DOMAIN-DRIVEN SERVICE IDENTIFICATION AND DESIGN WITH MICROSERVICE API PATTERNS (MAP)

Keynote 1
2nd VSS Seminar (DevOps and Microservices APIs)

Vienna, August 28, 2019

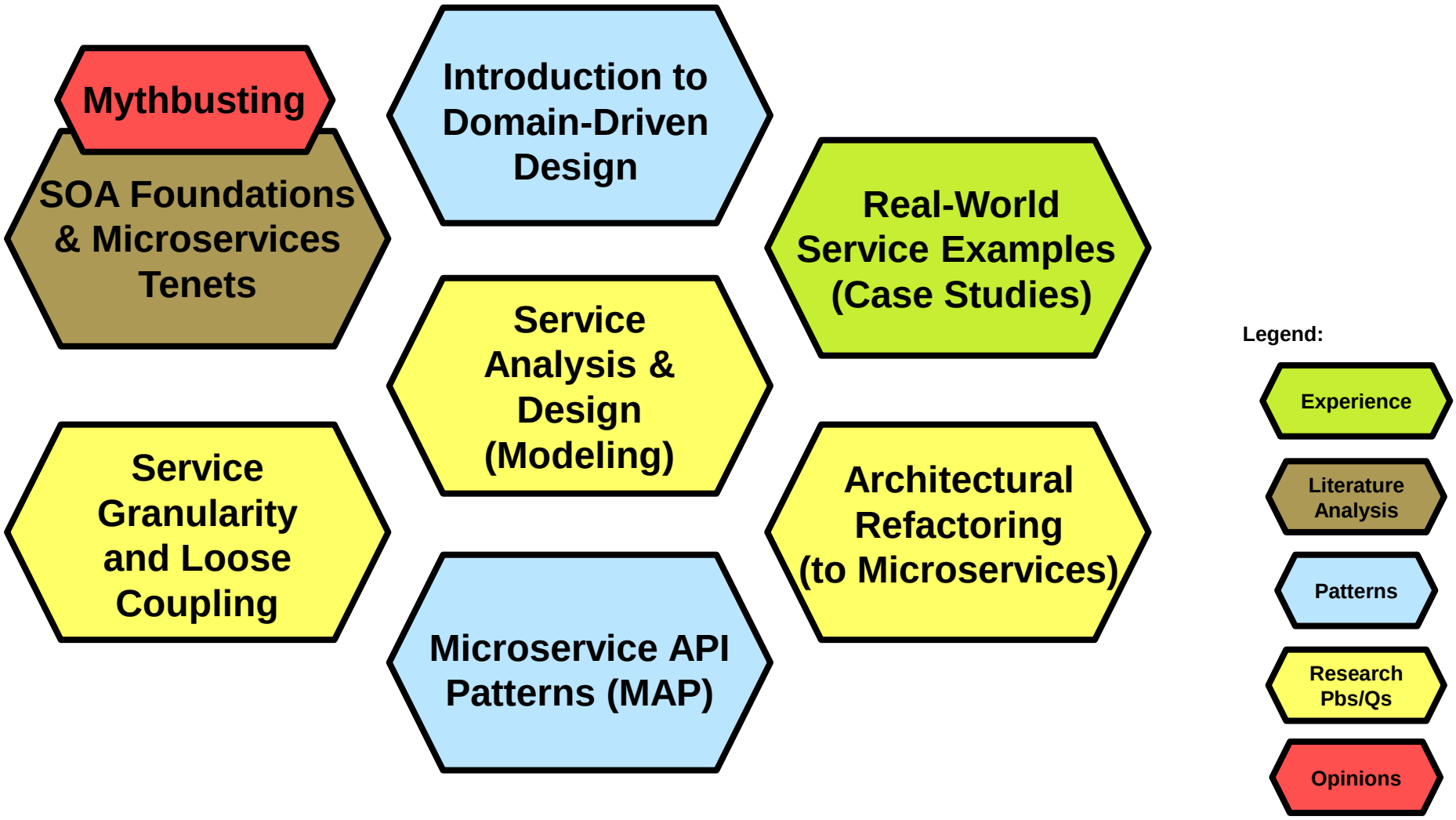
Prof. Dr. Olaf Zimmermann (ZIO)
Certified Distinguished (Chief/Lead) IT Architect
Institute für Software, HSR FHO
ozimmerm@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

ZIOs “Research” Objectives (Projects and this Presentation)

1. **Share Architectural Knowledge (AK) such as Arch. Decisions and patterns for state-of-the-art architectural styles and technologies.**
   - *Focus here (partially published):* Microservices API Patterns (MAP)
   - *Halted:* Cloud Computing Decision Model
   - *Completed previous work:* SOA Decision Modeling (2006-2011)
2. **Make AK Management (AKM) practical and assist architects with tools.**
   - *Here (quick overview):* Context Mapper, Service Cutter
   - *Also available:* AD Mentor, MADR (open source projects)
   - *Emerging:* Microservices Domain-Specific Language (MDSL)
3. **Help bridge the gap between agile practices and software architecture along the way.**
   - *Here (quick overview):* architectural refactoring
   - *Another time:* SMART quality attributes, Y-shaped architectural decision records (in Markdown, code)

Available “Micropresentations” (Breakout Session Topics?)



Agenda Today

1. **Microservices Clarification and Level Setting**
   - It's the API contract that matters (not or not only the middleware)
2. **Microservice API Patterns (MAP)**
   - Motivation, publication status, website, examples; tools
3. **Open Research Questions**
   - … and early/partial solutions to them

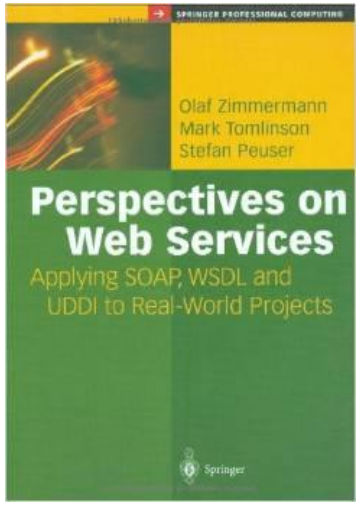
https://vss.swa.univie.ac.at/2019/

Agenda

1. ***Microservices Clarification and Level Setting***
   - *It's the API contract that matters (not or not only the middleware)*
2. **Microservice API Patterns (MAP)**
   - Motivation, publication status, website, examples; tools
3. **Open Research Questions**
   - … and early/partial solutions to them

https://vss.swa.univie.ac.at/2019/

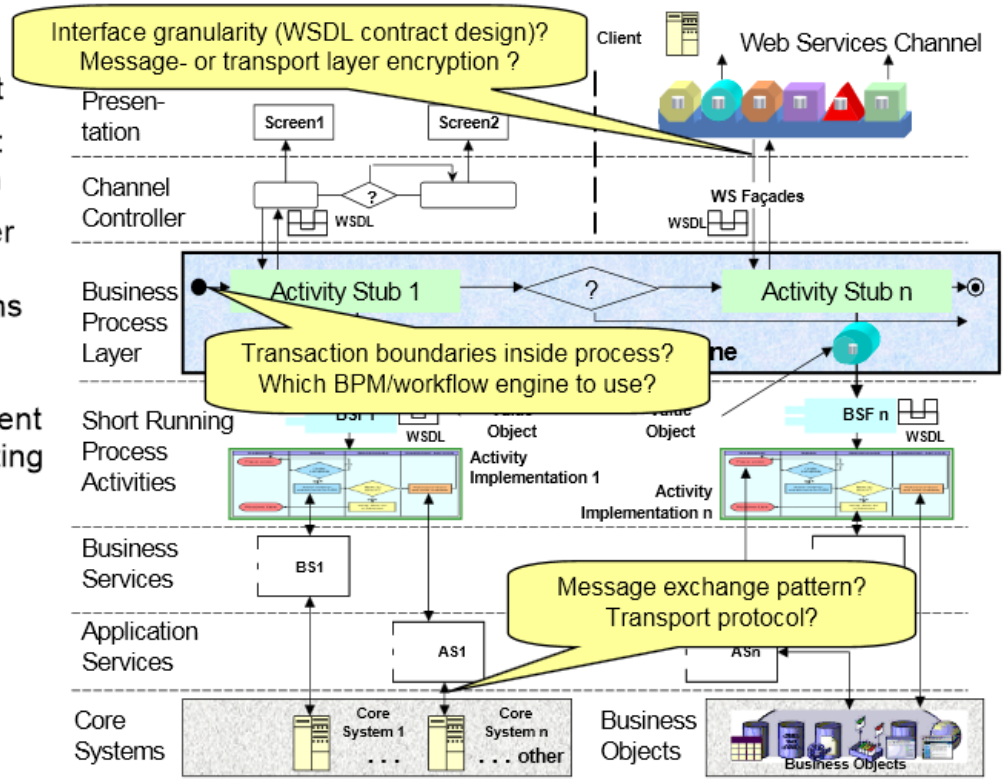
Sample Project: Order Management Application (Telecommunications)



Multi-Channel Order Management SOA in the Telecommunications Industry (in production since Q1/2005) [OOPSLA 2005]

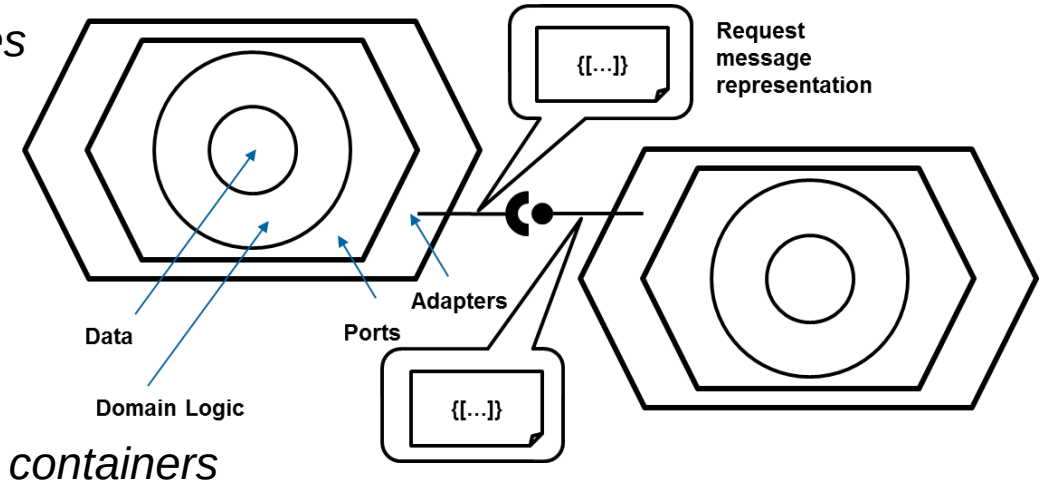
Reference: IBM, ECOWS 2007

- **Functional domain**
  - Order entry management
  - Two business processes: new customer, relocation
  - Main SOA drivers: deeper automation grade, share services between domains
- **Service design**
  - Top-down from requirement and bottom-up from existing wholesaler systems
  - Recurring architectural decisions:
    - Protocol choices
    - Transactionality
    - Security policies
    - Interface granularity



A Consolidated Definition of Microservices (“Hexagonions”)

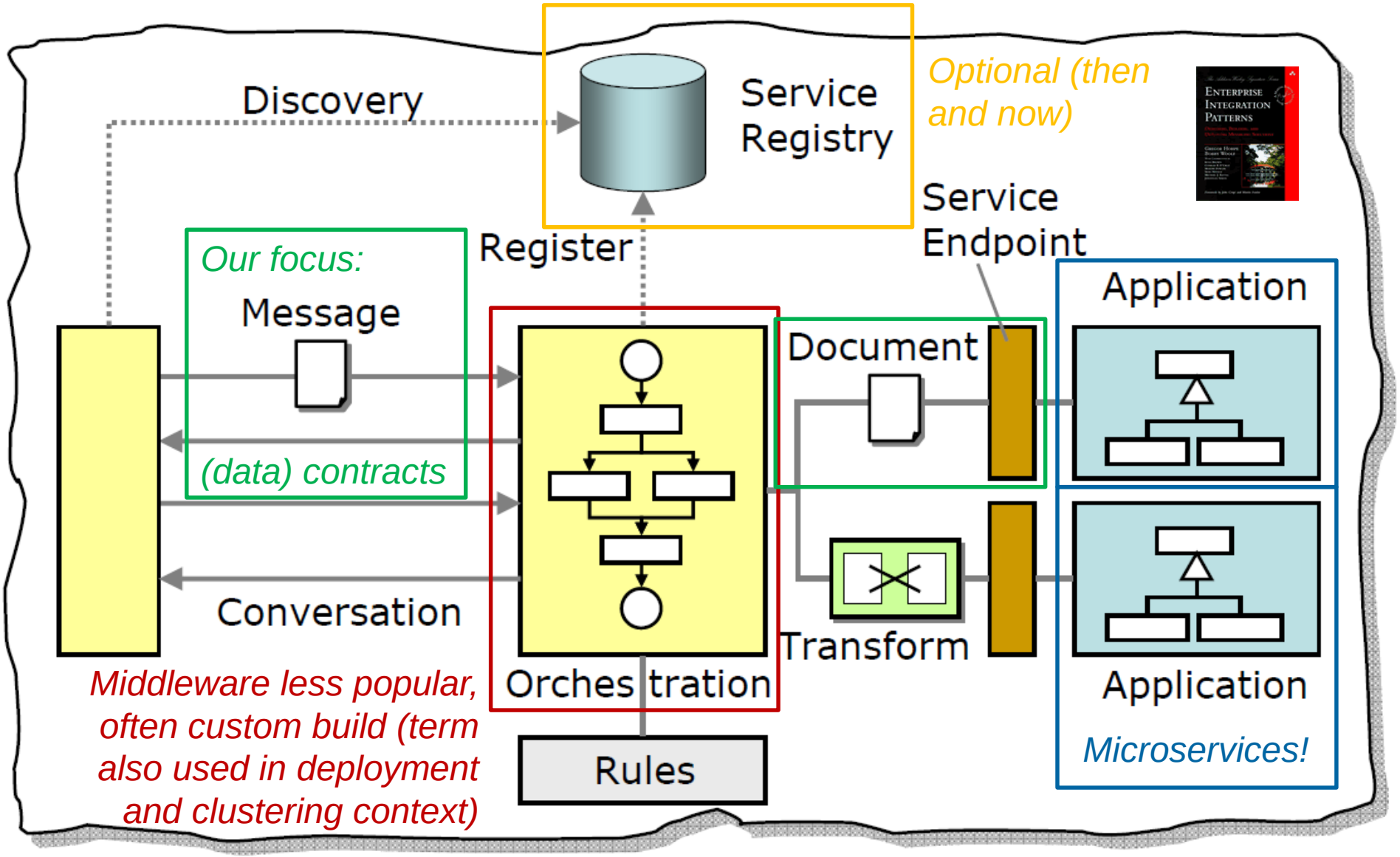
- **Microservices architectures evolved from previous incarnations of Service-Oriented Architectures (SOAs):**
  - *Independently deployable, scalable and changeable* services, each having a single responsibility
  - Modeling *business capabilities*



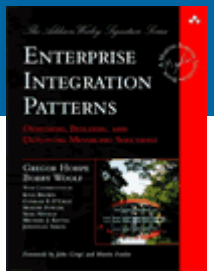
More information: Zimmermann, O., *Microservices Tenets: Agile Approach to Service Development and Deployment*, Springer Journal of Computer Science Research and Development (2017)

  - Often deployed in *lightweight containers*
  - Encapsulating their *own state*, and communicating via *message-based remote APIs* (HTTP, queueing) in a loosely coupled fashion
  - Leveraging *polyglot programming and persistence*
  - DevOps practices including decentralized *continuous delivery and end-to-end monitoring* (for business agility and domain observability)

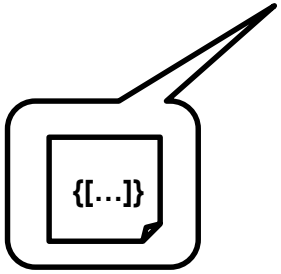
"Napkin Sketch" of SOA Realizations (Adopted from G. Hohpe)



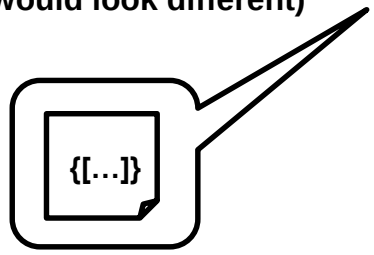
Calls to Service Operations are EIP-style Messages




```
curl -X GET "http://localhost:8080/customers/rgpp0wkpec" -H "accept: */*"
```



Sample request message (note: PUTs and POSTs would look different)

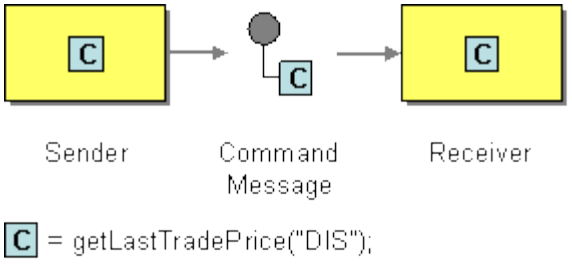


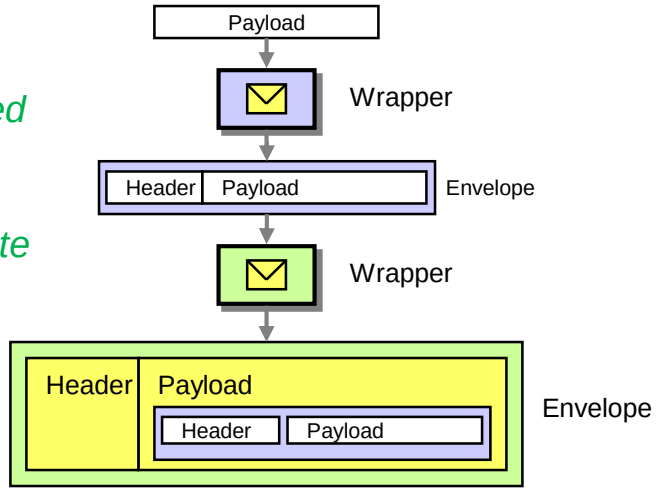
Response message structure

```
{
  "_links": [
    {
      "deprecation": "string",
      "href": "string",
      "hreflang": "string",
      "media": "string",
      "rel": "string",
      "templated": true,
      "title": "string",
      "type": "string"
    }
  ],
  "birthday": "2019-02-12T09:10:07.370Z",
  "city": "string",
  "customerId": "string",
  "email": "string",
  "firstname": "string",
  "lastname": "string",
  "moveHistory": [
    {
      "city": "string",
      "postalCode": "string",
      "streetAddress": "string"
    }
  ],
  "phoneNumber": "string",
  "postalCode": "string",
  "streetAddress": "string"
}
```

+/-?

Embed nested entity data?
or
Link to sparate iresource?





{[…]} -- some JSON (or other MIME type)

https://www.enterpriseintegrationpatterns.com/patterns/messaging/CommandMessage.html

Teaser Question (*not* from AppArch Lecture Exam at HSR FHO)

- **You have been tasked to develop a RESTful HTTP API for a master data management system that stores customer records and allows sales staff to report and analyze customer behavior. The system is implemented in Java and Spring. An additional access channel is RabbitMQ.**
- **What do you do?**

a) *I hand over to my software engineers and students because all I manage to do these days is attend meetings and write funding proposals.*

b) *I annotate the existing Java interfaces with @POST and @GET, as defined in Spring MVC, JAX-RS etc. and the job is done.*

c) *I install an API gateway product in Kubernetes and hire a sys admin, done.*

d) *I design a service layer and write an Open API Specification (f.k.a. Swagger) contract as well as Data Transfer Objects (DTOs). I worry about message sizes, transaction boundaries, compensation and coupling while implementing the contract. To resolve such issues, I create my own novel solutions. Writing infrastructure code and frameworks is fun after all!*

e) _____ *?*

Agenda

1. **Microservices Clarification and Level Setting**
   - It's the API contract that matters (not or not only the middleware)
2. ***Microservice API Patterns (MAP)***
   - *Motivation, publication status, website, examples; tools*
3. **Open Research Questions**
   - … and early/partial solutions to them

https://vss.swa.univie.ac.at/2019/

How to find suited granularities and achieve loose coupling?

Context

We have decided to go the SOA and/or microservices way. We use DDD for domain modeling and agile practices for requirements elicitation.

Problems (Industry, Academia)

How to identify an adequate number of API endpoints and operations?

How to design (command/document) message representation structures so that API clients and API providers are loosely coupled and meet their (non-)functional requirements IDEALly?

Which patterns, principles, and practices do you use? Do they work?

Microservice API Patterns (MAP)

Introducing… Microservice API Patterns (MAP)

europlop

- **Identification Patterns:**
  - DDD as one practice to find candidate endpoints and operations

Foundation Patterns

- What type of (sub-)systems and components are integrated?
- Where should an API be accessible from?
- How should it be documented?

Structure Patterns

- What is an adequate number of representation elements for request and response messages?
- How are these elements structured?
- How can they be grouped and annotated with usage information?

READ MORE →

Quality Patterns

- How can an API provider achieve a certain level of quality of the offered API, while at the same time using its available resources in a cost-effective way?
- How can the quality tradeoffs be communicated and accounted for?

READ MORE →

Responsibility Patterns

- Which is the architectural role played by each API endpoint and its operations?
- How do these roles and the resulting responsibilities impact (micro-)service size and granularity?

READ MORE →

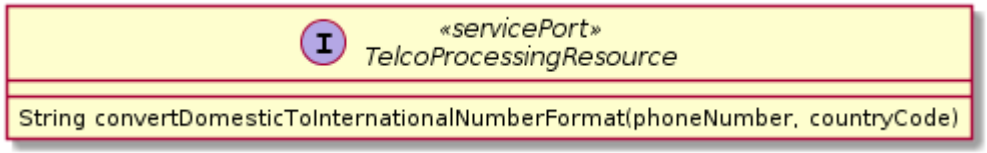
- **Evolution Patterns:**
  - Recently workshopped (EuroPLoP 2019)

http://microservice-api-patterns.org

Microservice API Patterns (MAP)

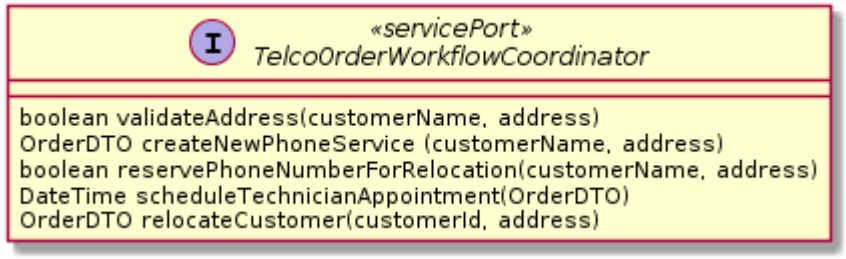
Exemplary Services in Order Management (Telecomunications)

Computation Function: **no read, no write**



Event Processor: **write only (append)**

Retrieval Operations: **read only**

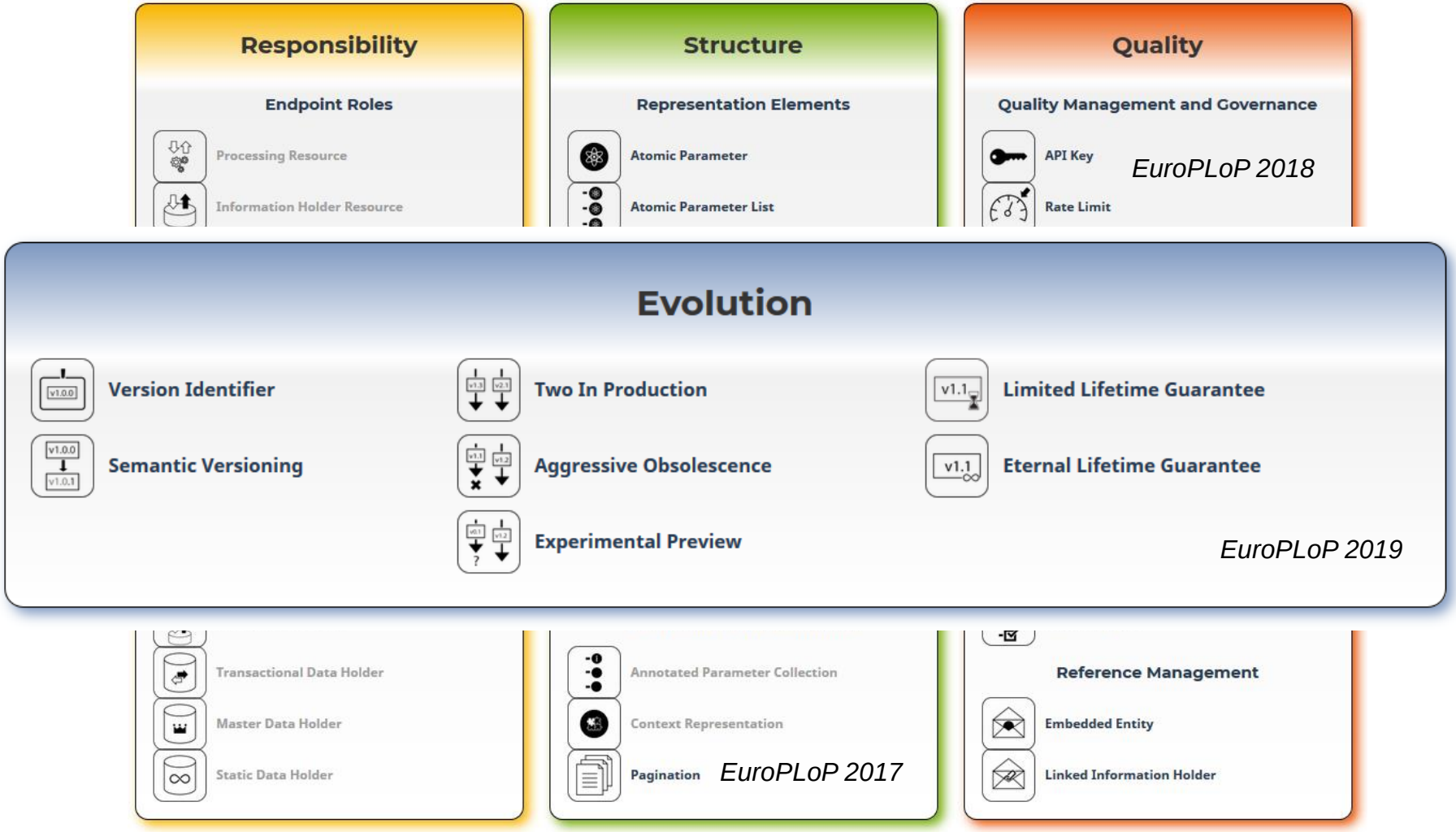


Business Activity Processors: **read-write (update)**

- **Endpoints play different *roles* in microservices architectures – and their operations fulfill certain *responsibilities*:**
  - Pre- and postconditions differ
  - Conversational state?
  - Data consistency vs. currentness

Impact on scalability and changeability?

Microservices API Patterns (MAP): Pattern Index by Category

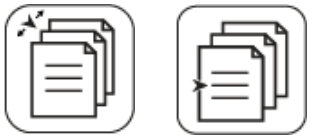


http://microservice-api-patterns.org

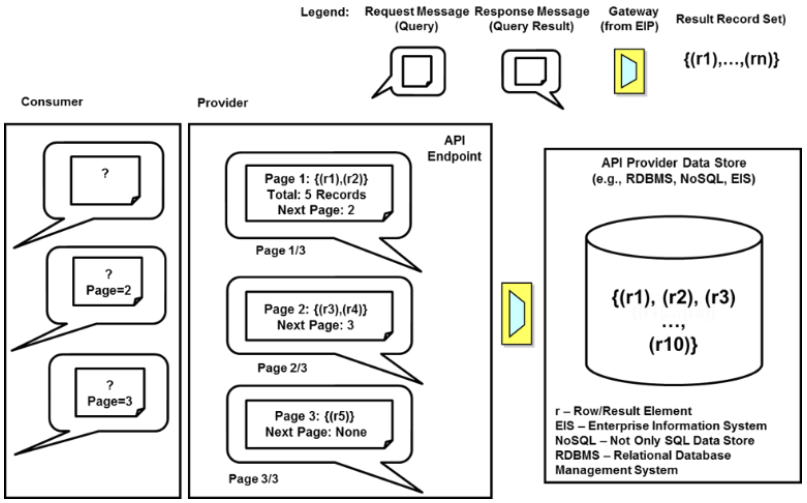
MAP Example: Pagination (1/2)

- **Context**
  - An API endpoint and its calls have been identified and specified.
- **Problem**
  - *How can an API provider optimize a response to an API client that should deliver large amounts of data with the same structure?*
- **Forces**
  - Data set size and data access profile (user needs), especially number of data records required to be available to a consumer
  - Variability of data (are all result elements identically structured? how often do data definitions change?)
  - Memory available for a request (both on provider and on consumer side)
  - Network capabilities (server topology, intermediaries)
  - Security and robustness/reliability concerns

MAP Example: Pagination (2/2)



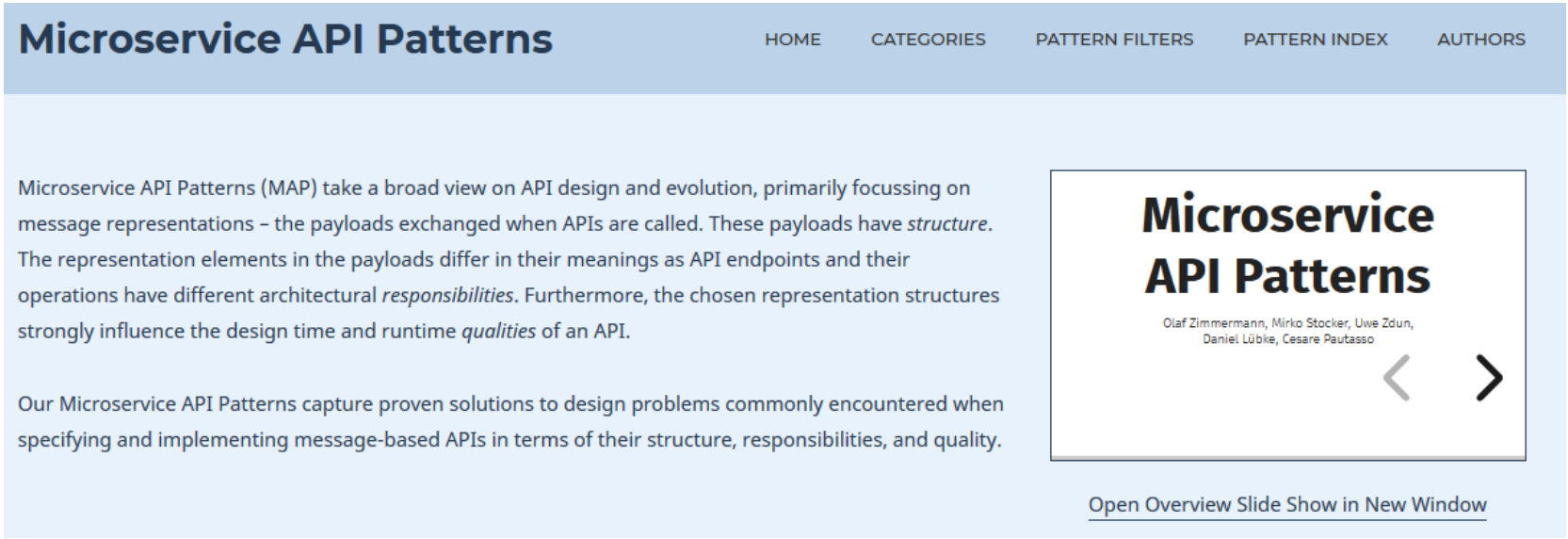
- **Solution**
  - *Divide large response data sets into manageable and easy-to-transmit chunks.*
  - Send only partial results in the first response message and inform the consumer how additional results can be obtained/retrieved incrementally.
  - Process some or all partial responses on the consumer side iteratively as needed; agree on a request correlation and intermediate/partial results termination policy on consumer and provider side.
- **Variants**
  - Cursor-based vs. offset-based
- **Consequences**
  - E.g. state management required
- **Know Uses**
  - Public APIs of social networks



Microservice API Patterns (MAP)

Mini-Exercise: Can MAP serve as a map/guide to API design?

- **Let's have a look at the language organization and selected patterns…**
  - http://microservice-api-patterns.org
    - Website public since 2/2019; experimental preview site available to beta testers
  - Sample patterns (suggestions):
    - Request Bundle, Embedded Entity, Wish List, API Key, Two in Production



Agenda

1. **Microservices Clarification and Level Setting**
   - It's the API contract that matters (not or not only the middleware)
2. **Microservice API Patterns (MAP)**
   - Motivation, publication status, website, examples; tools
3. ***Open Research Questions***
   - *… and early/partial solutions to them*

https://vss.swa.univie.ac.at/2019/

Which BizDevOps issues are still open (community perspective)?

MARCH/APRIL 2017 | IEEE SOFTWARE MAY/JUNE 2018 | IEEE SOFTWARE

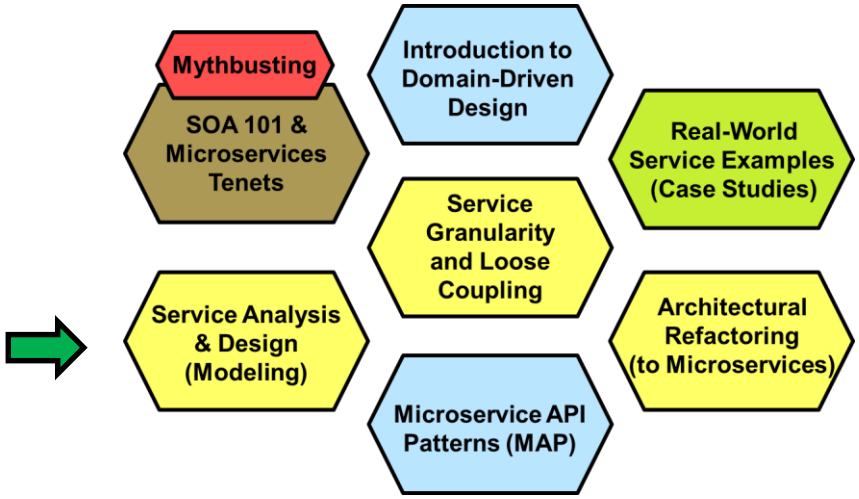
- **IEEE Software Insights, two-part Interview (2016 to 2017):**
  - System-level tools, context boundary management (MA); teach large/distributed system design better, or at all (NJ); data analysis/ML for MSA (JL); biz-MSA isomorphism, team organization(s); autonomy (MA)
- **IEEE Software special Theme Issue on Microservices (2018):**
  - Service Modularization and Refactoring; Service Granularity; Front-end Integration; Resource Monitoring and Management; Failure, Recovery, and Self-Repair; Organizational Culture and Coordination
- **Gray literature (e.g., blog posts, online developer magazines):**
  - Development: DDD and microservices, refactoring to microservices, …
  - Operations: microservices infrastructure architecture patterns, …

Can we come up with an update/a research roadmap here at VSS V2.0?

Critical Success Factors (GI AK Microservices & DevOps, World Cafe)

- **Top Seven (IMHO)**
  - 1. Focus («Zielorientierung»), 2. Pragmatism, 3. Faithfulness («Stiltreue»), 4. Patience («langer Atem»), 5. Learn lessons from the past, 6. Continuous improvement culture, 7. Master of your destiny (no vendor/OSS lock in)
- **Balance amount of (de-)centralization and team autonomy**
  - Just enough freedom vs. just enough governance
  - One DBA per service? Version (test) data (along with code)!
  - Does each team decide its platform stack? Test for sure.
- **Meet Operations Quality Attributes (QAs):**
  - Responsibility (people), accountability
  - Traceability
  - Auditability
  - Reproducability (data, operations at date X)
  - Self recoverability

Open Problem: Service Identification/Design (“DDD 4 SOA/MSA”)



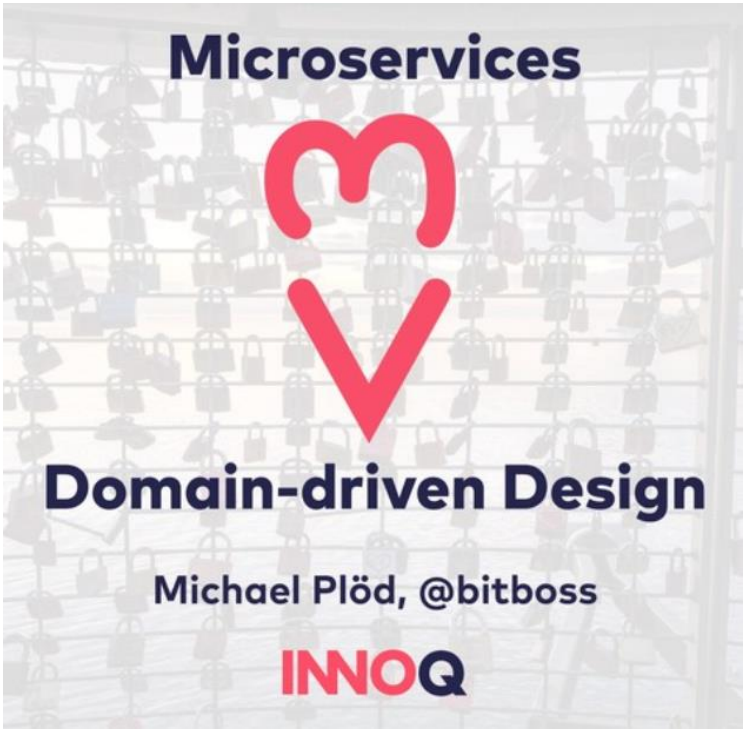
Research Questions

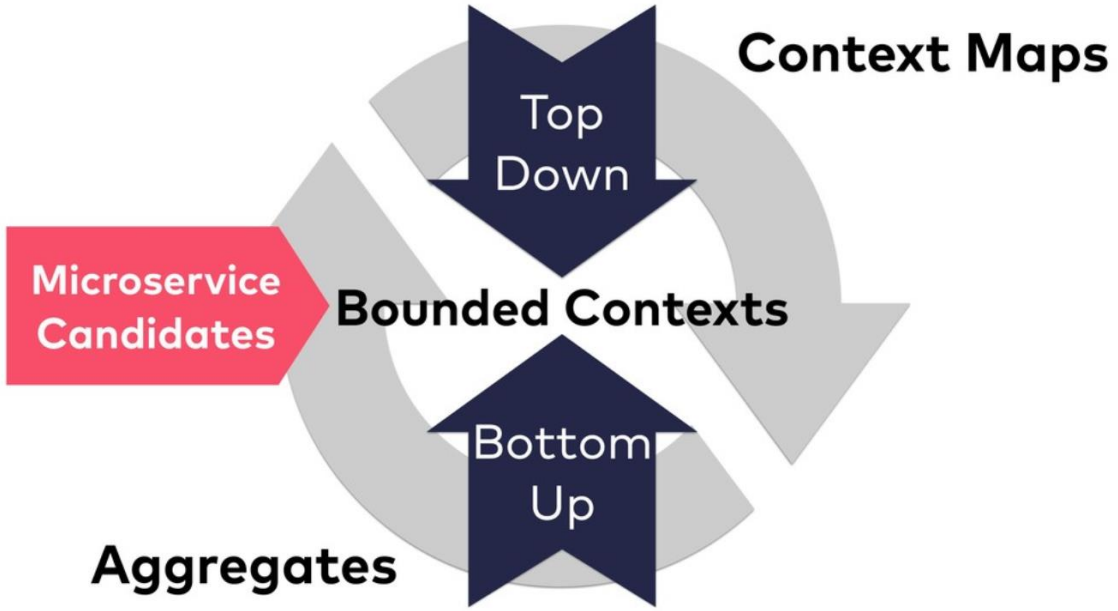
Which existing approaches are particularly suited to analyze and design cloud-native applications and to modernize existing systems (monoliths/megaliths)? How can these patterns be combined with Microservices API Patterns (MAP) – and other SOA/microservices design heuristics – to yield a holistic yet pragmatic *service-oriented analysis and design* practice?

Which patterns and practices do you apply? What are your experiences?

DDD Applied to (Micro-)Service Design

- **M. Ploed is one of the "go-to-guys" here (find him on Speaker Deck)**
  - Applies and extends DDD books by E. Evans and V. Vernon

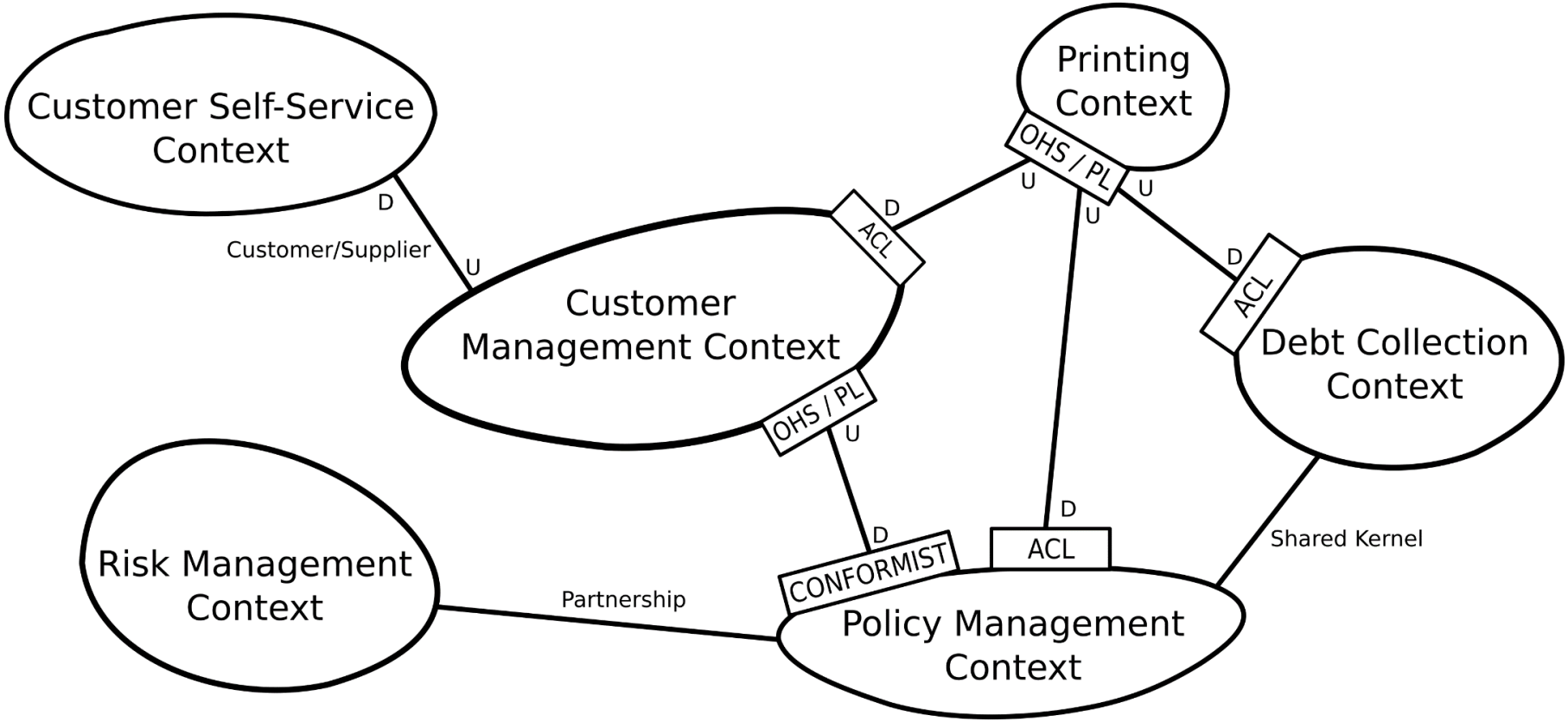




Reference: JUGS presentation, Bern/CH, Jan 9, 2019

Strategic DDD Context Map: Relationship Example

- **Insurance scenario, source: https://contextmapper.github.io/**



D: Downstream, U: Upstream; ACL: Anti-Corruption Layer, OHS: Open Host Service

Context Mapper: A DSL for Strategic DDD

What is Context Mapper?

Context Mapper provides a DSL to create context maps based on **Domain-driven Design (DDD)** and its strategic patterns. DDD and its bounded contexts further provide an approach for **decomposing a domain** into multiple bounded contexts. With our **Service Cutter** integration we illustrate how the Context Mapper DSL (CML) can be used as a foundation for **structured service decomposition approaches**. Additionally, our context maps can be transformed into **PlantUML** diagrams.

CONTEXT MAPPER

- **Eclipse plugin based on:**
  - Xtext
  - ANTLR
  - Sculptor (tactic DDD DSL)
- **Author: S. Kapferer**
  - Term project HSR FHO

```
ContextMap {
  type = SYSTEM_LANDSCAPE
  state = AS_IS

  contains CargoBookingContext
  contains VoyagePlanningContext
  contains LocationContext

  CargoBookingContext <-> VoyagePlanningContext : Shared-Kernel
}
```

Proposal: Microsevice Domain-Specific Language (MDSL)

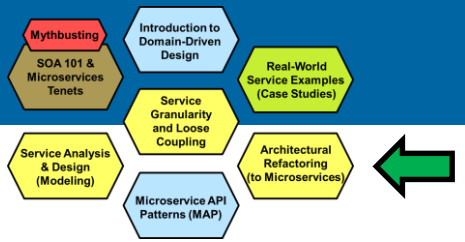
```
API description SpreadSheetExchangeAPI

data type CSVSpreadsheet CSVSheetTab*
data type CSVSheetTab {"name": V<string>,
                       "content": Rows*}
data type Rows {"line": ID<int>,
                "columns":Column+}
data type Column {"position": ID<string>,
                  "header": V<string>?,
                  <<Entity>> "cell": Cell}
data type Cell {"formula":V<string>
               | "intValue": V<int>
               | "longValue": V<long>
               | "text": V<string>}

endpoint type SpreadSheetExchangeEndpoint
exposes
  operation uploadCSVFile
    expecting payload CSVSpreadsheet
    delivering payload "successCode":V<bool>

  operation downloadCSVFile
    expecting payload ID
    delivering payload CSVSpreadsheet
      reporting error "SheetNotFound"

API provider SpreadSheetExchangeAPIProvider
offers SpreadSheetExchangeEndpoint

API client SpreadSheetExchangeAPIClient
consumes SpreadSheetExchangeEndpoint
```

- **Data contract**
  - Compact, technology-neutral
  - Inspired by JSON, regex
- **Endpoints and operations**
  - Elaborate, terminology from MAP domain model
    - Abstraction of REST resource
    - Abstraction of WS-* concepts
- **Plus client, provider, gateway, governance (SLA, version, …)**

How does this notation compare to Swagger/JSON Schema and WSDL/XSD?

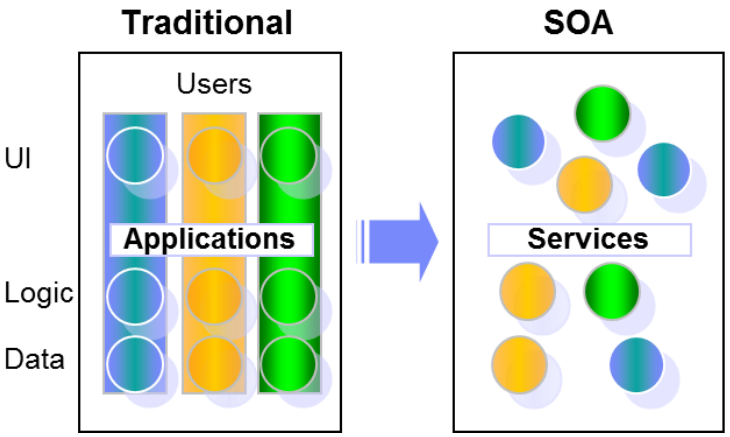
Reference: https://socadk.github.io/MDSL/index

Open Problem: Service Decomposition



On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University



How Do Committees Invent?

Melvin E. Conway

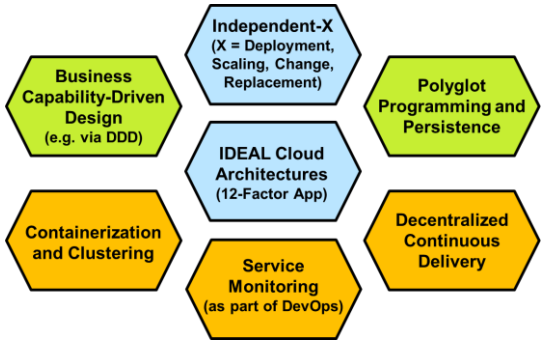
Research Questions

How can systems be decomposed and cut into services (forward engineering)?
How do the applied criteria and heuristics differ
from software engineering and software architecture “classics”
such as *separation of concerns* and *single responsibility principle*?

Which methods and practices do you use? Are they effective and efficient?

Heuristics that do not suffice (IMHO)

- **Two-pizza rule (team size)**
- **Lines of code (in service implementation)**
- **Size of service implementation in IDE editor**



What is wrong with these "metrics" and "best practice" recommendations?

- **Simple if-then-else rules**
  - E.g. "If your application needs coarse-grained services, implement a SOA; if you require fine ones, go the microservices way" (I did not make this up!)
- **Non-technical traits such as "products not projects"**
  - Because context matters, as M. Fowler pointed out at Agile Australia 2018

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

Bachelor Thesis Fall Term 2015

Software

Service Cutter (Proc. Of ESOCC 2016, Springer LNCS)


Lukas Kölbener Michael Gysel

Advisor: Prof. Dr. Olaf Zimmermann

Co-Examiner: Prof. Dr. Andreas Rinkel

Project Partner: Zühlke Engineering AG

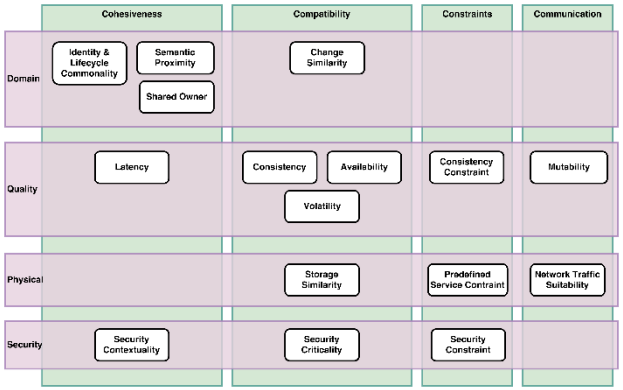
A Software Architect's Dilemma….

How do I split my system into services?

Step 1: Analyze System

- Entity-relationship model
- Use cases
- System characterizations
- Aggregates (DDD)

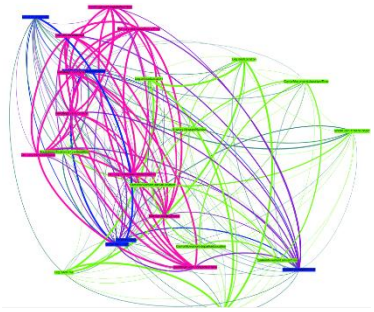
Coupling information is extracted from these artifacts.



The catalog of 16 coupling criteria

Step 2: Calculate Coupling

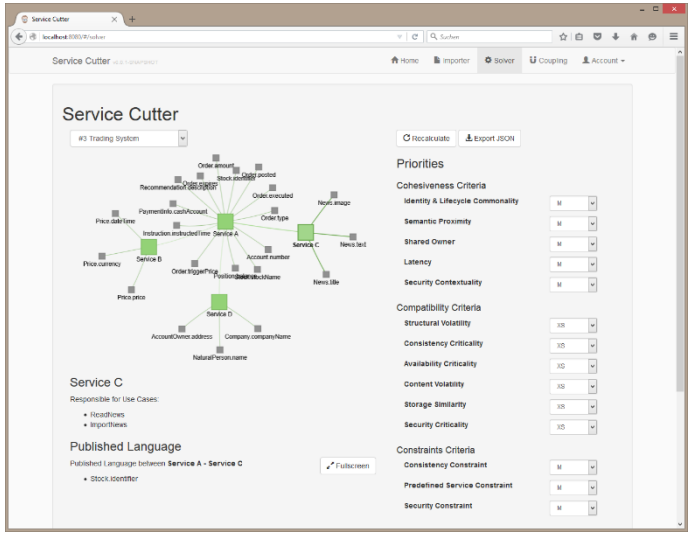
- Data fields, operations and artifacts are nodes.
- Edges are coupled data fields.
- Scoring system calculates edge weights.
- Two different graph clustering algorithms calculate candidate service cuts (=clusters).



A clustered (colors) graph.

Step 3: Visualize Service Cuts

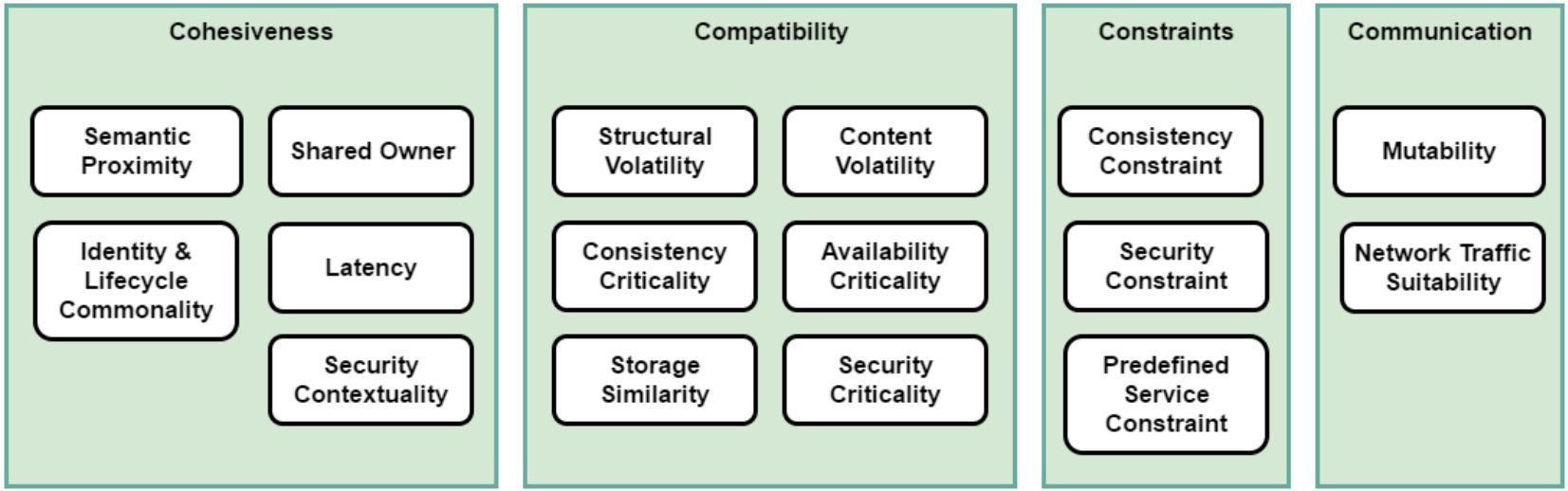
- Priorities are used to reflect the context.
- Published Language (DDD) and use case responsibilities are shown.



Technologies:
Java, Maven, Spring (Core, Boot, Data, Security, MVC), Hibernate, Jersey, JHipster, AngularJS, Bootstrap

https://github.com/ServiceCutter

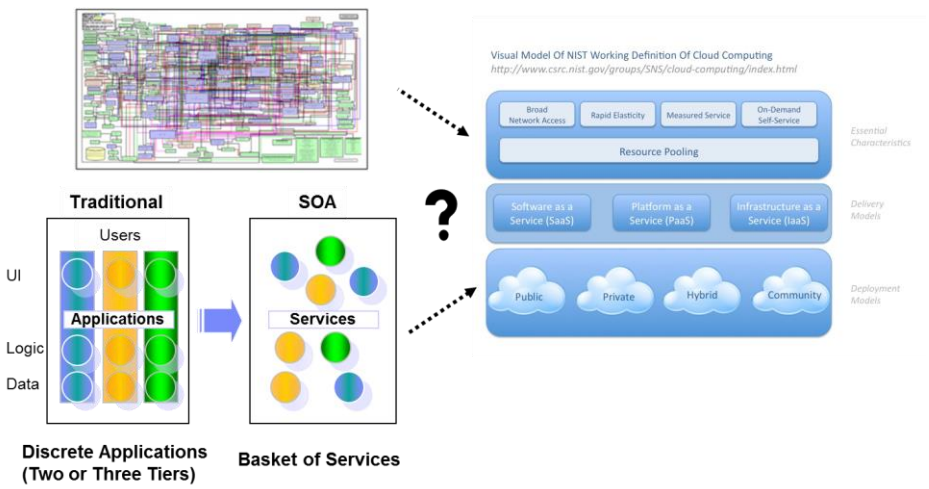
Coupling Criteria (CC) in “Service Cutter” (Ref.: ESOCC 2016)



Full descriptions in CC card format: https://github.com/ServiceCutter/ServiceCutter/wiki/Coupling-Criteria

- **E.g. *Semantic Proximity* can be observed if:**
  - Service candidates are accessed within same use case (read/write)
  - Service candidates are associated in OOAD domain model
- **Coupling impact (note that coupling is a relation not a property):**
  - Change management (e.g., interface contract, DDLs)
  - Creation and retirement of instances (service instance lifecycle)

Open Research Problem: Refactoring to Microservices



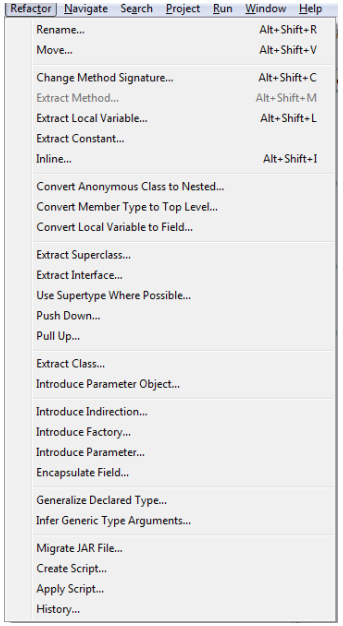
Research Questions

How to *migrate* a modular monolith to a services-based cloud application (a.k.a. cloud migration, brownfield service design)?
Can “micro-migration/modernization” steps be called out?

Which techniques and practices do you employ? Are you content with them?

Code Refactoring vs. Architectural Refactoring

- **Refactoring are “small behavior-preserving transformations” (M. Fowler 1999)**
- **Code refactorings such as “extract method”:**
  - Operate on Abstract Syntax Tree (AST)
  - Based on compiler theory, so well understood and automation possible (e.g., in Eclipse Java/C++)
  - Catalog and commentary:
    - http://refactoring.com/ and https://refactoring.guru/



- ***Architectural refactorings* are different:**
  - Resolve one or more *architectural smells*, have an impact on quality attributes
    - Architectural smell: suspicion that architecture is no longer adequate (“good enough”) under current requirements and constraints (which may differ form original ones)
  - Are carriers of reengineering knowledge (patterns?)
  - Can only be partially automated

Refactoring to Microservices API Patterns

- **Template and cloud refactorings**
  - First published @ SummerSoc 2016

Computing (2017) 99:129–145
DOI 10.1007/s00607-016-0520-y

Architectural refactoring for the cloud: a decision-centric view on cloud migration

Olaf Zimmermann[1]

Coupling Smells

Smell	Suggested Refactoring(s)
API clients and their providers can only be deployed and updated jointly due to a tight coupling	Downsize data contract by adding Linked Information Holders replacing Embedded Entities

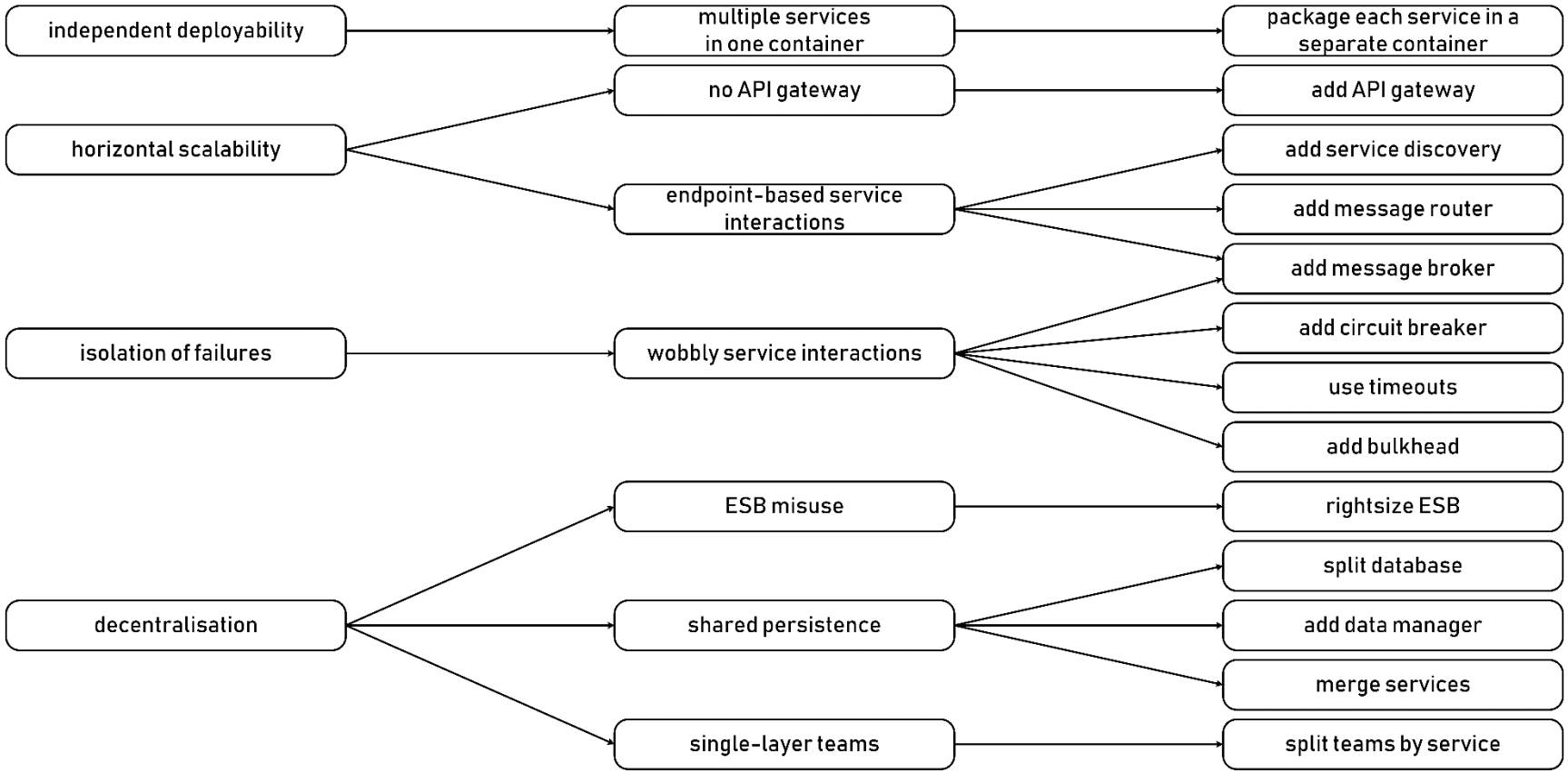
Granularity Smells

Smell	Suggested Refactoring(s)
God service with many operations that takes long to update, test and deploy	*Split Service*
Fat Information Holder violating SRP	*Split Information Holder* according to data lifetime and incoming dependencies
Big Ball of Service Mud (doing processing and data access)	Split into Processing Resource and Information Holder Resource (CQRS for API)
Service proliferation syndrome (unmanageable)	Consolidate different processing responsibility types into single Business Activity Processor

- **Microservices refactorings:**
  - Future work for MAP

Work in progress!

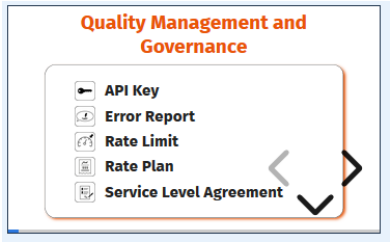
SummerSoC 2019: Joint Work with University to Pisa



Reference: Brogi, A., Neri D., Soldani, J., Zimmermann, O., *Design Principles, Architectural Smells and Refactorings for Microservices*: A Multivocal Review. CoRR abs/1906.01553 and Springer SICS (2019, to appear)

Key Messages of this Talk

- **It is the API contract (and its implementations) that make or break projects, not (or not only) middleware and tools**
  - Frameworks and infrastructures come and go, APIs stay



- ***Microservices API Patterns (MAP)* language**
  - Public MAP website now available in Version 1.2.1
  - 20+ patterns, sample implementation in public repo, supporting tools
- ***Context Mapper* tool supporting strategic Domain-Driven Driven Design (DDD) and architectural refactoring**
  - Other tools emerging

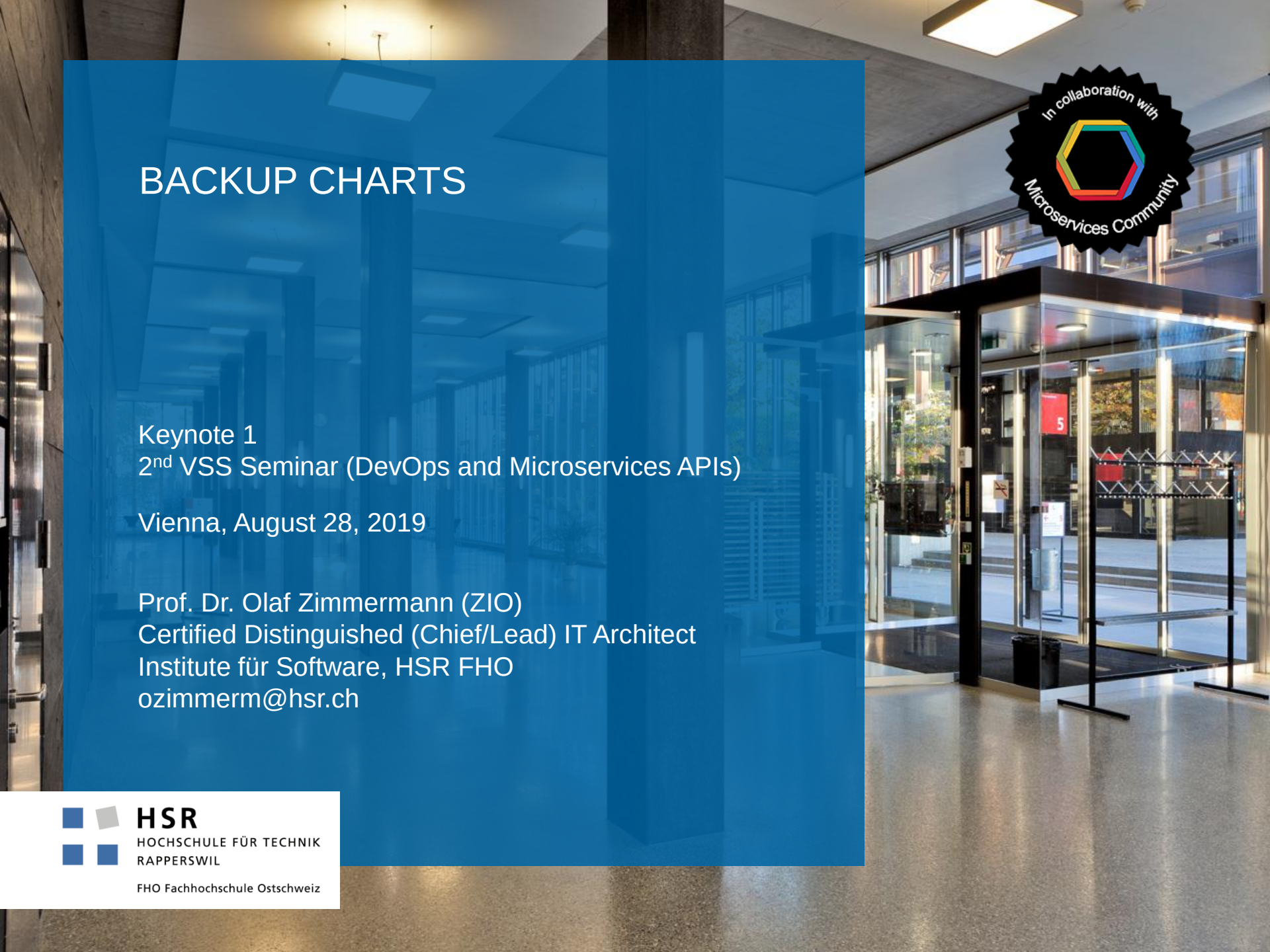
- ***Microservices Domain-Specific Language (MDSL)***
  - Uses MAP as language constructs
  - Can be generated from DDD bounded contexts

```
data type Customer {"name": V<string>, "address"

endpoint type CustomerLookup
exposes
  operation findCustomer
    expecting payload "searchFilter": V<string>
    delivering payload "customerList": Customer*
```

- **Research areas:**
  - Service modeling, identification, decomposition, refactoring

Teaser Question Revisited

- **You have been tasked to develop a RESTful HTTP API for a master data management system that stores customer records and allows sales staff to analyze customer behavior. The system is implemented in Java and Spring. An additional access channel is RabbitMQ.**
- **What do you do?**
  a) *I hand over to my software engineers and students because all I manage to do these days is attend meetings and write funding proposals.*
  b) *I annotate the existing Java interfaces with @POST and @GET, as defined in Spring MVC or JAX-RS etc. and the job is done.*
  c) *I install an API gateway product in Kubernetes and hire a sys admin, done.*
  d) *I design a service layer and write an Open API Specification (f.k.a. Swagger) contract as well as Data Transfer Objects (DTOs). I worry about message sizes, transaction boundaries, compensation and coupling while implementing the contract. To resolve such issues, I create my own novel solutions. Writing infrastructure code and frameworks is fun after all!*
  e) *I leverage the patterns in MAP during API design and development ☺*



BACKUP CHARTS

Keynote 1
2nd VSS Seminar (DevOps and Microservices APIs)

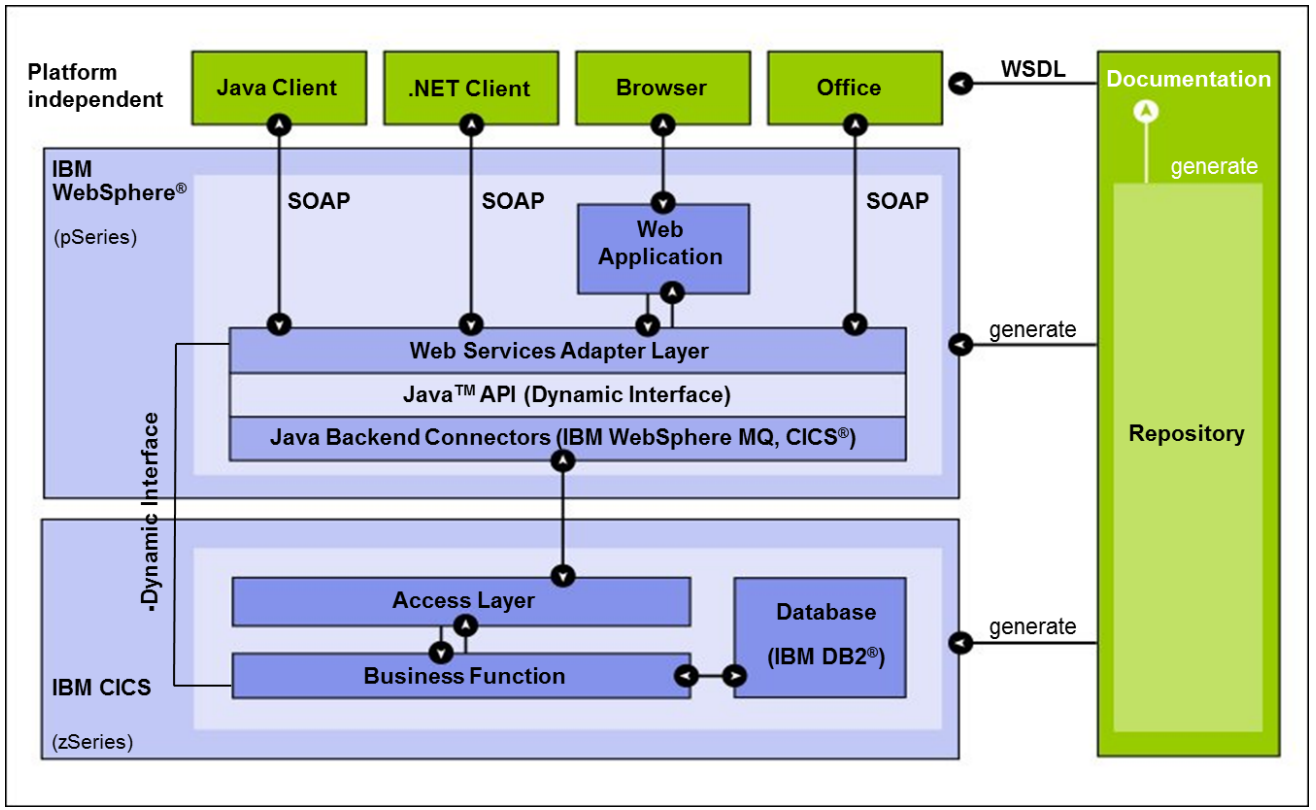
Vienna, August 28, 2019

Prof. Dr. Olaf Zimmermann (ZIO)
Certified Distinguished (Chief/Lead) IT Architect
Institute für Software, HSR FHO
ozimmerm@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Sample Project: Financial Services Provider (for Retail Banks)

Reference: IBM, ACM OOPSLA 2004



- **Supports – and partially automates – core banking business processes**
  - More than 1000 of business services, each providing a single operation
  - One database repository, logically partitioned

Exemplary Service Operations in Core Banking

	Fine (business)	*Coarse (business)*
Fine (technical)	"Hello world" of core banking: `int getAccountBalance (CustomerId)`	"Big data" customer profiling (condensed): `ActivityClassificationEnum scoreMonthlyInvestmentActivity (CustomerId, Month, Year)`
Coarse (technical)	Single domain entity, but complex payload (search/filter capability): `CustomerDTOSet searchCustomers (WildcardedCustomerName, CustomerSegment, Region)`	Deep analytics («Kundengesamtübersicht»): `BankingProductPortfolioCollection prepareCustomerAnalysisForMeeting (CustomerId, Timeframe)`

- **Business granularity:**
  - Functional scope, domain model coverage
- **Technical granularity:**
  - Structure of message representations a.k.a. Data Transfer Object (DTOs)

Business alignment/agility?
Independent deployability?
Client/server coupling?

What is Service-Oriented Architecture (SOA)?

No single definition – “SOA is different things to different people”:

- A set of **services** and operations that a business wants to expose to their customers and partners, or other portions of the organization.
  - *Note: no scope implied, enterprise-wide or application!*
- An architectural style which requires a **service provider**, a **service requestor** (consumer) and a **service contract** (a.k.a. client/server).
  - *Note: this is where the “business-alignment” becomes real!*
- A set of architectural patterns such as **service layer** (with remote facades, data transfer objects), enterprise service bus, service composition (choreography/orchestration), and service registry, promoting principles such as modularity, layering, and **loose coupling** to achieve design goals such as reuse, and flexibility.
  - *Note: not all patterns have to be used all the time!*
- A **programming and deployment model** realized by standards, tools and technologies such as Web services (WSDL/SOAP), RESTful HTTP, or asynchronous message queuing (AMQP etc.)
  - *Note: the “such as” matters (and always has)!*

Business Domain Analyst

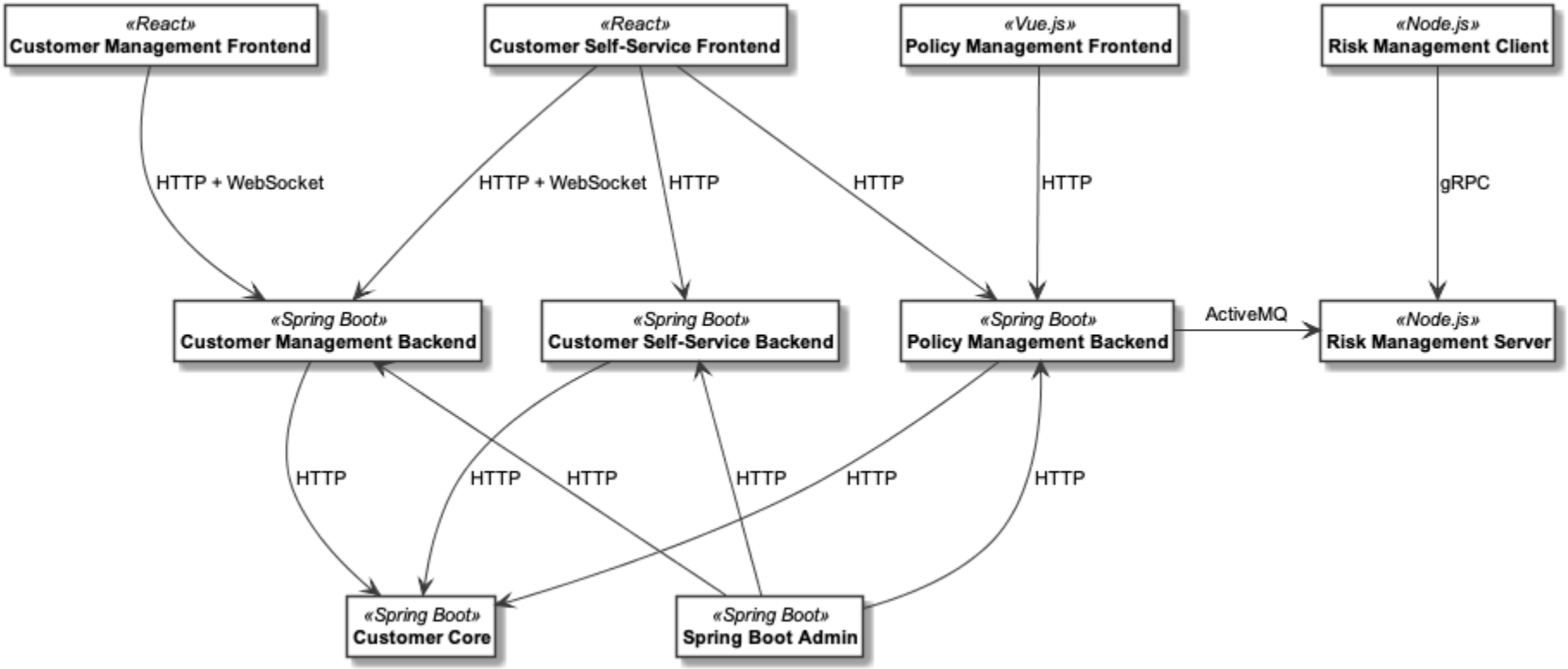
IT Architect

Developer, Administrator

Based on and adapted from: IBM SOA Solution Stack, IBM developerWorks

A Bigger Microservices and MAPing Example: Lakeside Mutual

- https://github.com/Microservice-API-Patterns/LakesideMutual



Microservices Publications Beyond MAP (2017, 2018)

- **Zimmermann, O.:** [Microservices Tenets – Agile Approach to Service Development and Deployment]
  - Springer Comp Sci Res Dev, 2017, http://rdcu.be/mJPz

(screen captions are hyperlinks)

- **Pardon, G., Pautasso, C., Zimmermann, O.:** [Consistent Disaster Recovery for Microservices: the Backup, Availability, Consistency (BAC) Theorem]
  - In: IEEE Cloud Computing, 5(1) 2018, pp. 49-59.
- **Pahl, C., Jamshidi, P., Zimmermann, O.:** [Architectural Principles for Cloud Software]
  - In: ACM Trans. on Internet Technology (TOIT), 18 (2) 2018, pp. 17:1-17:23.
- **Furda, A., Fidge, C., Zimmermann, O., Kelly, W., Barros, A.:** [Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency]
  - In: IEEE Software, 35 (3) 2018, pp. 63-72.