing information. Furthermore, transaction boundaries and deployment information are recorded during the profiling. The results of the dynamic analysis can optionally be complemented with a static analysis to improve coverage. The structural information gathered by this step is stored in an Application Structure Model (ASM).

In Step **S3**, the planned modernization path alternatives are specified, where each path consists of several increment models. Each such model describes the planned changes to the application in the corresponding implementation increment. According to our experience, it is impractical to plan the entire modernization in advance, as such projects commonly take several years. Therefore, we propose that only the next few increments are modeled. We intend to use a domain-specific language (DSL) for this task, which provides a set of operations on ASMs. Currently, we envision the following operations: (i) introduction of a new service, which includes an effect specification of its implementation, (ii) substitution of an access site by one or more service invocations, (iii) replacement of a service implementation, (iv) change of deployment of a service, (v) changes of the deployment infrastructure (e.g., addition of servers), and (vi) retirement of a service.

Steps **S4** to **S6** are performed separately for each modernization path. We aim at fully automating these steps. In Step **S4**, a sequence of future ASMs is generated by applying the specified increments to the current ASM. Subsequently, a corresponding performance model is derived in Step **S5** for each of these ASMs. In Step **S6**, the increments are also applied to the traces obtained in Step **S2**. Then, the modified traces and workload distributions are used to simulate the expected performance using the performance models generated by the previous step.

Once the simulation is complete for all modernization paths, the results are checked against the defined performance constraints. In Step **S7**, the simulation results as well as the check results are presented to the developer. Based on these results, he or she may then choose to select a path for implementation, or to return to Step **S3** to improve the paths if the expected performance is deemed insufficient.

If a path has been selected for implementation, the first increment of this path is implemented in Step **S8**. We explicitly refrain from "frozen zones" during the modernization. Therefore, the changes made due to the modernization are also merged with the independent changes in the course of this step.

With the completion of Step **S8**, the iteration ends. Unless the migration is considered complete, a new iteration is started. In subsequent iterations, the existing models are

[2]See martinfowler.com/bliki/BranchByAbstraction.

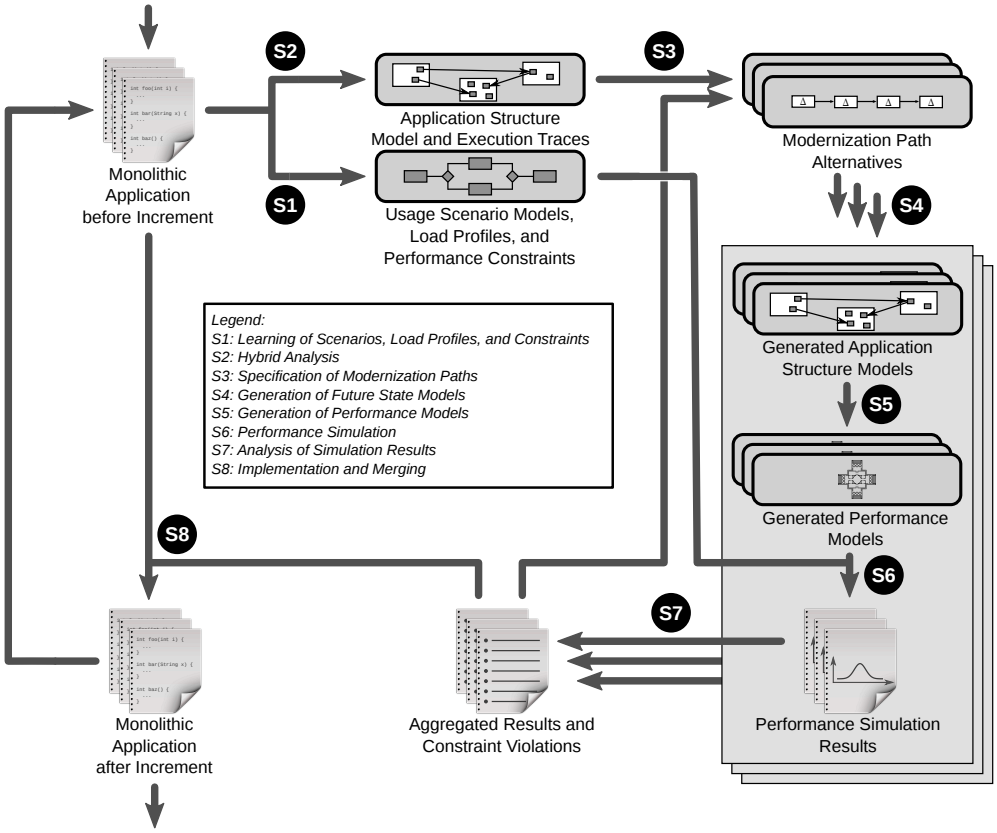

Figure 1: Overview of the proposed approach

updated to appropriately reflect the current state of the application. The first increment of the modernization path that was taken is removed, and, if applicable, a new increment is appended. The paths that were not taken are removed entirely.

4. RESEARCH PLAN

Our planned research is structured into five work packages, which are described below. For every work package, the associated research questions, intended research methods, and expected results are presented, where applicable.

The first work package, **WP1**, is almost completed and was concerned with initial research, topic development, and research planning. In addition, we were able to acquire industrial case studies for our intended research, and to initiate research collaborations for later work packages. One of these collaborations already led to a joint publication on extending the Palladio Component Model to support transactional behavior [10]. Work on this package has begun in Q4 2014, and its completion is scheduled for Q4 2015.

Work package **WP2** aims at further confirmation of the assumptions of our approach. The guiding research questions of this work package are: *What challenges exist in the transition to Microservices with respect to performance? What are common source and target architectures? What are preferred modes of transition?* To address these questions, we plan to carry out further literature research. In addition, we intend to conduct expert interviews with architects of monolithic applications to gather knowledge about common source and target architectures as well as transition modes and migration architectures. For the identification of performance challenges, we furthermore plan to perform exemplary migrations of benchmark applications in laboratory experiments. Work on this package is scheduled from Q4 2015 to Q2 2016.

Work package **WP3** is concerned with the development of the required metamodels, domain-specific languages, and tools to create and edit them. Of particular interest are the models for the specification of the modernization paths. The guiding research questions of this work package are: *Which elements and operations are required for modeling modernization paths? How can the models be efficiently created and edited? How can the models be kept in sync with the actual implementation?* To answer these questions, we intend to perform laboratory experiments with benchmark applications and our industrial case studies. To evaluate the usability of our tool set, we plan to have developers from our case studies use our tools and answer a questionnaire afterwards. For the dynamic analysis, we intend to employ Kieker [18], which also provides support for legacy languages such as COBOL [7]. Work on this work package is scheduled from Q1 2016 to Q3 2016.

Work package **WP4** is concerned with the generation of state and performance models from modernization paths, and the performance simulation thereof. The guiding research questions of this work package are: *How can ASMs and performance models be generated from modernization*

path models? How can the existing execution traces be transformed? How can transactional behavior be appropriately simulated? How can patterns common to Microservice architectures (e.g., circuit breakers) be simulated? How precise are the simulation results? To answer these questions, we intend to perform laboratory experiments with benchmark application and our case studies. In order to simulate transactional behavior, we are working to extend the Palladio Component Model accordingly, as described in [10]. Work on this package is scheduled from Q2 2016 to Q4 2016.

In work package **WP5**, we plan to focus on the analysis and presentation of the simulation results, and the implementation of appropriate tools. The guiding research questions are: *How can the results be aggregated and presented so that the developer can choose an appropriate path? Which information is required to quickly locate causes for performance constraint violations?* To answer these questions, we intend to conduct experiments with developers, who are asked to analyze results from an analysis containing known violations. After the analysis is complete, it is evaluated whether the probands were able to correctly locate the cause of the violations, how much time they took to locate them, and how usable they found our tool set. Work on this package is scheduled from Q4 2016 to Q2 2017.

Work package **WP6** is concerned with the overall evaluation of our approach in industrial migration projects. We intend to evaluate the approach using three industrial case studies, a medium-sized Java EE application, a large Java EE application, and a large COBOL application. The guiding research questions are: *Is the approach and tool set scalable enough to handle industrial projects? Is it flexible and adaptable enough to handle migrations of software in different programming languages and environments? To what extent can the migration process be guided by our strategies and tools?* To answer these questions, we plan to create models of our case studies to investigate the scalability and flexibility of the tool set. In addition, we intend to migrate selected parts of the case studies using our approach to check the overall applicablility. Work on this package is scheduled from Q2 2017 to Q4 2017. In the course of the evaluation, we furthermore intend to investigate to which extent cultural aspects associated with Microservices, such as DevOps, are adopted in industrial practice.

The work packages can be mapped to the steps of the approach as follows. **WP3** addresses Steps **S1** to **S3** and **S8**, **WP4** addresses **S4** to **S6**, and **WP5** addresses **S7**. The remaining work packages cross-cut the entire approach.

5. REFERENCES

[1] A. A. Almonaies, J. R. Cordy, and T. R. Dean. Legacy System Evolution towards Service-Oriented Architecture. In *Proc. Intl. Workshop on SOA Migration and Evolution (SOAME 2010)*, 2010.

[2] S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1), 2009.

[3] P. Brebner, L. O'Brien, and J. Gray. Performance Modeling for Service Oriented Architectures. In *Companion of the 30th Intl. Conf. on Software Engineering*, 2008.

[4] U. Breitenbücher, O. Kopp, F. Leymann, M. Reiter, D. Roller, and T. Unger. Six Strategies for Building High Performance SOA Applications. In *Proc. 4th Central-European Workshop on Services and their Composition (ZEUS 2012)*, 2012.

[5] E. Di Nitto, D. Meiländer, S. Gorlatch, A. Metzger, H. Psaier, S. Dustdar, M. Razavian, D. A. Tamburri, and P. Lago. Research Challenges on Engineering Service-oriented Applications. In *Proc. 1st Intl. Workshop on European Software Services and Systems Research: Results and Challenges*, 2012.

[6] R. Khadka, A. Idu, A. Saeidi, J. Hage, and S. Jansen. Legacy to SOA Evolution: A Systematic Literature Review. In *Migrating Legacy Applications – Challenges in Service Oriented Architecture and Cloud Computing Environments*. Premier Reference Source, 2013.

[7] H. Knoche, A. van Hoorn, W. Goerigk, and W. Hasselbring. Automated Source-Level Instrumentation for Dynamic Dependency Analysis of COBOL Systems. In *Proc. 14. Workshop Software-Reengineering (WSR 2012)*, 2012.

[8] J. Lewis and M. Fowler. Microservices, 2014. http://martinfowler.com/articles/microservices.html, last accessed: 2015-09-21.

[9] M. Litoiu. Migrating to Web Services: A Performance Engineering Approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(1-2), 2004.

[10] P. Merkle and H. Knoche. Extending the Palladio Component Model to Analyze Data Contention for Modernizing Transactional Software Towards Service-Orientation. In *Symposium on Software Performance (SSP 2015)*, 2015. To appear.

[11] S. Newman. *Building Microservices*. O'Reilly, Sebastopol, CA, 2015.

[12] L. O'Brien, P. Brebner, and J. Gray. Business Transformation to SOA: Aspects of the Migration and Performance and QoS Issues. In *Proc. 2nd Intl. Workshop on Systems Development in SOA Environments*, 2008.

[13] G. Pardon and C. Pautasso. Atomic Distributed Transactions: A RESTful Design. In *Proc. 23rd Intl. Conference on World Wide Web*, 2014.

[14] M. Razavian and P. Lago. A Frame of Reference for SOA Migration. In *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*. Springer, 2010.

[15] M. Razavian and P. Lago. Towards a Conceptual Framework for Legacy to SOA Migration. In *Service-Oriented Computing*, volume 6275 of *Lecture Notes in Computer Science*. Springer, 2010.

[16] M. Razavian and P. Lago. A Lean and Mean Strategy for Migration to Services. In *Proc. WICSA/ECSA 2012 Companion Volume*, 2012.

[17] R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Addison-Wesley, Boston, 2003.

[18] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proc. 3rd Intl. Conf. on Performance Engineering (ICPE 2012)*, 2012.