HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Service Computation 2020
October 25 – 29, 2020 – Nice, France

Keep it in Sync!
Consistency Approaches for Microservices
- An Insurance Case Study -

A. Koschel, A. Hausotter (presenter)
M. Lange, S. Gottwald

Faculty of Business and Computer Science
University of Applied Sciences and Arts, Hannover
Ricklinger Stadtweg 120
30459 Hannover
{arne.koschel | andreas.hausotter}@hs-hannover.de

Presenter

Dr. ANDREAS HAUSOTTER is a professor emeritus for distributed information systems and database systems at the University for Applied Sciences and Arts, Hanover, Germany, Faculty of Business and Computer Science. His area of specialization comprises service computing – including service-oriented Architectures (SOA) and microservices – Java EE, webservices, distributed information systems, business process management, business rules management, and information modeling.

In 1979 he received his PhD in mathematics at Kiel University, Faculty of Mathematics and Natural Sciences. After graduation he started his career with KRUPP ATLAS ELEKTRONIK, Bremen, as a systems analyst and systems programmer in the area of real time processing. In 1984 he was hired as systems engineer and group manager SNA Communications for NIXDORF COMPUTER, Paderborn. After that, he worked for HAAS CONSULT, Hanover, as a systems engineer and product manager for traffic guidance systems.

In 1996 he was appointed professor of operating systems, networking and database systems at the University of Applied Sciences and Arts, Hanover. He has been retired since March 2018.

From the beginning he was involved in several research projects in cooperation with industry partners. During his research semester he developed a Java EE / EJB application framework. Based on this framework a web-based simulation software for securities trading was implemented by his research group to train the apprentices of the industry partner.

In 2005, the Competence Center IT & Management (CC_ITM) was founded in cooperation with industry partners. Different ambitious research projects have since then been carried out in the context of service-computing, microservices, cloud computing, business process management, and business rules management.

CC_ITM

CC_ITM Competence Center Information Technology and Management Institute at the University of Applied Sciences and Arts Hannover | cc_itm@hs-hannover.de

HOCHSCHULE HANNOVER
UNIVERSITY OF APPLIED SCIENCES AND ARTS
Fakultät IV Wirtschaft und Informatik

- **Competence Center Information Technology & Management (CC_ITM)**
  - Institute at the University of Applied Sciences and Arts, Hannover
  - Founded in 2005 by colleagues from the departments of **Business Information Systems and Computer Science**
  - Members: **Faculty staff**, **industry partners** (practitioners) of different areas of businesses
- Main objective
  - **Knowledge transfer** between university and industry
- Research topics
  - Management of information processing
  - **Service computing,** including Microservices, Service-oriented Architectures (SOA), Business Process Management (BPM), Business Rules Management (BRM)
  - Cloud Computing

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- **Introduction**
- **Background**
- **Shaping the Microservice Architecture**
- **Consistency Assurance in the Microservice Architecture**
- **Conclusion & Future Work**
- **References**

Motivation

- **Microservices**
  - **Architectural style** for complex application systems
  - Software is split into lightweight, independently deployable components.
  - Each component may use different technologies.
  - Components communicate over standardized network protocols.
- **Potential** and **benefits** of the architectural style
  - Maintainability, scalability, fault tolerance, ...
  - → **Attracted attention** in the software development industry
- **Challenges** of the architectural style
  - Data often persisted redundantly to provide fault tolerance
  - **Synchronization** of those data to provide consistency may be an issue
  - → **Hampers the adoption**

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Goal & Major Contribution

- Case study from the insurance industry
  - **Migration of a monolithic core application** towards microservices
  - Focuses on consistency issues.
- Model-driven design
  - Based on **Domain Driven Design (DDD)**
    - Bounded Contexts & Domain Models
  - Consideration of **compliance requirements**.
- Consistency
  - General **approaches for synchronizing redundant data** in microservices
    - Trade-off: loose coupling vs. level of consistency
  - **Best practice** for synchronizing data in the migrated core application.
- Overall Goal
  - **Examine the suitability of microservice architecture for the insurance industry.**

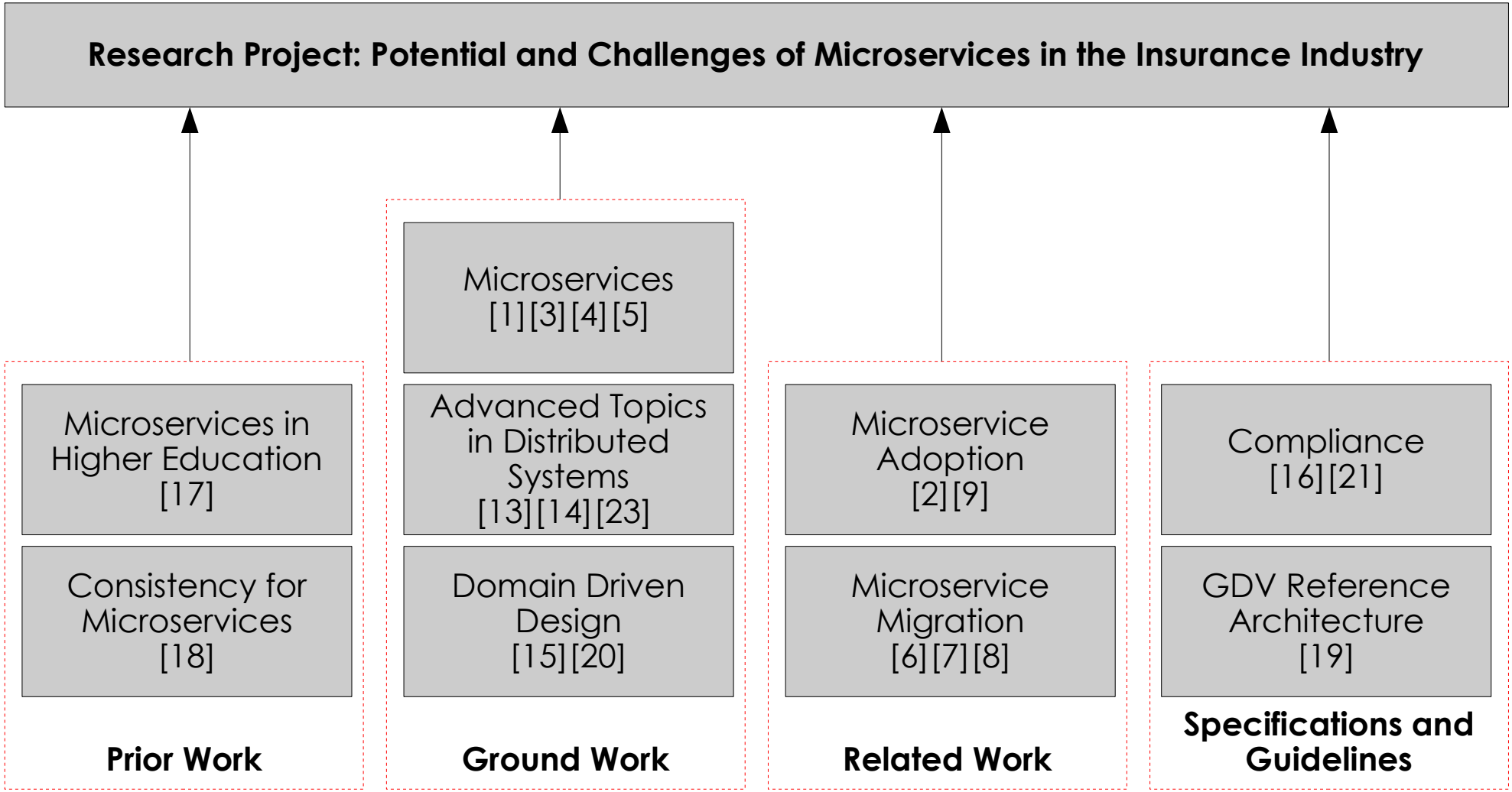
HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- Introduction
- Background
- Shaping the Microservice Architecture
- Consistency Assurance in the Microservice Architecture
- Conclusion & Future Work
- References

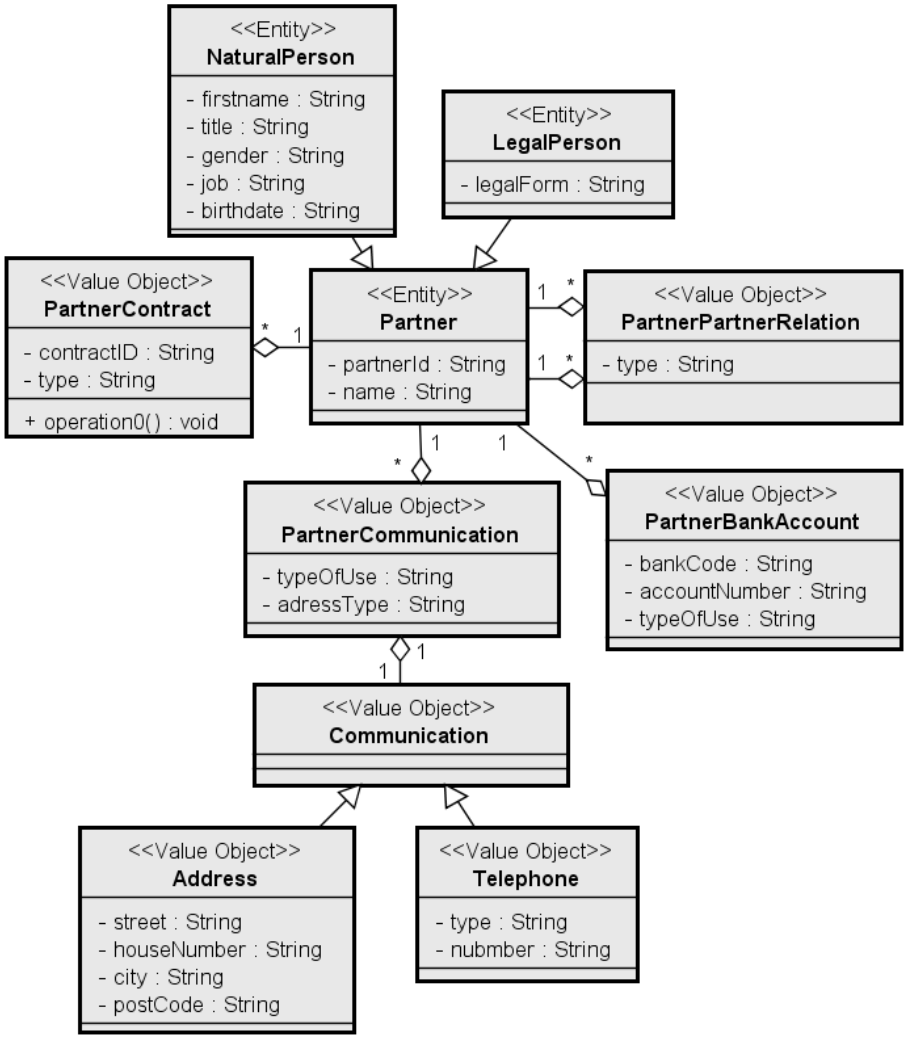
HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Related & Prior Work



Partner Management System

- Overview
  - Core application for **managing partners** of an insurance company
  - Based on the **Reference Architecture for German Insurance Companies (VAA)**
  - Basically a **CRUD** application.
- Challenges
  - Implemented as a **single deployment unit**
  - Heavily **changing load profile**
  - **Poor flexibility, scalability** and **fault tolerance.**



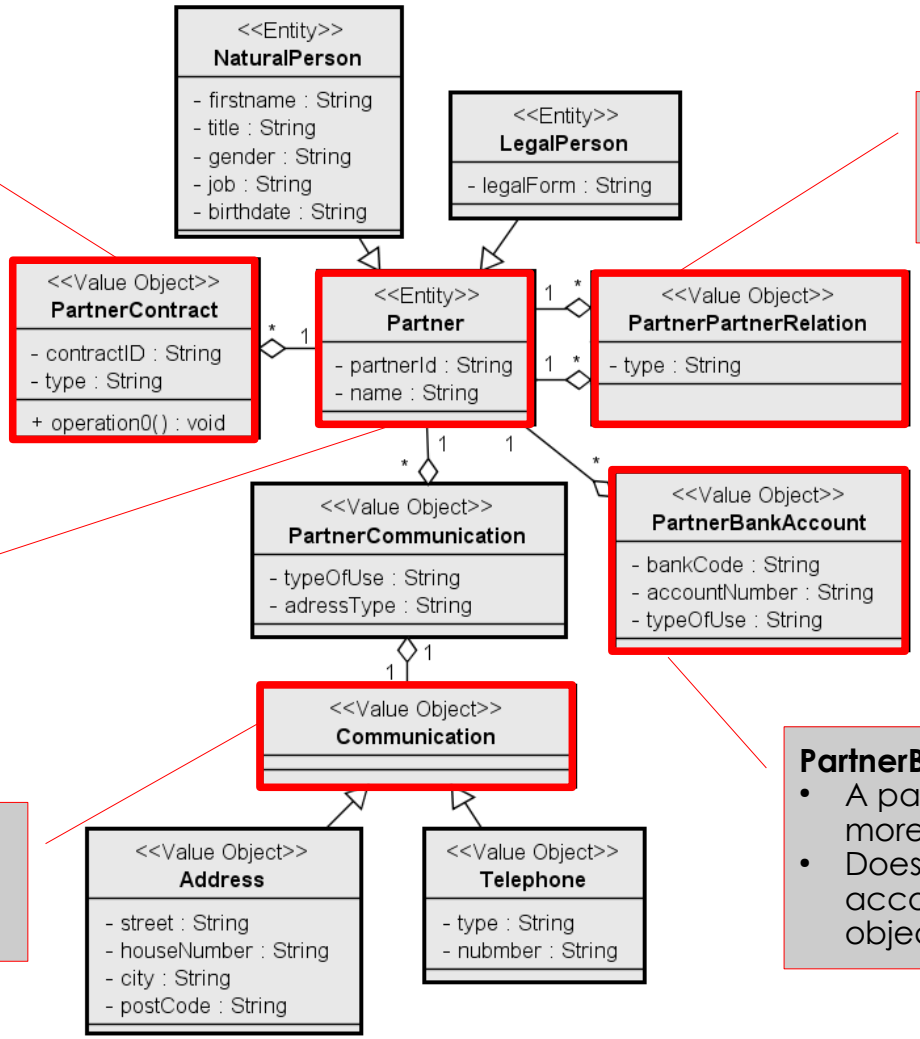
HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Partner Management System

PartnerContract
- Usually, a partner enters one ore more contracts with the insurance company.
- Does not represent the contract itself, but its proxy object.

Partner
- Natural (eg. clients, appraisers, lawyers) or legal (eg. other insurance companies) person
- Is in relation to the insurance company.

Communication
- The company may communicate with the partner via mail and/or phone



PartnerPartnerRelation
- A partner may have relations to other partners.

PartnerBankAccount
- A partner must have one or more bank accounts.
- Does not represent the bank account itself, but its proxy object.

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- Introduction
- Background
- Shaping the Microservice Architecture
- Consistency Assurance in the Microservice Architecture
- Conclusion & Future Work
- References

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Domain Driven Design (DDD) at a Glance

- Domain Driven Design
  - Approach to Software Development (Evans, 2004)
  - Capturing relevant domain knowledge in **Domain Models**
  - **Collaborative modeling** of domain experts and software engineers.
- Basic concepts of DDD
  - **Bounded Context** – Description of a boundary within which a particular model is defined and applicable
  - **Domain Model** – System of abstractions, describes and relevant aspects of the domain.
- **DDD patterns**
  - Refine the structural domain models for **Model-driven Design**.
- **Strategic patterns** of interactions
  - Manage the trade-off between **loose coupling** between bounded contexts and the **communication needs** between development teams.

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Bounded Contexts – Design Objectives

- **Self-contained** and highly **cohesive**
  - Ideally, modifications and adjustments effect just one bounded context / microservice.
    → Promotes a better manageability of the business complexity.
- Efficient **scalability**
  - Run multiple instances of the same microservice.
    → Allows the system to adapt to the changing load during the day.
- **Compliance – IT alignment**
  - Insurance companies are considered as "critical infrastructure", i.e. they are essential for society and economy.
  - Application systems in the insurance industry have to strictly comply with compliance requirements, eg.
    - "Supervisory Requirements for IT in Insurance Undertakings (VAIT)"
    - General Data Protection Regulation (GDPR).

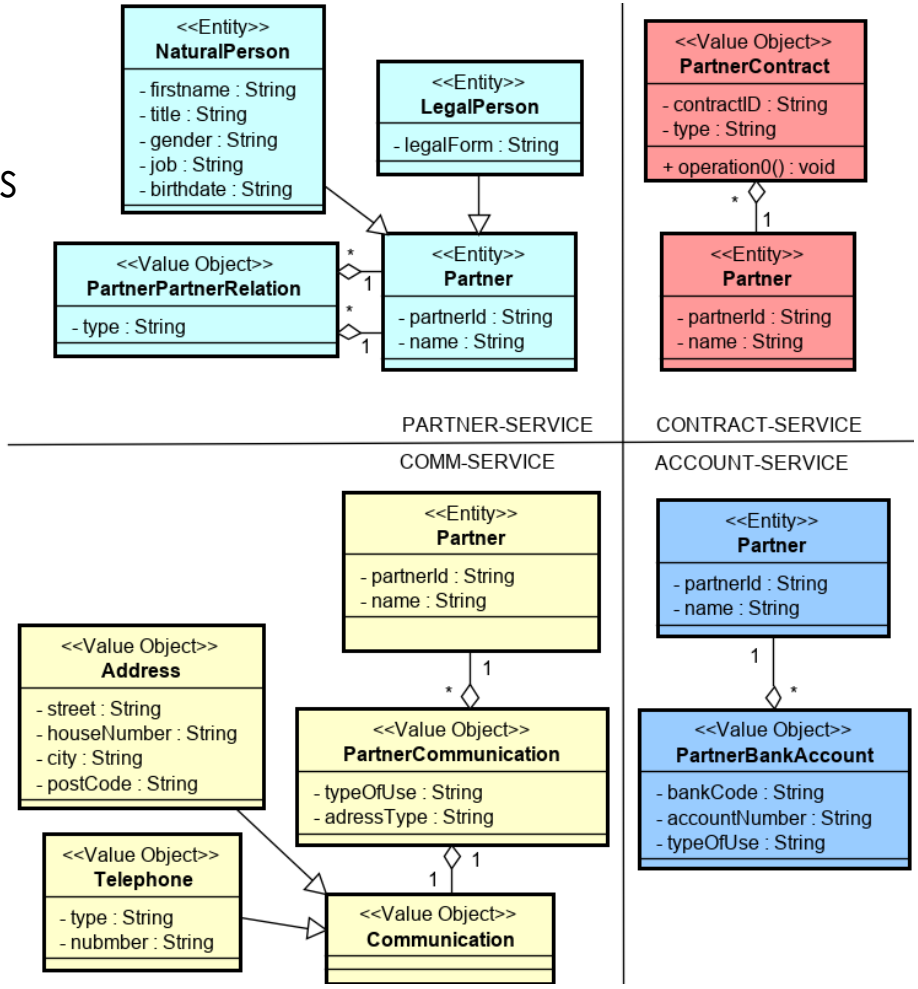
Bounded Contexts – Compliance – IT alignment

- Modeling of the bounded contexts promotes different **protection levels**.
  - `Partner Management System` only processes data belonging to protection levels A up to C.

Protection Level	Personal Data ...	Examples	Degree of Damage
A	… which have been made freely available by the persons concerned.	Data visible in the phone book.	minor
B	… whose improper handling is not expected to cause particular harm, but which has not been made freely accessible by the person concerned.	Restricted public files.	minor
C	… whose improper handling could damage the person concerned in his social position or economic circumstances (“reputation”).	Income, property tax, administrative offenses.	manage-able
D	… whose improper handling could significantly effect the social position or economic circumstances of the person concerned (“existence”).	Criminal offenses, employment evaluations, health data.	substantial
E	… whose improper handling could impair the health, life of freedom of the person concerned.	Data on persons who my be victims of a criminal offense.	major

Bounded Contexts & Domain Models

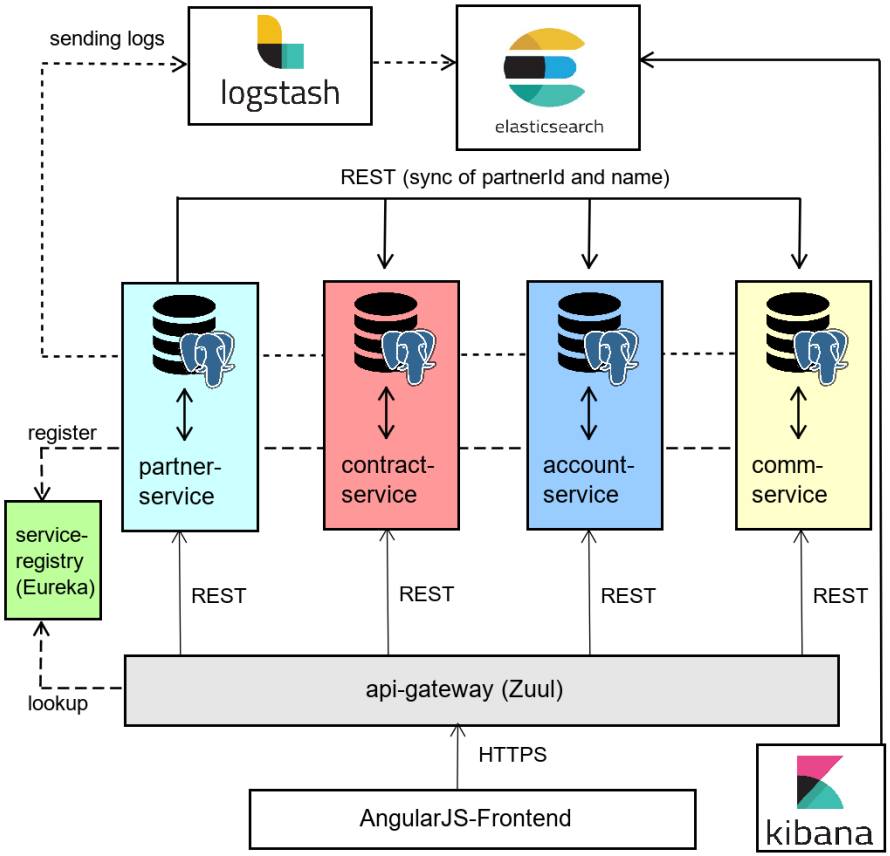
- **Bounded contexts**
  - Modeled by the insurance company's domain experts and software engineers using event storming
    - `Partner, Contract, Communications, Account.`
- **DDD patterns**
  - Entities – model objects with identity
  - Value Objects – model values without an identity (such as proxy objects).
- **Strategic patterns** of interaction
  - Anticorruption Layer – each Bounded Context has a different understanding of the entity `Partner`.



- In our case, each Bounded Context is mapped to exactly one microservice.

Technical Microservice Architecture

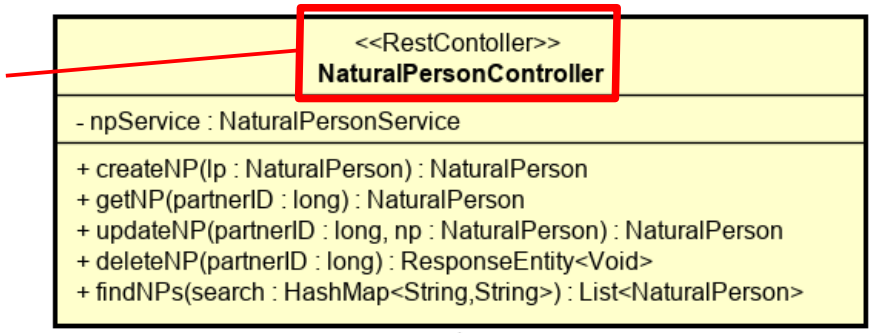
- Microservices are implemented as **REST Web Services** based on the **Spring framework.**
- Persistency
  - Each microservice has its **own data management** using PostgreSQL DB.
  - Partner **data kept in sync** across all microservices by the `partner-service` using REST calls.
- Infrastructure and technical services
  - AngularJS: Single page front end
  - Netflix OSS stack: Service discovery, API gateway
  - ELK (Elasticsearch, Logstash, Kibana) stack: monitoring & logging.



- All components are deployed in separate **Docker** containers.

Design of the Microservices – `partner-service`

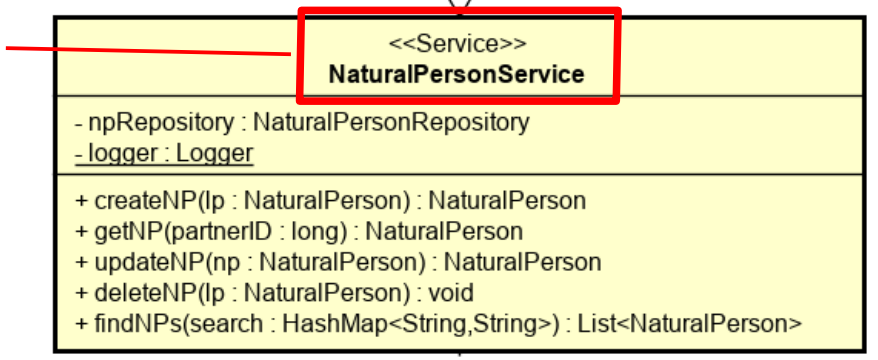
- **Annotation** `@RestController` forces the Spring Framework to create an instance of this class.
- Gets an instance of the `NaturalPersonService` **injected**.
- The controller **delegates** incoming **HTTP requests** to the `NaturalPersonService`.



Routing Layer

- Interacts with the service's consumers.
- **HTTP facade** for the business layer.

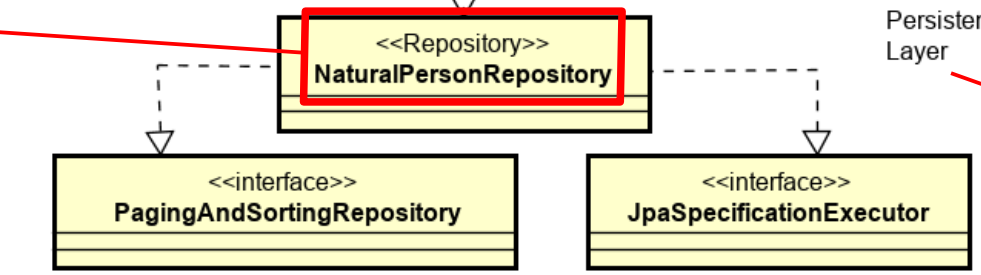
- **Annotation** `@Service` forces the Spring Framework to manage an instance of this class.
- Implements the business logic of the `partner-service`.
- Gets an instance of the `NaturalPersonRepository` injected.



Business Logic Layer

Provides the service's **business logic**.

- **Annotation** `@Repository` forces the Spring Framework to use **Spring Data JPA** as an OR mapping framework.
- Managed entity classes are annotated with `@Entity`.

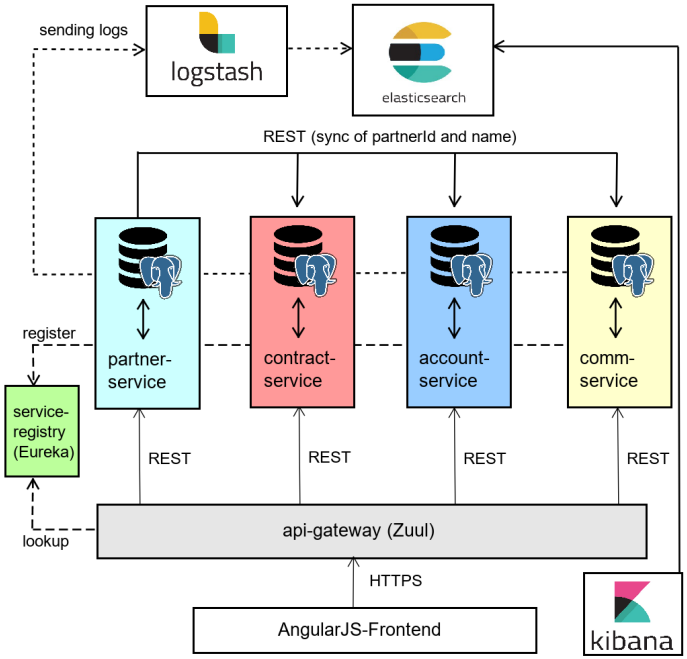


Persistence Layer

Provides **object-relational (OR) mapping** features.

Evaluation of the Microservice Architecture

- Benefits
  - **Scalability**, i.e. run multiple instances of the same microservice
    → System can adapt to changing loads.
  - **Robustness** & **fault tolerance**, i.e. Partner data are kept redundant.
    → Services can resolve key relationships to partner data even if the `partner-service` is unavailable.



- Challenges
  - Distributed **Monitoring** and **Logging**
    → Provided by the ELK (Elasticsearch, Logstash, Kibana) stack
  - **Consistency assurance**
    → As partner data are kept redundant, they must be synchronized across all services.

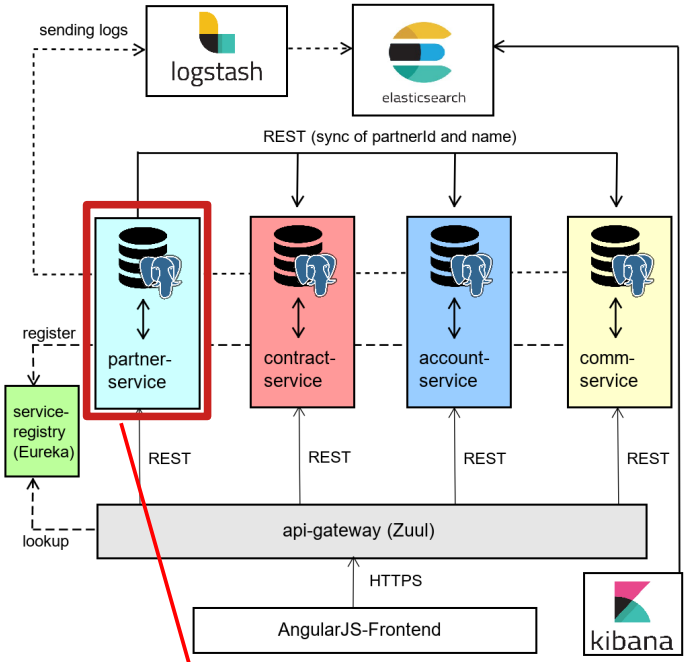
Evaluation of the Microservice Architecture

- Drawbacks of the implemented synchronization method
  - **Fault tolerance**: If the `partner-service` is temporarily unavailable, other services are not notified about changes.

    → Effects **consistency** across the services
  - **Synchronicity**: In case of changes of partner data, a thread of the `partner-service` is in a blocked state until all services have been notified.

    → Effects **response time** of the `partner-services`.
  - **Extensibility**: `partner-Service` keeps a list of services that have to be notified on changes.

    → A static list requires the system to be **redeployed** after modifications.



Synchronization method: Partner data are kept in sync across all services by the `partner-service` using synchronous REST calls.

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- Introduction
- Background
- Shaping the Microservice Architecture
- **Consistency Assurance in the Microservice Architecture**
- Conclusion & Future Work
- References

General Approaches to Ensuring Consistency

- **Synchronous distribution**
  - Approach
    - Owner of the data propagates any change to all interested services.
    - A service registry may be used to keep the information which services are interested in which data.
  - Pros
    - Provides a **high degree of consistency**.
  - Cons
    - Notifying a large number of services may **effect load** and **response time.**

General Approaches to Ensuring Consistency

- **Polling**
  - Approach
    - The responsibility of synchronization is shifted to the interested services themselves.
    - Services ask for new data using an interface provided by the service containing the master data.
  - Pros
    - The size of the **inconsistency window can be controlled** by each service independently.
  - Cons
    - The frame in which the data differs depends on the polling interval.

      → Provides **eventual consistency**

General Approaches to Ensuring Consistency

- **Publish – Subscribe**
  - Approach
    - Following the Publish-subscribe pattern, an event is published to a topic on every change of the master data.
    - Interested services subscribe to this topic, receive events and update their own data accordingly.
  - Pros
    - The system is **robust against failures,** if the underlaying messaging oriented middleware persists events.
      → Approach is suitable for the resilient and lightweight nature of microservices.
  - Cons
    - Data are inconsistent until the event is delivered and processed.
      → Provides **eventual consistency**

General Approaches to Ensuring Consistency

- **Event sourcing**
  - Approach
    - Upon any change in master data an event, that represents the state change, is published.
    - The sequence of all events is persisted in an append-only event store.
    - Interested services can recreate the current state by replaying the events.
  - Pros
    - **High degree of consistency**
  - Cons
    - Centralized event store **weakens loose coupling**.
    - Generating the current data from the sequence is a challenge.

Best Practices for Synchronizing Partner Data

HOCHSCHULE HANNOVER
UNIVERSITY OF APPLIED SCIENCES AND ARTS
Fakultät IV Wirtschaft und Informatik

- Initial situation
  - As shown above, there are **different approaches** to manage consistency.
  - Each approach has its own **advantages** and **disadvantages**.
  - Above all, there are **trade-offs** between
    - Level of **consistency**
    - Microservice **design principles**, i.e. loose coupling and decentralized data management.
- Goal
  - Find the **most suitable level** of **consistency**.
- Method
  - Specify possible **inconsistent states** of the data.
  - Combine them with typical **use cases** of your system.

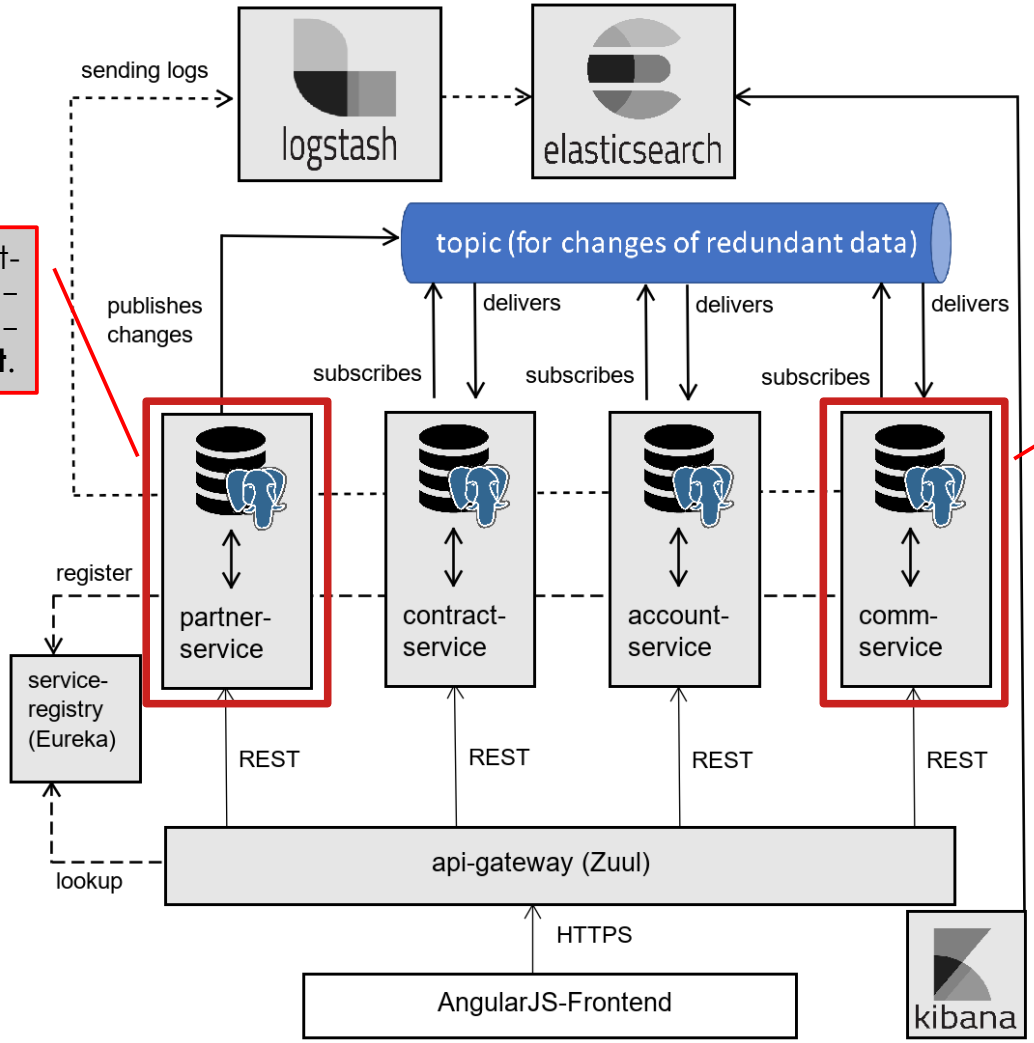
Best Practices for Synchronizing Partner Data

- **Redundant data**
  - `Name, partnerID`
- **Inconsistent states**, due to CRUD operations
  - New partner ist not yet present in the system.
  - `Name` and / or `partnerID` is not up-to-date.
  - Deleted partner is not yet deleted everywhere.
- Typical **use cases**
  - Sending a letter via mail
  - Conclusion of an insurance contract.
- Result
  - The system is robust against inconsistent states.
  - Reason: Business processes are designed resilient against delays and errors.

  → **Publish-Subscribe** approach is the most suitable solution.

Technical Microservice Architecture – Update

On any **change** of partner data (`name`, `partnerID`) `partner-service` **publishes an event**.



- A service which is interested in any change, **subscribes on this topic**.
- If the services **receives an event** it can **update** their own data accordingly.

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- Introduction
- Background
- Shaping the Microservice Architecture
- Consistency Assurance in the Microservice Architecture
- Conclusion & Future Work
- References

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Conclusion & Future Work

- Conclusion
  - Our paper presents **case study** on microservices from the insurance industry.
    - Insights are given in the process to **migrate** a monolithic **core application** towards a modern **microservice architecture** including ...
    - … design, implementation, and special topics such as compliance.
  - **Synchronization** of redundant data across microservices is key issue.
    - Four **approaches** to manage the consistency are pointed out.
    - A **best practice** to identify the most suitable approach was designed and implemented.
- Future work
  - **Tests** of the microservice architecture **under real-world conditions**
  - Integration of tests in a **CI/CD pipeline**
  - Application of our findings to a more **sophisticated example**.

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Agenda

- Introduction
- Background
- Shaping the Microservice Architecture
- Consistency Assurance in the Microservice Architecture
- Conclusion & Future Work
- References

References

[1] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," https://martinfowler.com/articles/microservices.html, March 2014, [retrieved: 05, 2020].

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

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

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

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

[6] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, "Microservices migration patterns," Software: Practice and Experience, vol. 48, no. 11, 2018, pp. 2019–2042.

References

[7] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," IEEE Software, vol. 33, no. 3, 2016, pp. 42–52.

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

[9] 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.

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

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

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

References

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

[14] M. Fowler, "Event Sourcing," https://martinfowler.com/eaaDev/EventSourcing.html, December 2005, [retrieved: 05, 2020].

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

[16] "Versicherungsaufsichtliche Anforderungen an die IT (VAIT) (2019) vom 20.03.2019,"https://www.bafin.de/SharedDocs/Downloads/DE/Rundschreiben/dl rs 1810 vait va.html, March 2019, [retrieved: 05, 2020].

[17] M. Lange, A. Hausotter, and A. Koschel, "Microservices in Higher Education - Migrating a Legacy Insurance Core Application," in 2nd International Con-ference on Microservices (Microservices 2019), Dortmund, Germany, 2019, https://www.confmicro.services/2019/papers/Microservices 2019 paper 8.pdf, [retrieved: 05, 2020].

References

[18] A. Koschel, A. Hausotter, M. Lange, and P. Howeihe, "Consistency for Microservices - A Legacy Insurance Core Application Migration Example," in SERVICE COMPUTATION 2019, The Eleventh International Conference on Advanced Service Computing, Venice, Italy, 2019, https://www.thinkmind.org/index.php?view=article&articleid=service computation 2019 1 10 18001, [retrieved: 05, 2020].

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

[20] V. Vernon, Implementing domain-driven design. Addison-Wesley, 2013.

[21] "Schutzstufenkonzept des LfD Niedersachsen," https://www.lfd.niedersachsen.de/technik und organisation/schutzstufen/schutzstufen-56140.html, October 2018, [retrieved: 05, 2020].

References

[22] A. Roland, "Secrecy, technology, and war: Greek fire and the defense of byzantium, 678-1204," Technology and Culture, vol. 33, no. 4, 1992, pp. 655–679.

[23] 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, [retrieved: 05, 2020].