SERVICE COMPUTATION 2019

The Eleventh International Conferences on Advanced Service Computing

ISBN: 978-1-61208-702-3

May 5 - 9, 2019

Venice, Italy

SERVICE COMPUTATION 2019 Editors

Andreas Hausotter, Hochschule Hannover - University of Applied Sciences and Arts, Germany

SERVICE COMPUTATION 2019

Forward

The Eleventh International Conferences on Advanced Service Computing (SERVICE COMPUTATION 2019), held May 5 - 9, 2019 - Venice, Italy, continued a series of events targeting computation on different facets.

The ubiquity and pervasiveness of services, as well as their capability to be context-aware with (self-) adaptive capacities posse challenging tasks for services orchestration, integration, and integration. Some services might require energy optimization, some might require special QoS guarantee in a Web-environment, while others a certain level of trust. The advent of Web Services raised the issues of self-announcement, dynamic service composition, and third party recommenders. Society and business services rely more and more on a combination of ubiquitous and pervasive services under certain constraints and with particular environmental limitations that require dynamic computation of feasibility, deployment and exploitation.

The conference had the following tracks:

- Service innovation, evaluation and delivery
- Service quality
- Challenges
- Advanced Analysis of Service Compositions

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the SERVICE COMPUTATION 2019 technical program committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to SERVICE COMPUTATION 2019. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the SERVICE COMPUTATION 2019 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope SERVICE COMPUTATION 2019 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the

area of computation. We also hope that Venice provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

SERVICE COMPUTATION 2019 Chairs

SERVICE COMPUTATION 2019 Steering Committee

Mihhail Matskin, KTH, Sweden
Bernhard Hollunder, Hochschule Furtwangen University – Furtwangen, Germany
Paul Humphreys, Ulster Business School/University of Ulster, UK
Arne Koschel, Hochschule Hannover, Germany
Michele Ruta, Technical University of Bari, Italy
Alfred Zimmermann, Reutlingen University, Germany
Aida Omerovic, SINTEF, Norway
Annett Laube, Bern University of Applied Sciences (BUAS), Switzerland

SERVICE COMPUTATION 2019 Industry/Research Advisory Committee

Marcelo De Barros, Microsoft, USA
Steffen Fries, Siemens Corporate Technology - Munich, Germany
Matthias Olzmann, noventum consulting GmbH - Münster, Germany
Rong N. Chang, IBM T.J. Watson Research Center, USA
Jan Porekar, SETCCE, Slovenia

SERVICE COMPUTATION 2019

Committee

SERVICE COMPUTATION Steering Committee
Mihhail Matskin, KTH, Sweden
Bernhard Hollunder, Hochschule Furtwangen University – Furtwangen, Germany
Paul Humphreys, Ulster Business School/University of Ulster, UK
Arne Koschel, Hochschule Hannover, Germany
Michele Ruta, Technical University of Bari, Italy
Alfred Zimmermann, Reutlingen University, Germany
Aida Omerovic, SINTEF, Norway
Annett Laube, Bern University of Applied Sciences (BUAS), Switzerland

SERVICE COMPUTATION 2019 Industry/Research Advisory Committee
Marcelo De Barros, Microsoft, USA
Steffen Fries, Siemens Corporate Technology - Munich, Germany
Matthias Olzmann, noventum consulting GmbH - Münster, Germany
Rong N. Chang, IBM T.J. Watson Research Center, USA
Jan Porekar, SETCCE, Slovenia

SERVICE COMPUTATION 2019 Technical Program Committee

Antonia Albani, Institute of Information Management | University of St. Gallen, Switzerland
Pelin Angin, Middle East Technical University, Turkey
Irina Astrova, Tallinn University of Technology, Estonia
Jocelyn Aubert, Luxembourg Institute of Science and Technology (LIST), Luxembourg
Souvik Barat, Tata Research Development and Design Center, India
Carlos Becker Westphall, Federal University of Santa Catarina, Brazil
Ali Beklen, HotelRunner, Turkey
Kadda Beghdad Bey, EMP, Algiers, Algeria
Sulabh Bhattarai, Dakota State University, USA
Devis Bianchini, University of Brescia, Italy
Eugen Borcoci, University "Politehnica" of Bucharest, Romania
Juan Boubeta-Puig, University of Cádiz, Spain
Uwe Breitenbücher, University of Stuttgart, Germany
Antonio Brogi, University of Pisa, Italy
Oscar Mauricio Caicedo Rendon, University of Cauca, Brazil
Isaac Caicedo-Castro, University of Córdoba, Colombia
Juan-Carlos Cano, Universidad Politecnica de Valencia, Spain
Wojciech Cellary, Poznan University of Economics, Poland
Stephanie Challita, Inria Lille - Nord Europe, France
Chin-Chen Chang, Feng Chia University, Taiwan
Rong N. Chang, IBM T.J. Watson Research Center, USA
Huaming Chen, University of Wollongong, Australia

Dickson Chiu, The University of Hong Kong, Hong Kong
Marcelo De Barros, Microsoft, USA
Chiara Di Francescomarino, Fondazione Bruno Kessler (FBK), Italy
Leandro Dias da Silva, Instituto de Computação | Universidade Federal de Alagoas, Brazil
Chen (Cherie) Ding, Ryerson University, Canada
Erdogan Dogdu, TOBB University of Economics and Technology, Turkey
Cédric Eichler, LIFO-INSA Centre Val de Loire, France
José Enrique Armendáriz-Íñigo, Public University of Navarre, Spain
Sören Frey, Daimler TSS GmbH, Germany
Steffen Fries, Siemens Corporate Technology - Munich, Germany
Somchart Fugkeaw, Thai Digital ID Co. Ltd., Thailand
Filippo Gaudenzi, Università Degli Studi di Milano, Italy
Verena Geist, Software Competence Center Hagenberg GmbH, Austria
Katja Gilly, Universidad Miguel Hernández, Spain
Victor Govindaswamy, Concordia University - Chicago, USA
Steven Guan (Sheng-Uei Guan), Jiaotong-Liverpool University, China
Tom Guérout, LAAS-CNRS | Université de Toulouse, France
Maki K. Habib, The American University in Cairo, Egypt
Andreas Hausotter, Hochschule Hannover - University of Applied Sciences and Arts, Germany
Martin Henkel, Stockholm University, Sweden
Bernhard Hollunder, Hochschule Furtwangen University – Furtwangen, Germany
Wladyslaw Homenda, Warsaw University of Technology, Poland
Tzung-Pei Hong, National University of Kaohsiung, Taiwan
Wei-Chiang Hong, School of Education Intelligent Technology - Jiangsu Normal University, China
Sun-Yuan Hsieh, National Cheng Kung University, Taiwan
Marc Hüffmeyer, Furtwangen University, Germany
Marc-Philippe Huget, Polytech Annecy-Chambery-LISTIC | University of Savoie, France
Paul Humphreys, Ulster University, UK
Emilio Insfran, Universitat Politecnica de Valencia, Spain
Hemant Jain, University of Tennessee at Chattanooga, USA
E. E. Jan, IBM Global Technology Services, USA
Maria João Ferreira, Universidade Portucalense, Portugal
Eleanna Kafeza, Athens University of Economics and Business, Greece
Yu Kaneko, Research and Development Center - Toshiba, Japan
Fu-Chien Kao, Da-Yeh University, Taiwan
Marouane Kessentini, University of Michigan - Dearborn, USA
Moahmmad Maifi Hasan Khan, University of Connecticut, USA
Peter Kilpatrick, Queen's University Belfast, UK
Hyunsung Kim, Kyungil University, Korea
Alexander Kipp, Robert Bosch GmbH, Germany
Janusz Klink, Wroclaw University of Science and Technology, Poland
Christos Kloukinas, City University London, UK
Michal Kökörcený, Unicorn College / University of Hradec Králové, Czech Republic
Arne Koschel, Hochschule Hannover - University of Applied Sciences and Arts, Germany
Maria Krotsiani, City University London, UK
Andrew Kusiak, The University of Iowa, USA
Annett Laube, Bern University of Applied Sciences (BUAS), Switzerland
Tian Lan, George Washington University, USA

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Consistency for Microservices
A Legacy Insurance Core Application Migration Example

Arne Koschel
Andreas Hausotter

Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hannover, Germany
Email: arne.koschel@hs-hannover.de
Email: andreas.hausotter@hs-hannover.de

Moritz Lange
Phillip Howeihe

Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hannover, Germany
Email: moritz.lange@stud.hs-hannover.de
Email: phillip.howeihe@stud.hs-hannover.de

***Abstract*—In microservice architectures, data is often hold redundantly to create an overall resilient system. Although the synchronization of this data proposes a significant challenge, not much research has been done on this topic yet. This paper shows four general approaches for assuring consistency among services and demonstrates how to identify the best solution for a given architecture. For this, a microservice architecture, which implements the functionality of a mainframe-based legacy system from the insurance industry, serves as an example.**

Keywords–Microservices; Consistency; Insurance Industry.

I. INTRODUCTION

A current trend in software engineering is to divide software into lightweight, independently deployable components. Each component of the system can be implemented using different technologies because they communicate over standardized network protocols. This approach to structure the system is known as the microservice architectural style [1].

Typical goals of a microservice architecture are an overall resilient system and independent scalable components. To reach these goals, it is beneficial to decouple the services as much as possible. Therefore, according to Martin Fowler, each microservice should have its own data management [1]. Furthermore, services often hold redundant versions of data records to be able to operate independently. The synchronization of these data records however is a significant challenge. In context of our research we identified four general approaches for assuring consistency among services. This paper presents these and demonstrates how to identify the best solution for a given architecture. A migrated application from the insurance industry serves as an example.

The *Competence Center Information Technology and Management* (CC_ITM) is an institute at the University of Applied Sciences and Arts Hanover. Main objective of the CC_ITM is the transfer of knowledge between university and industry. As part of ongoing cooperation between the CC_ITM and two regional insurance companies, the research project *Potential and Challenges of Microservices in the Insurance Industry* was carried out. The goal was to examine the suitability of microservice architectures for the insurance industry. One part of this project was the migration of a monolithic mainframe-based core application, namely the `Partner Management System`. The resulting microservice architecture holds some data records redundantly and is hereby a good object for scientific research in context of consistency assurance.

The remainder of this article is organized as follows: After discussing related work in Section II, we show the core application system and address issues with the monolithic approach in Section III. Section IV introduces the architecture of the migrated system. In Section V we evaluate the outcomes with a focus on consistency aspects. Section VI discusses general approaches to ensure consistency in microservice architectures and how these approaches can be applied to get a suitable consistency solution for the `Partner Management System`. Section VII summarizes the results and draws a conclusion.

II. RELATED WORK

The basis of our research is the literature of well-known authors in the field of microservices. Worth mentioning are the basic works of Martin Fowler and James Lewis [1] as well as those of Eberhard Wolff [2]. For practical parts of our research, mainly the elaborations of Sam Newman (see [3]) were used. Especially for the migration of the legacy application, the works of Knoche and Hasselbring (see [4]) were consulted. As a study from the year 2019 shows (see [5]), microservice architectures are barely used in the insurance and financial services industry in Germany. Therefore, results from other industries had to be used for our research (for example [6]).

Although the basic literature is extensive, not much scientific research has been done about synchronizing services. Because microservices should use independent database schemes and can even differ in persistence technology, the traditional mechanisms of replicating databases (see, e.g., Tanenbaum and Van Steen [7, chap. 7]) cannot be applied as well. Instead, ideas and patterns from other areas of software engineering had to be transferred to the context of microservices.

So, in addition to general microservices research, more fundamental concepts of operating systems (e.g. [8]) and object-oriented programming (e.g. [9]) were considered by our research. Furthermore, concepts of general database research like the SAGA-Pattern [10], which was already applied to microservices by Chris Richardson [11], were considered as

 ISBN: 978-1-61208-702-3

well. Event-based approaches like Event Sourcing, described by Fowler [12], were also applied to microservices within our research.

Previous work has already evaluated the outcomes of the research project mentioned in the introduction (see [13]). However, the project and the evaluation based on it did not discuss the issue of consistency in detail. For this reason, our research focused on this topic after the completion of the project. In addition, a bachelor thesis written by one of the project members further dealt with the issue of consistency assurance in context of the project results (see [14]).

This paper gives an overview of existing research and summarizes it in a single document. To the best knowledge of the authors, this is the first summarizing scientific work on this topic.

III. Core Application: Partner Management System

As mentioned in the introduction, one part of the research project was to design a system for managing partners of an insurance company - the `Partner Management System`. In this context, partners are defined as natural or legal persons who are in a relation with the insurance company (e.g., clients, appraisers, lawyers or other insurance companies). Additionally, to general information about the person, a partner may also have information on communication, bank details, business relations and relations with other partners. Figure 1 shows the domain model and additional information about the service design, which is covered in the next section. This section focuses on the domain model, which was developed in cooperation with the insurance companies and is strongly based on the reference architecture for German insurance companies (VAA) [15].

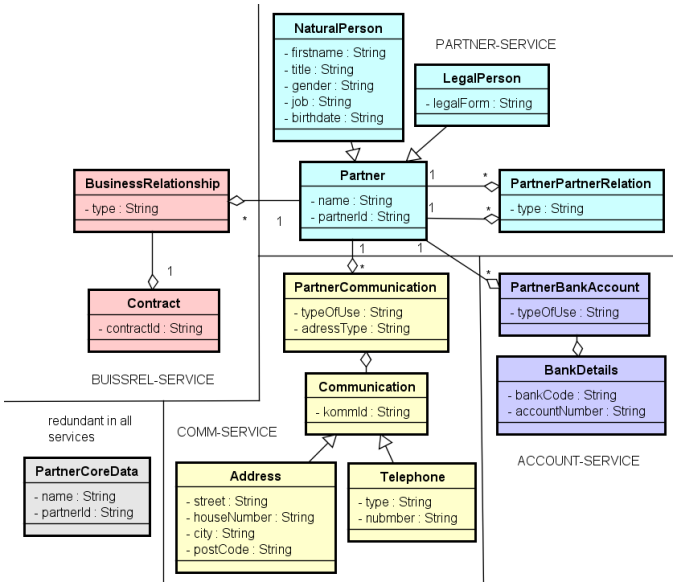

Figure 1. Domain model of the system.

At its core, the system is a simple CRUD application that manages the entity *Partner* and its properties. Usually, this functionality is implemented in a single SOA service. Modularization through SOA services provides significant benefits to the overall system (e.g., scalability and resilience), but the `Partner Management System` is still an atomic deployment unit that scales as a whole and fails as a whole. Since this system is a fundamental service of an insurance company, different parts of it are subject to varying levels of stress. Especially at night, the load profile differs. While only minor changes are made to existing datasets during the day, low occupancy during the night is used to slowly persist all new datasets collected on the day so as not to overburden the mainframe-based application. However, in practice this approach makes crashes at night extremely critical as the entire system does not work all night and only few people are available to fix the problem. The major issue of the existing implementation of the `Partner Management System` is obviously the poor flexibility, scalability and fault tolerance. This makes a microservices approach attractive for this use case.

IV. Microservice Architecture: Partner Management System

Figure 2 shows the architecture developed in close cooperation with the insurance companies, which subdivides the application into four independent services. Such a separation allows parts of the system to be scaled independently. Such a separation allows parts of the system to be scaled independently. For example, when the insurance company collects monthly premiums from its customers, this results in an increased load on the system that can be responded to by the scaling of the `account-service`.

To determine the separation into services, the original domain was divided into subdomains (as shown in Figure 1) and then the specific requirements (e.g., resilience) of each subdomain were analyzed. In order to make good use of the microservice architecture, the subdomains must be as independent as possible. That means there must be use cases where a subdomain can be used without any other. To achieve this independence, the architecture keeps certain parts of the partner (*PartnerCoreData*) redundant in all domains. This corresponds to the creation of bounded contexts, as described by Evans [16].

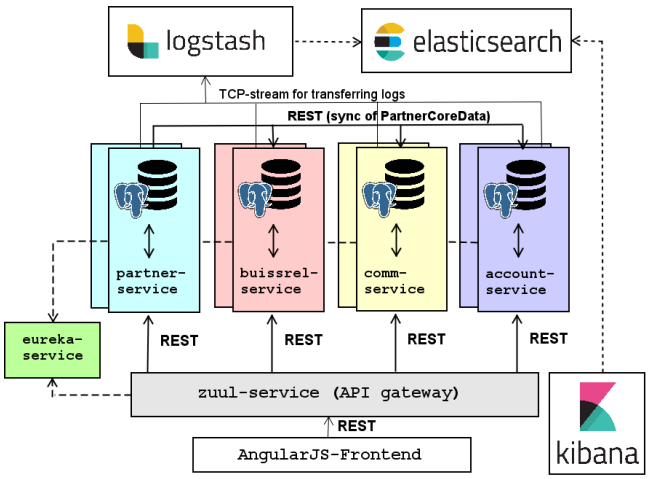

Figure 2. Infrastructure of the system.

Based on the technical specifications of the companies involved, the resulting subdomains were implemented as REST web services (see Figure 2) in Java using the Spring framework. As mentioned, each service should have its own data management, here realized as dedicated PostgreSQL databases. The *PartnerCoreData* is kept in sync across all services using REST calls of the `partner-service`. Parts of the Netflix OSS stack were used for the system infrastructure: Netflix Eureka (`eureka-service`) as a service discovery and Netflix Zuul (`zuul-service`) as an API gateway. Zuul also provides the web frontend of the application, which was realized as a single-page application using AngularJS. The ELK stack (Elasticsearch, Logstash and Kibana) was set up for monitoring and logging. All shown components of the architecture are deployed in separate Docker containers and connected by a virtual network using Docker Compose. In combination with the stateless architecture of the services, it is possible to run any number of instances of each service in a separate Docker container.

V. Challenges of the microservice architecture

Looking at the architecture described in sections III and IV, it looks like the microservice architecture can solve the problems of the current implementation. In particular, the scalability and fault tolerance of individual parts of the system are a crucial advantage over the current solution. The system can adapt to the changing load during the day, eliminating the need for risky nightly batch jobs. With the benefits of finer granularity however, there are also many new challenges that need to be mastered. One major challenge, for example, is the distributed monitoring and logging, which is handled by the ELK stack. As already mentioned, another key challenge of the developed microservice architecture is the consistency assurance across the services. More specifically, the synchronization of the *PartnerCoreData*, which serves as an example for the application of our research results.

As mentioned briefly in Section IV, the synchronization is realized by REST calls of the `partner-service`. Whenever a *PartnerCoreData* record is created, deleted or changed, the `partner-service` distributes this information to the other services in a synchronous way. This means that the `partner-service` is responsible for ensuring the consistency of the overall system. Furthermore, the development team of the `partner-service` is responsible for the data model of the *PartnerCoreData*, because it is part of their bounded context. This approach is known as Customer/Supplier Development (described by Evans [16]) in the context of domain-driven design.

This ensures that services are as independent of each other as possible. Even if, e.g., the `partner-service` is unavailable, the other services can still resolve foreign key relationships to partner data, because they keep a redundant copy of it. Moreover, the system reduces service-to-service calls, because other services don't need to call the `partner-service` on every operation. This ensures loose coupling of services, which is a key aspect of microservice architectures [1].

Since the synchronization of the *PartnerCoreData* is a critical part of the application, its implementation must be closer discussed. Since the goal of the first phase of the project was building the architecture in general, a synchronous solution was chosen for simplicity. This has several drawbacks:

- **Fault tolerance.** If the `partner-service` crashes during synchronization, some services might not be notified about the changes. Conversely, if another service can not be contacted by the `partner-service`, it will also not be notified. This is due to the transient communication.
- **Synchronicity.** After a change of *PartnerCoreData*, a thread of the `partner-service` is in a blocked state until all other services have been notified. Because multiple network calls are necessary for the synchronization, the general performance of the `partner-service` is affected. Since microservices should be lightweight, a large number of network calls and busy threads are a serious problem.
- **Extensibility.** Since the extension of a microservice architecture is a common occurrence, extensibility is a major aspect. In the current implementation, the `partner-service` holds a static list of services that need to be notified upon a change of *PartnerCoreData*. If a new service is added to the system, which is interested in *PartnerCoreData*, the `partner-service` must be redeployed. Additionally, the bigger the number of services to notify gets, the more the performance of the `partner-service` is impaired.

As part of our research, further alternative solutions were explored, which will be discussed in the next sections.

VI. Consistency assurance in the Partner Management System

In order to find a suitable solution for the specific problem of synchronizing *PartnerCoreData*, the general approaches have to be examined.

A. General approaches

The central research question to identify general approaches for consistency assurance is how a change in master data can be distributed to other interested services without breaking general microservices patterns like loose coupling and decentral data management. Especially the latter makes this a major challenge: Because the data stores and schemes should be separated, the standard mechanisms of synchronizing databases cannot be used here.

Based on our research, there are four possible solutions for synchronizing redundant data in microservices:

- **Synchronous distribution.** One approach is that the owner of the data distributes every change to all interested services. As discussed in Section V, the developed microservices architecture already follows this approach. To provide loose coupling however, the addresses of services to notify should not be contained in the master service's code. A better solution is to hold those addresses in configuration files, or even better, establish a standard interface where services can register themselves at runtime. For example, an existing service registry (e.g. Netflix Eureka) can be used to store this information. This approach roughly corresponds to the *Observer-Pattern* of object orientated software development.

 ISBN: 978-1-61208-702-3

As already discussed, notifying a large number of interested services might cause significant load of the service containing the master data. This can become a disadvantage. The distribution takes place in a synchronous fashion however, directly after the change of data itself. This means that this solution provides a high degree of consistency among services.

- **Polling.** One other solution might be to relocate the responsibility of aligning the redundant data to the interested services themselves. A straight forward approach here is to periodically ask for new data using an interface provided by the service containing the master data. Based on timestamps, multiple data updates can be transferred in one go. The size of the inconsistency window can be controlled by each interested service independently via the length of the polling interval. However, despite being consistent in the end, the time frame in which the data sets might differ is a lot larger than the one when using a synchronous solution. This model of consistency is known as eventual consistency (see [17]).
- **Publish-Subscribe.** To completely decouple the service containing the master data from the other services, message-oriented middleware can be used. On every data change, an event is broadcasted on a messaging topic following the publish-subscribe-pattern. Interested services subscribe to this topic, receive events and update their own data accordingly. Multiple topics might be established for different entities. If the messaging system is persistent, it even makes the architecture robust against services failures. This approach suits the resilient and lightweight nature of microservices. It must be noted however that it also falls in the category of eventual consistent solutions - until the message is delivered and processed, the system is in an inconsistent state.
- **Event Sourcing.** Instead of storing the current application state, for some use cases it might be beneficial to store all state transitions and accumulate those to the current state when needed. This approach can also be used to solve the problem of distributing data changes. Upon changes in master data, events are published. Unlike the publish-subscribe solution however, the history of all events is persistent in a central, append-only event storage. All services can access it and even generate their own local databases from it, each fitting their respective bounded context. This solution provides a high degree of consistency: Each data change can be seen immediately by all other components of the system. It must be noted that a central data store, which microservices try to avoid, is introduced. This weakens the loose coupling and might be a scalability issue - the append-only nature of the data storage enables high performance though.

If none of the consistency trade-offs above is bearable, this can be an indicator that the determined subdomains are not optimal. In some cases, subdomains are coupled so tightly that keeping data redundantly is not feasible. In this case, it should be discussed if the services can be merged. Furthermore, if this issue occurs in several parts of the architecture, it should be evaluated if a microservices approach is the right choice for this domain.

B. Approaches for synchronizing PartnerCoreData

The discussion in the previous section has shown that some approaches tend to guarantee a stronger level of consistency then others. This means that before all non-functional requirements can be considered as decision criteria, the required consistency degree of the underlying business processes must be examined. This can be done by first specifying the possible inconsistent states and then combining them with typical use cases of the system.

In case of the `Partner Management System`, only the *PartnerCoreData*, containing the name and id of every *Partner*, is saved in a redundant fashion. Combining these with the CRUD-operations, the following inconsistent states are possible:

- A new *Partner* might not yet be present in the whole system.
- Name or Id might not be up-to-date.
- A deleted *Partner* might not yet be deleted everywhere.

As part of our research, we combined these inconsistent states with typical use cases and business processes in which the `Partner Management System` is involved, like sending a letter via mail or a conclusion of an insurance contract.

The result of this examination is that the partner management of insurance companies is surprisingly robust against inconsistent states. This is mainly due to the reason that the business processes itself are already subject to inconsistency: If a customer changes its name for example, the inconsistency window of the real world is much larger than the technical one (the customer e. g. might not notify the insurance company until several days have passed). The postal service or bank already needs to cope with the fact that the name might be inconsistent. The discussion of other potential situations brought similar results. This makes sense because the business processes of insurance companies originated in a time without IT, which means that they are already designed resilient against delays and errors caused by humans. Cases where the customer notices the delay (e.g., a wrong name on a letter) are rare and justifiable.

In summary, the combination of the inconsistent states and the use cases of the `Partner Management System` revealed, that a solution which promotes a weaker consistency model can be used - no approach has to be excluded beforehand. So, the choice of a synchronization model is only influenced by the non-functional requirements. The analysis of the partner domain showed that the main non-functional requirements are lose coupling, high scalability and easy monitoring. Especially because of loose coupling, the publish-subscribe pattern is the most viable solution.

VII. Conclusion and Future Work

Synchronizing redundant data across services is a key challenge of microservice architectures. However, our research showed that the solutions can be reduced to four general approaches. For choosing a suitable solution for a given

architecture, the underlying domain needs to be analyzed: Possible inconsistent states need to be combined with typical use cases. As the example of the `Partner Management System` shows, this combination can reveal that domains might be much more resilient against inconsistencies as one would first assume.

The next steps will be to work out a specific design of the publish-subscribe approach for the `Partner Management System`. Furthermore, the complete implementation must be done, and the system must be tested under real-world conditions.

To demonstrate the procedure we have developed for finding a suitable consistency assurance solution, the example of the `Partner Management System` is sufficient. To further underpin our findings however, they need to be applied to more complex examples.

REFERENCES

[1] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," https://martinfowler.com/articles/microservices.html, March 2014, [retrieved: 04, 2019].

[2] E. Wolff, Microservices: Flexible Software Architecture. Addison-Wesley Professional, 2016.

[3] S. Newman, Building microservices: designing fine-grained systems. O'Reilly Media, Inc., 2015.

[4] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," IEEE Software, vol. 35, no. 3, 2018, pp. 44–49.

[5] ——, "Drivers and barriers for microservice adoption–a survey among professionals in germany," Enterprise Modelling and Information Systems Architectures (EMISAJ), vol. 14, 2019, p. 10.

[6] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, 2017, pp. 243–246.

[7] A. S. Tanenbaum and M. Van Steen, Distributed Systems: Pearson New International Edition - Principles and Paradigms. Harlow: Pearson Education Limited, 2013.

[8] A. S. Tanenbaum, Modern Operating Systems. New Jersey: Pearson Prentice Hall, 2009.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software. Amsterdam: Pearson Education, 1994.

[10] H. Garcia-Molina and K. Salem, "Sagas," vol. 16, no. 3. ACM, 1987.

[11] C. Richardson, Microservices Patterns: With examples in Java. Manning Publications, 2018.

[12] M. Fowler, "Event Sourcing," https://martinfowler.com/eaaDev/EventSourcing.html, December 2005, [retrieved: 04, 2019].

[13] M. Lange, A. Hausotter, and A. Koschel, "Microservices in Higher Education - Migrating a Legacy Insurance Core Application," in 2nd International Conference on Microservices (Microservices 2019), Dortmund, Germany, 2019, https://microservices.fh-dortmund.de/papers/Microservices_2019_paper_8.pdf [retrieved: 04, 2019].

[14] P. Howeihe, "Transactions and consistency assurance in microservice architectures using an example scenario from the insurance industry," bachelor thesis at Univ. of Applied Sciences and Arts Hanover, 2018.

[15] GDV, "The application architecture of the insurance industry - applications and principles," 1999.

[16] E. J. Evans, Domain-driven Design - Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional, 2004.

[17] W. Vogels, "Eventually Consistent," Commun. ACM, vol. 52, no. 1, Jan. 2009, pp. 40–44. [Online]. Available: http://doi.acm.org/10.1145/1435417.1435432

 ISBN: 978-1-61208-702-3

Applying Microservice Principles to Simulation Tools

Richard Pump Arne Koschel Volker Ahlers
Department of Computer Science
University of Applied Sciences and Arts
Hannover, Germany
Email: {richard.pump | arne.koschel | volker.ahlers}@hs-hannover.de

***Abstract*—The usage of microservices promises a lot of benefits concerning scalability and maintainability, rewriting large monoliths is however not always possible. Especially in scientific projects, pure microservice architectures are not feasible in every project. We propose the utilization of microservice principles for the construction of microsimulations for urban transport. We present a prototypical architecture for the connection of MATSim and AnyLogic, two widely used simulation tools in the context of urban transport simulation. The proposed system combines the two tools into a singular tool supporting civil engineers in decision making on innovative urban transport concepts.**

Keywords–Microservices; Simulation; Urban Logistics

I. INTRODUCTION

Urban transport and logistics are evolving fields of research. Modern concepts ranging from crowd-sourced delivery platforms like Foodora, to drone based delivery are changing the transport of goods. At the same time, services like Uber shift personal transport away from public transport solutions towards crowd-sourced ride sharing. A modern civil engineer not only has to keep an overview over the ever evolving modern concepts, but also know their advantages and disadvantages, as well as their impact on different factors like CO2-emissions and noise pollution.

To support civil engineers, the *University of Applied Sciences and Arts Hannover* is working in cooperation with the *Leibniz University Hannover*, the *City of Hannover* and the *Technical University Braunschweig* to create a decision and support tool for urban logistics [1]. The support tool allows the simulation of novel ideas for urban transport, comparing the impact of different ideas and visualizing the results adequately for easy comprehension. This helps decision makers to test their ideas against the real world without committing significant resources.

The tool combines two simulation frameworks with data storage and a unified front end, improving the usability of complex software. To combine the simulation frameworks, we decided to use a microservice-based approach. Using microservice principles allows us to develop parts of the system independently, which conforms to the project's organizational structures. With multiple partners building the system in different physical locations, independence becomes paramount. However, a conventional application of microservices, 'splitting the monolith' as it is often called, is not possible, since the main goal of the project is the development of new software using existing frameworks, instead of building a new system.

This paper will present our current work on the design of the tool. In Section II we will present a short overview of works relevant to this paper. Section III gives a short overview of the goals and requirements of the decision support tool and a rough system sketch is shown in Section IV. The Sections V and VI contain the main contents of the paper, showing microsimulkations and discussion advantages and disadvantages of the presented approach. The paper ends with a conclusion in Section VII.

II. RELATED WORK

The usage of microservices is a widely discussed topic in current research. Sam Newman provides a thorough overview over the microservice paradigms, as well as their advantages in [2]. While giving general recommendations the work does not follow our specific use-case of combining two simulation tools. As we lack the time and resources, splitting up existing applications isn't feasible within the boundaries of the project. Our work differs by combining microservices with production monoliths into a conglomerate design, splitting where sensible and feasible but purposefully keeping monoliths where not, as opposed to replacing the entire application design with microservices.

In [3], Nicola Dragoni et al. provide a comprehensive overview about microservices, as well as some definitions. The article however focuses on the conceptual ideas behind microservices, instead of a single application. A history of issues is presented concerning past issues, current issues and issues most likely appearing in the future. In the conclusion, the authors provide an opinion, viewing microservices as evolutionary rather than revolutionary, which our application of microservice principles shares. We intend to apply microservice paradigms only on parts of our architecture, providing a next step in designing simulation tools, rather than completely revolutionizing simulation architecture.

The term microsimulation used in this paper differs from the microsimulation models described by Merz [4] and others. Microsimulation models are based on microinformation and model only small parts of the system to be simulated, while not defining an architectural pattern used in the development for those models. While the simulation scenarios described in our paper might fall into the category of microsimulation models, the modeling process and simulation-theoretic backgrounds are out of scope.

Lastly, we use the simulation frameworks MATSim and Anylogic. Horni, Nagel and Axhausen present MATSim in [5]. They describe an open source, Java-based simulation tool for agent-based transport simulation. The MATSim architecture is modular to allow for flexible extension of simulation models but does not utilize microservices. AnyLogic is mainly presented in the works of Grigoryev [6] and Borshchev [7].

 ISBN: 978-1-61208-702-3

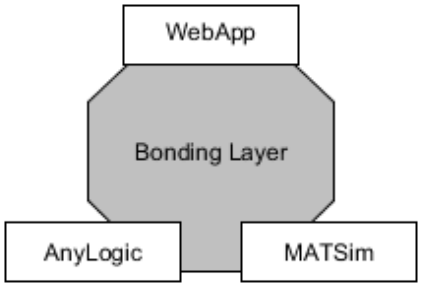

Figure 1. Rough system sketch of the decision support tool, showing the four major components.

AnyLogic is a proprietary, closed source, Java-based general purpose simulation tool. Again, microservices are not used in the simulation design.

Overall there are no works known to the authors that apply microservices to simulations.

III. REQUIREMENTS

The overall goal of the system is providing relevant information about novel logistic concepts to city planners. A city planner must be able to evaluate the impact of a logistic concept on his city, town or village.

The first step to make a decision is gathering information. The decision support and information system therefore has to inform the user about the different novel logistic concepts. Every concept has to be presented in an easily comprehensible way, e.g., by a short video. Not only concepts have to be presented, but also information about the city areas that can be used to evaluate the impact of the concept.

After informing the user about concept and city area, the software is to provide a configuration utility, allowing the user to modify parameters of the concept or the city area. The user should be able to save his configuration, as well as to load and edit old configurations, before starting the simulation.

The last step in using the information system is to evaluate the simulation results. The system has to present relevant information about the logistic concept stemming from the simulation in an easy to comprehend way. Also, comparisons with other simulated scenarios must be possible.

Not all of the information available within the tool is public, therefore an access control has to be implemented. The access control allows fine grained configuration of function and information access.

Furthermore the two simulation frameworks AnyLogic and MATSim have to be utilized to build the decision support tool. The project development team has experience in utilizing the two tools and switching to another framework is too costly.

IV. ROUGH SYSTEM SKETCH

Building upon the aforementioned requirements, the rough system sketch helps to recognize the architecture as a whole and is rather abstract. Figure 1 shows the four major components of the software.

The component ANYLOGIC is responsible for microscopic traffic simulation, modeling individual behavior using the simulation framework AnyLogic. For example, within the component ANYLOGIC, acceptance profiles for novel logistic solutions are simulated, giving a realistic representation of the individual person using logistics. Also, a precise modeling of the novel concept parts is achieved, e.g., the differently cooled compartments of e-grocery delivery vans are simulated. While giving a detailed view on the microscopic level of logistics, macroscopic events are not within scope of the ANYLOGIC component.

For the simulation of macroscopic events, the component MATSIM is used. Based on the equivalently named multi-agent transport simulation-framework, the component is responsible for the simulation of an entire city. For performance reasons, it models just the general movement of agents within the city, instead of concrete agent decisions. This allows the user to evaluate the impact of concepts on traffic flow and vice versa. Combining the MATSIM component with the ANYLOGIC component creates a holistic simulation for novel logistic concepts, modeling microscopic individual behavior and macroscopic events.

With results and analysis generated, results have to be presented within a graphical user interface, easily accessible from many locations. The component WEBAPP provides the graphical interface to access functions of the system, configure and run simulations, and compare the impacts of different novel logistic concepts. In contrast to the other two components, no external frameworks dictate architectural decisions, therefore the WEBAPP will be the only pure microservice-component.

The last component will be the bonding layer, which is a purely technical component, bridging the monolithic frameworks with the microservice-oriented WEBAPP. It is not a component in the traditional sense, since it does not group parts of the software which are responsible for a single piece of the domain logic. Also, the bonding layer contains elements that interact with the different simulation frameworks.

Generally, to evaluate a certain novel logistic concept, the user will select the concept within the WEBAPP, configure a simulation using one or both of the simulation frameworks, simulate her ideas and use the WEBAPP to evaluate the simulation results. The user can only choose between predefined logistic concepts.

In the following section, we will further explain the bonding layer.

V. THE BONDING LAYER

Utilizing two monolithic simulation frameworks results in problems when employing microservices. A completely clean, monotheistic architecture would require a reconstruction of the frameworks, which isn't feasible. We therefore propose to employ microservice ideas in the utilization of the tools, effectively encapsulating the rigidity of the frameworks within a flexible shell. This creates a flexible bonding layer, enabling rapid development.

Figure 2 shows the microservice-like approach to simulation using an external framework. Instead of developing a single, holistic simulation, we propose the development of small, interchangeable microsimulations. The microsimulation only implements the specifics of the logistic concept to be simulated and uses core functionalities or other microsimulations to further increase model accuracy.

 ISBN: 978-1-61208-702-3

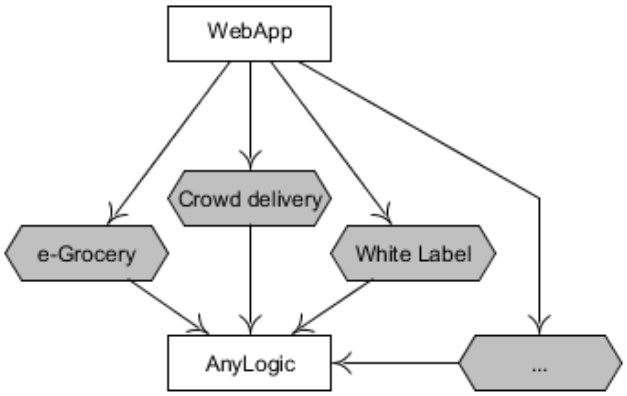

Figure 2. Microsimulations using Anylogic.

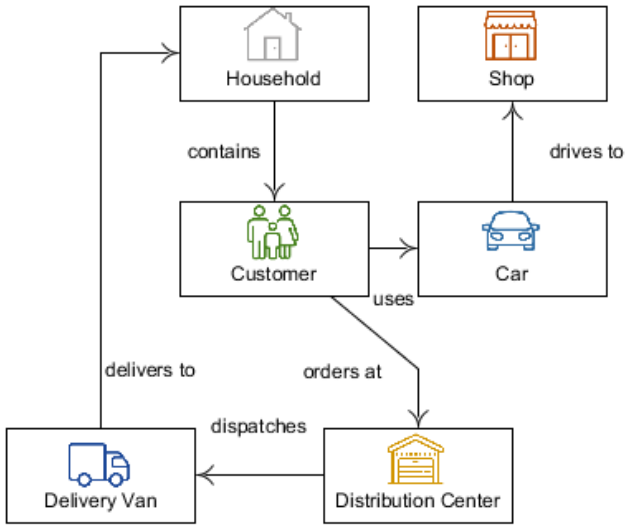

Figure 3. E-Grocery microsimulation using AnyLogic.

A. Microsimulations in AnyLogic

Traditional simulation software is often created by building a simulation framework that encompasses the technical aspects of the simulation. For example the Framework OMNet++, presented by Varga and Hornig in [8], provides an event-based simulation for networks in general. Further functionality is than added on top in the form of modules or add-ons to model specific types of scenarios. To model Internet traffic in OMNet++, an external library called INET is often used, which provides modeling of network stacks, switches, Ethernet cables, etc. The add-ons often model different domains, but no singular scenario. In our architecture we propose to build another layer on top of the simulation framework and the add-ons, with each module containing a specific, highly configurable scenario that is to be simulated.

For example, the microsimulation e-Grocery only implements the necessary supply chain for on-line grocery shopping. It uses the framework AnyLogic and its add-ons that provide GIS-support, agent-based simulation, database access, etc. Figure 3 shows the implemented agents for the simulation.

Each agent represents a different step of the digital or the conventional grocery shopping process. Following the classic agent-based simulation design, the agents all have specific behavior and interactions with other agents. The whole simulated process is a result of emergent behavior.

The CUSTOMER-agent models the customer in the e-grocery process. Depending on certain factors like age and income, the agent decides to either buy her groceries via the online store or the conventional local grocery store. If the online store is used, the agent places an order to the store and awaits delivery.

The STORE agent represents a local grocery store that provides a certain inventory and is opened during a specific time frame, depending on the store type. For the e-grocery model, a rather high abstraction for the shopping process is adequate, CUSTOMERS arrive via CAR and spend a predefined amount of time in the STORE before returning to their HOUSEHOLD.

The DISTRIBUTION CENTER however is part of the e-grocery concept. It receives orders from the CUSTOMERS and sorts the orders into delivery trips for the DELIVERY VANS to execute. Specific aspects of grocery delivery are considered during the trip planning phase. For example, if the order contains refrigerated goods the delivery window specified by the customer needs to be hit by the van, otherwise the goods might spoil. After planning trips, the DISTRIBUTION CENTER sends out the DELIVERY VANS to distribute the ordered goods.

The DELIVERY VAN is a simulation of a delivery van built for grocery delivery, containing multiple different temperature zones that can store different kinds of groceries and perishables. It receives a delivery plan from the DISTRIBUTION CENTER, is loaded with the ordered goods and proceeds to traverse a GIS map (provided by AnyLogic), stopping at the HOUSEHOLDS to deliver the cargo.

Each agent is very simple in implementation and the whole microsimulation can be replaced by another version within a month. Furthermore the agents are highly configurable. For example, the CUSTOMER follows a pre-defined schedule for daily activities, which can be configured by the user, changing delivery windows, ordered goods and traffic.

Overall we implement small specific scenarios for urban logistic in independent AnyLogic simulations, called microsimulations. Keeping scope small allows for rapid development and independent deployment.

B. Macroscopic Simulation

With AnyLogic-based simulations relatively small in scope, macroscopic events need to simulated by another framework. We use MATSim for this purpose. The aforementioned microsimulation allows evaluation of economical factors and provides a basis for the more abstract complete city model simulated by MATSim.

Figure 4 shows the general workflow of simulations using MATSim. To simulate traffic, MATSim creates a population of agents and optimizes each agent's day plan cost using a genetic algorithm, changing modes of transportation between activities. Optimization of agent plans ends after a pre-defined,

 ISBN: 978-1-61208-702-3

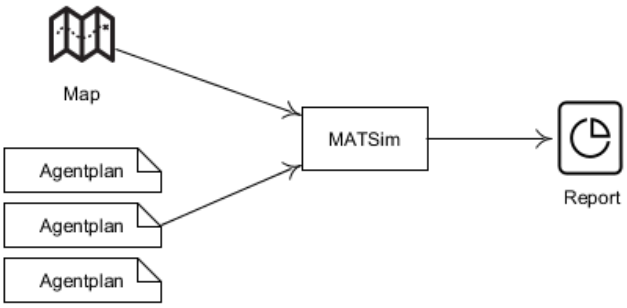

Figure 4. General Workflow using MATSim.

user-controllable amount of iterations. The simulation creates an output in form of reports, showing traffic flows, emissions and road utilization to be displayed by the WEBAPP.

VI. DISCUSSION

The usage of microservices often implies a completely new architecture and replacing the old system piece by piece with a microservice-conforming re-implementation. Microservices provide extreme advantages at scale and rapid development of new components. However, the necessary organizational structures for microservices cannot always be implemented and rewriting software is not always possible.

In our current research project, we can not rewrite extensive simulation frameworks to allow rapid implementation of new simulations. We therefore provided an adaptation of microservice principles to simulations, creating small scale microsimulations. This approach has a couple of advantages and disadvantages.

First, the usage of microsimulation relies on monolithic simulation frameworks. By modeling only small parts of the scenario, a rich framework is necessary to provide extensive functionalities like routing services, message channels and logging. A loose coupling between simulated scenario and framework is difficult to achieve. API changes to the frameworks also impact all microsimulations.

This approach also only works for simulations with a low degree of complexity. Very complex simulation models can rarely be written by a very small team in a reasonable time frame, without producing a complex to change system.

On the other hand, the usage of microsimulations allows rapid technical development of new simulations, reducing overall development time. The development of a simulation model often consists of a long phase gathering data about real life behavior and structure before beginning implementation. Reducing the necessary time for technical development frees up time for more detailed research, increasing simulation accuracy.

Furthermore, keeping simulation scenarios small allows for single person simulation development. Implementation of functionalities is reduced extremely if a rich framework can be used. By only modeling a single concept and using parameters for configuration scenarios, the necessary code base to be implemented by the developer is very small. This is a big advantage in research projects where the software is to be used as a tool in the project context.

VII. CONCLUSION

In this paper, we first presented our requirements for a web-application for city planners. The web tool aims to support city planners and decision makers in their evaluation of novel logistic concepts in regards to the impact of novel concepts on emissions, traffic flow and road utilization.

To achieve this goal, we presented a design for the decision support tool that incorporates two different simulation frameworks. The frameworks are combined using a bonding layer that bridges the microservice-paradigms and the organizational challenges in rewriting foreign software. The bonding layer applies the concept of small independent components to the simulations themselves by constraining the simulations to a single logistic concept, relying on the frameworks and other microsimulations to model more holistic views.

We also discussed advantages and disadvantages of the microsimulations. Most importantly, dependencies on simulation frameworks prevent microsimulations from adhering to a pure microservice approach. The impact of this disadvantage is however dependent on the software itself. In projects where software is produced as a prototype and not subject to further development and maintenance, the changes to used libraries do not impact the development process, as older versions of the library can easily be used. therefore, in a research project this trade-off might be considered, as some advantages from the microservice paradigm are immensely useful in small scale experimental work.

In further work, we plan to explore the applicability of microsimulations to the cloud context, as well as further evaluation of the proposed architecture.

ACKNOWLEDGMENT

This work was supported by the Federal Ministry of Education and Research of Germany (project USEfUL, grant no. 03SF0547). We would like to thank our colleagues from the Faculties for engineering and business information systems, as well as the colleagues from the other institutions and the City of Hannover.

REFERENCES

[1] Urbane Logistik Hannover (urban logistics Hannover). Retrieved April 2019. [Online]. Available: https://www.hannover.de/Urbane-Logistik-Hannover

[2] S. Newman, Building microservices: designing fine-grained systems. O'Reilly, 2015.

[3] N. Dragoni et al., "Microservices: yesterday, today, and tomorrow," in Present and Ulterior Software Engineering. Springer, 2017, pp. 195–216.

[4] J. Merz, "Microsimulation as an instrument to evaluate economic and social programmes," MPRA Paper 7236, 1993.

[5] A. Horni, K. Nagel, and K. W. Axhausen, The multi-agent transport simulation MATSim. Ubiquity Press London, 2016.

[6] I. Grigoryev, AnyLogic 6 in three days: a quick course in simulation modeling. AnyLogic North America, 2012.

[7] A. Borshchev, The big book of simulation modeling: multimethod modeling with AnyLogic 6. AnyLogic North America Chicago, 2013.

[8] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops. ICST, 2008, p. 60.

 ISBN: 978-1-61208-702-3

Towards a Microservices-based Distribution for Situation-aware Adaptive Event Stream Processing

Marc Schaaf

University of Applied Sciences Northwestern Switzerland,
Riggenbachstr. 16, 4600 Olten, Switzerland
Email: marc.schaaf@fhnw.ch

***Abstract*—This paper presents the central concepts for a microservices-based distribution of event stream processing pipelines as they are part of our situation-aware event stream processing system. For this, we outline changes to our specification language for a clear separation of the stream processing specification from the actual stream processing engine. Based on this separation, we then discuss our mapping approach for the assignment of the pipelines to stream processing nodes.**

Keywords–Event Stream Processing; Microservices; Service-oriented Architecture

I. INTRODUCTION

Event Stream Processing (ESP) applications play an important role in modern information systems due to their capability to rapidly analyze huge amounts of information and to quickly react based on the results. They follow the approach to produce notifications based on state changes (e.g., stock value changes) represented by events, which actively trigger further processing tasks. They contrast to the typical store and process approaches where data is gathered and processed later in a batch processing fashion, which typically involves a higher latency. ESP applications can achieve scalability even for huge amounts of streaming event data by partitioning incoming data streams and assigning them to multiple machines for parallel processing. Due to those properties, ESP based analytical systems are likely to have a further increasing relevance in future business systems. Also, it is likely that future ESP applications will have to handle even larger amounts of data while taking on increasingly complex processing tasks to allow for near real-time analytics to take place.

An example for such a scenario is the detection and tracing of solar energy production drops caused by clouds shading solar panels as they pass by [1]. The scenario requires a processing system to handle large amounts of streaming data to (1) detect a possible cloud (a possible situation), to (2) verify the possible situation and (3) to track the changes of the situation as the cloud moves or changes its size or shape. For the initial detection of a potential situation, a processing system needs to analyze the energy production of all monitored solar panel installations. However, for the second part, the verification of a potential cloud, only a situation specific subset of the monitoring data is needed. In the same way, the later tracking of the situation only requires a situation specific subset, which may change over time.

In order to handle such large numbers of events, a processing system needs to be capable of distributing the processing across several machines. A common mechanism for the distribution is to partition the overall data stream [2]. When a processing system partitions the incoming data streams in order to achieve scalability, such a partitioning will be suitable for the first part of the processing, the detection of a potential situation as the partitioning is *situation independent*. For the later processing part where a *situation specific* subset of the incoming data streams is required, a general stream partitioning scheme based on for example the processing system load, is not suitable as it does not incorporate the needs of currently analyzed situations. Here, a dynamic adaptation mechanism is needed that takes the investigated situations state into account.

While we presented the general high level architecture of our processing system in [3], this paper discusses two contributions, first the partial re-design of our specification language to become independent of the Drools Rule Engine and second the the assignment of the actual stream processing as a set of microservices to allow for the scalable distributed deployment of the stream processing pipelines.

The remainder of this paper is structured as follows: The next section discusses the related work followed by a presentation of the processing model. Section III and IV present the basics of our specification language for situation-aware event stream processing and outline the made changes. Section V presents the architecture and the mapping of stream processing pipelines to a microservice-based architecture.

II. RELATED WORK

Various systems for distributing a processing system in order to provide the needed scalability exist like Aurora* and Borealis [4][5]. Aurora*, for example, starts with a very crude data stream partitioning in the beginning and tries to optimize its processing system over time based on gathered resource usage statistics [6]. Furthermore, various approaches have been proposed, which employ adaptive optimizations to handle load fluctuations by utilizing the dynamic resource availability of cloud computing offerings like [7][8][9] in order to scale on demand.

In general, the discussed systems are capable of setting up distributed stream processing based on given queries and to optimize the system to provide the required processing capacity and response times. However, the systems have no mechanisms to adapt deployed stream queries based on detected situations and situation changes as they have no knowledge of the overall analytical task that deployed a given stream query.

 ISBN: 978-1-61208-702-3

On the other hand, there are systems aimed specifically at providing the surroundings for distributed processing but without providing processing languages like for example, Apache Storm [10] or Apache Spark Streaming [11]. Such systems could act as a potential basis for implementing our situation-aware adaptive processing model. However, they do not follow the microservice model we aim to explore with our architecture. For the realization of a reactive microservice like architecture as we aim for with our approach, lower level frameworks exist like for example Eclipse Vert.X [12] or Akka [13]. For our processing system architecture, we utilize Vert.X due to its integrated event bus functionality and define a mapping of our processing model to the available functionalities.

III. Processing Model and Language

We approach the initially outlined problem by defining a *situation-aware adaptive stream processing model* together with a matching *scenario definition language* to allow the definition of such processing scenarios for a scenario independent processing system [3][14][15][16]. The designed model defines situation-aware adaptive processing in three main phases (Figure 1):

- Phase 1: In the *Possible Situation Indication* phase, possible situations are detected in a large set of streaming data, were the focus lies on the rapid processing of large amounts of data, explicitly accepting the generation of false positives and duplicate notifications over precise calculations.
- Phase 2: The *Focused Situation Processing Initialization* phase determines whether an indicated possible situation needs to be investigated or if it can be ignored, for example because the situation was already under investigation. If a potential situation needs to be investigated, a new situation specific focused processing is started.
- Phase 3: In the *Focused Situation Processing* phase, possible situations are first verified and then an in-depth investigation of the situation including the adaptation of the processing setup based on interim results is possible.

For these three phases, event stream processing takes place during the Phases 1 and 3.

Based on our processing model, we defined the Scenario Processing Template Language (SPTL), which allows the specification of processing templates based on the concepts of the processing model in an implementation independent way.

IV. Specification Language

A Scenario Processing Template contains all scenario-specific information to parameterize a processing system for a scenario (e.g., How to detect a train delay and how to determine its impact). The template is divided into a preamble and three blocks which resemble the three major phases defined in the processing model as outlined in the listing in Figure 2.

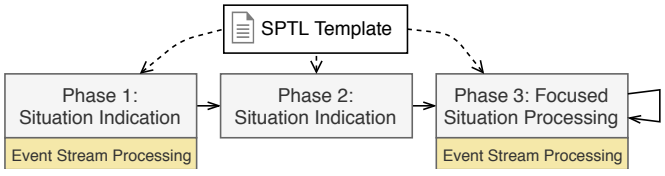

Figure 1. Three main phases of the Processing Model

```
name "ScenarioName"

PossibleSituationIndication {
  // [...] Specifications for Processing Phase 1
}

FocusedSituationProcessingInitialization {
  // [...] Specifications for Processing Phase 2
}

FocusedSituationProcessing {
  // [...] Specifications for Processing Phase 3
}
```

Figure 2. Structure of a processing template in the SPTL

Each block contains the specifications required for the setup and execution of the corresponding phase. Within the here discussed processing model, scenario specific event stream processing takes place during the Phases 1 and 3. Thus, the definitions of these two phases each contain the definition on how the scenario specific event stream processing has to be done. In the old version of the SPTL, the specification needed to be given as a Stream Processing Builder statement. This statement contained a mixture of several languages (Drools [17], MVEL [18], SPARQL [19]) in order to build/generate the actual event stream processing rules in the Drools Language.

A. Language Changes

One of the main limitations of this first version of the SPTL lies in its tight link to the Drools Rule engine, as well as in the complexity resulting from the combination of several languages. The tight link originates in the definition of the the actual stream processing statements which needed to be given based on the Drools Rule Language as shown in the listing in Figure 4.

In order to decouple the SPTL from the Drools Rule language and to further ease the stream processing specifications, the SPTL was extended with its own stream processing specification language. The new language is build around the concept of a stream processing pipeline as shown in the listing inf Figure 5.

The first statement $$focusArea.delay specifies the source of the events together with the type of the event ”DelayEvent”, the second statement ”filter(...)” defines a filtering condition followed by the last statement, which specifies a small function that shall be called for every event to allow a modification of the stream processing context ”context(...)”. The functions are separated from each other by ”=>” which indicates the forwarding of the event stream to the next processing step.

As the new stream processing specifications are independent of the actual stream processing engine used to execute them, alternative mappings of the language to a processing system implementation can now be defined aside from the Drools based mapping. As such we are investigating a mapping the processing pipelines to a microservice-based architecture based on the Vert.x tool-kit.

 ISBN: 978-1-61208-702-3

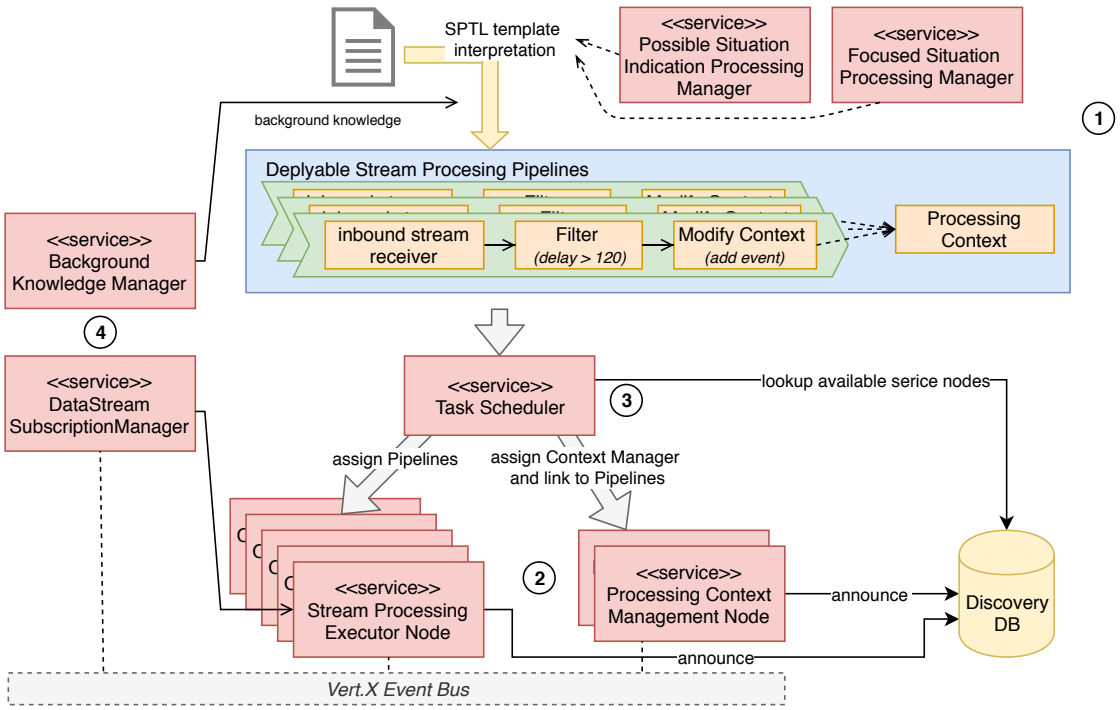

Figure 3. Overview of the stream processing pipeline assignment

```
IterationStreamProcessingBuilder {
  foreach $$focusArea as $$train {
    rule [DROOLS_TEMPLATE]
      when
        $a : DelayEvent( ) from entry-point "$${{train?
             delay}}"
      then
        CONTEXT.addToSet("$$trainEvents", $a  );
      end
      [/DROOLS_TEMPLATE] publishes no stream manipulates
           context;
  }
};
```

Figure 4. Stream Processing Builder definition in the old SPTL version

```
IterationStreamProcessing  {
  $$focusArea.delay : DelayEvent
    => filter( e.delay > 120 )
    => context( [M] $$trainEvents.add(e) [/M] );
}
```

Figure 5. Stream Processing definition in the extended version of the SPTL

V. Processing System Architecture

The overall processing system was subdivided into several components each with distinct functionality as discussed in [3]. For the communication between the components interfaces were defined, which, depending on the needed communication, are implemented as synchronous service-based interactions or asynchronous message based interactions.

However, in our initial architecture the stream processing itself was defined as one opaque component for the Phase 1 and as a second similar component for the Phase 3 processing with no further subdivision into smaller services. This initial design decision was caused by the use of one instance of the Drools Rule Engine for each of those components and the tight coupling of the SPTL to Drools. With the changes of the language as discussed in Section IV-A, the stream processing can now be subdivided into smaller sub-components based on the notion of defining a potentially distributed processing pipeline.

A. Mapping of Stream Processing Pipelines to a Microservice-oriented Architecture

For the processing system, such a processing pipeline consists of several separate stream processing statements where each statement takes a stream as input and potentially generates a stream as a result. Alternatively, a stream processing statement can also modify a so called processing context which is a shared data store in the context of the current situation's stream processing. Such pipeline definitions are the result of the interpretation of an SPTL template (Figure 3 Part 1).

In our microservices-oriented architecture, we map such pipelines to multiple small services where in the most fine grained form any stream processing operation could be provided by a separate service (Figure 3 Part 2). The services itself can then be distributed across several processing nodes in order to implement a distributed stream processing.

For the communication between the different microservices that form a processing pipeline the event bus mechanism provided by Vert.x will be used as it can act as a distributed peer-to-peer messaging system.

In order to deploy such a pipeline, we define a scheduler service. This scheduler has an overview over all available worker nodes which can execute stream processing tasks

 ISBN: 978-1-61208-702-3

(Figure 3 Part 3). This scheduler service is used by the two processing managers to assign their processing pipelines which they generated based on the given SPTL template. To allow the scheduler to find the available worker nodes, each node publishes itself as a service to a service registry, where the scheduler can thus find all available processing nodes.

Further services are provided that offer supporting facilities like an Event Stream Subscription Manager Service, allowing the pipeline nodes to request the needed event streams (Figure 3 Part 4). Moreover for the Phase 3 Stream processing, every situation specific processing requires a processing context that is shared between all stream processing pipelines associated with this situation. This processing context is again provided by a separate service assigned by the scheduler service (Figure 3 Part 2).

VI. Conclusion and Outlook

The paper outlines an extension of our service-based architecture towards the use of microservices for the distribution of the actual stream processing. The distribution is based on processing pipelines which we introduced by adapting the scenario template specification language. In our approach, the stream processing is realized by multiple microservices which together form a concrete stream processing pipeline potentially distributed across multiple machines. The actual distribution decision will be made by a scheduler service which oversees the available resources through their service registrations.

Currently, our model and architecture does not define any mechanisms for handling component failures during processing. We plan to add such a functionality in the form of an overseer service which monitors deployed pipelines, detects service failures and re-deploys the failed services. This however also requires an extension of the processing model itself so that a partial rollback of inconsistent processing state becomes possible, thus allowing the processing to resume in a defined state after a failure.

While the discussed language changes are already implemented, future work is the realization of the proposed microservice-based distribution as part of our prototype, thus, allowing for a detailed evaluation of the approach. In particular an evaluation of the performance of the Vert.x event bus in a distributed setup needs to be conducted as the later processing system will use this as its communication backbone and will thus rely on its performance.

Acknowledgments

Parts of the here presented work were done as part of the Eurostars Project E!7377 as well as Project 18014 funded by the Hasler Stiftung.

References

[1] G. Wilke, M. Schaaf, E. Bunn, T. Mikkola, R. Ryter, H. Wache, and S. G. Grivas, “Intelligent dynamic load management based on solar panel monitoring,” in Proceedings of the 3rd Conference on Smart Grids and Green IT Systems, 2014, pp. 76–81.

[2] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, “A Catalog of Stream Processing Optimizations,” ACM Comput. Surv., vol. 46, no. 4, mar 2014, pp. 46–1. [Online]. Available: {http://doi.acm.org/10.1145/2528412}

[3] M. Schaaf, “A service based architecture for situation-aware adaptive eventstream processing,” in The Tenth International Conference on Advanced Service Computing, Barcelona, Spain, February, 2018, pp. 40–44.

[4] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, “The Design of the Borealis Stream Processing Engine,” in In CIDR, 2005, pp. 277–289.

[5] Y. Xing, S. Zdonik, and J.-H. Hwang, “Dynamic Load Distribution in the Borealis Stream Processor,” in Proceedings of the 21st International Conference on Data Engineering, ser. ICDE ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 791–802.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. Zdonik, “Scalable distributed stream processing,” in In CIDR, vol. 3, 2003, pp. 257–268.

[7] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez, “StreamCloud: A Large Scale Data Streaming System,” in Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on, june 2010, pp. 126–137.

[8] S. Schneider, H. Andrade, B. Gedik, A. Biem, and K.-L. Wu, “Elastic scaling of data parallel operators in stream processing,” in Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, may 2009, pp. 1–12.

[9] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch, “Balancing load in stream processing with the cloud,” in Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops, ser. ICDEW ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 16–21. [Online]. Available: {http://dx.doi.org/10.1109/ICDEW.2011.5767653}

[10] “Apache Storm,” Online: https://storm.apache.org/, retrieved: March/04/19.

[11] “Apache Spark Streaming,” Online: https://spark.apache.org/streaming/, retrieved: March/13/19.

[12] “Vert.X Homepage,” Online: https://vertx.io/, retrieved: March/13/19.

[13] “Akka Homepage,” Online: https://akka.io/, retrieved: March/13/19.

[14] M. Schaaf, “Event processing with dynamically changing focus: Doctoral consortium paper,” in RCIS, ser. IEEE 7th International Conference on Research Challenges in Information Science, RCIS 2013, Paris, France, May 29-31, 2013, R. Wieringa, S. Nurcan, C. Rolland, and J.-L. Cavarero, Eds. IEEE, 2013, pp. 1–6.

[15] M. Schaaf, G. Wilke, T. Mikkola, E. Bunn, I. Hela, H. Wache, and S. G. Grivas, “Towards a timely root cause analysis for complex situations in large scale telecommunications networks,” Procedia Computer Science, vol. 60, 2015, pp. 160–169, knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings.

[16] M. Schaaf, “Situation aware adaptive event stream processing. a processing model and scenario definition language,” Ph.D. dissertation, Technical University Clausthal, 2017, verlag Dr. Hut, ISBN: 978-3-8439-3376-6.

[17] “Drools Business Rules Management System,” Online: http://www.drools.org/, retrieved: March/13/19.

[18] “MVEL Language Guide for 2.0,” Online: http://mvel.documentnode.com/, retrieved: March/13/19.

[19] T. W. S. W. Group, “SPARQL 1.1 Overview,” Tech. Rep., March 2013, retrieved: 13.01.18. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/

 ISBN: 978-1-61208-702-3