FROM DOMAIN-DRIVEN DESIGN TO MICROSERVICE APIS OF QUALITY AND STYLE: CONTEXT, CONTRACTS, COMPONENTS

GI-Arbeitskreis Microservices und DevOps

Berlin, March 9, 2020

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

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. A backend B2B channel uses message queues (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 let libraries and frameworks finish the job.*
  c) *I install an API gateway product in Kubernetes and hire a sys admin, done.*
  d) *I design a service layer (Remote Facade with Data Transfer Objects) and publish an Open API Specification (f.k.a. Swagger) contract. I worry about message sizes, transaction boundaries, error handling and coupling criteria while implementing the contract. To resolve such issues, I create my own novel solutions. Writing infrastructure code and test cases is fun after all!*
  e) ____ ?

© Olaf Zimmermann, 2020.

Agenda Today (And Key Take Away Messages)

1. ***Context* matters**
   - One size does not fit all (top-level design heuristic: "it depends")
   - Strategic and tactic Domain-Driven Design (DDD)
   - Context Mapper DSL and tools
2. ***Contracts* rule**
   - Unified interfaces are great, but not enough
   - More SOA and microservices myth busting
   - Microservice Domain-Specific Language (MDSL)
3. ***Components* contain (cost and risk)**
   - Towards a context-driven, contract-first service identification method
   - Microservice API Patterns (MAP) to structure the solution space
   - (time permitting) Industry trends and resulting research questions
     - Microfrontends, containerization, cloud-native 12-factor applications

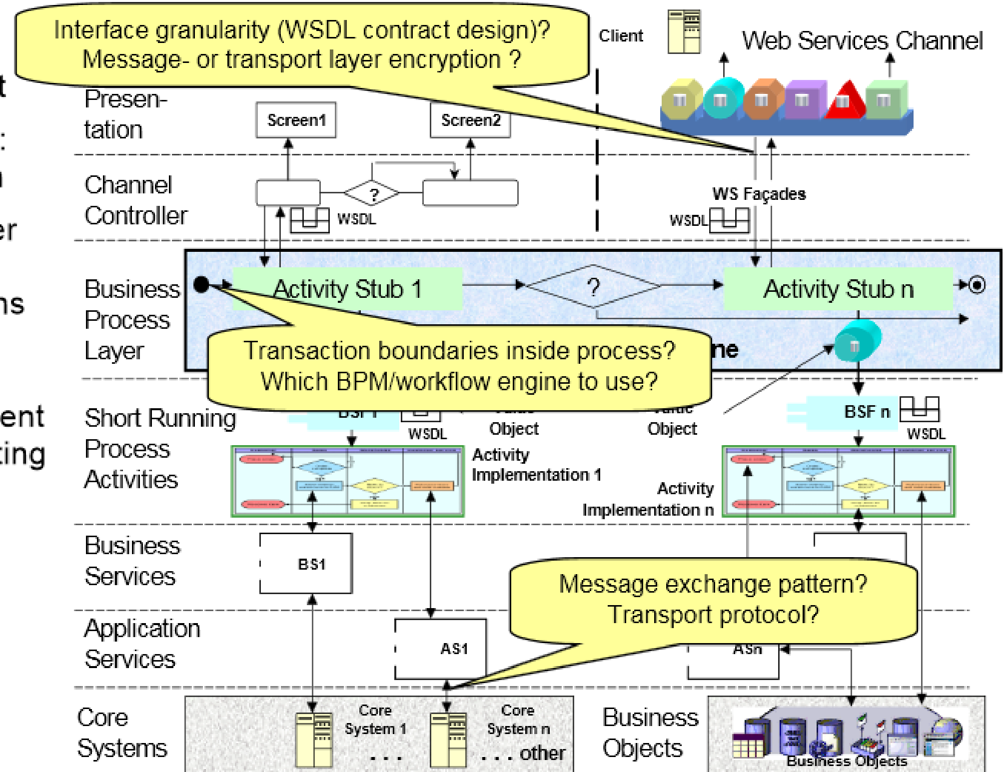
© Olaf Zimmermann, 2020.

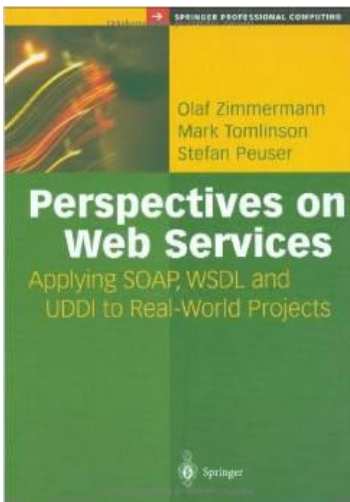
SOA 1.0: Order Management Application (Telecommunications)

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



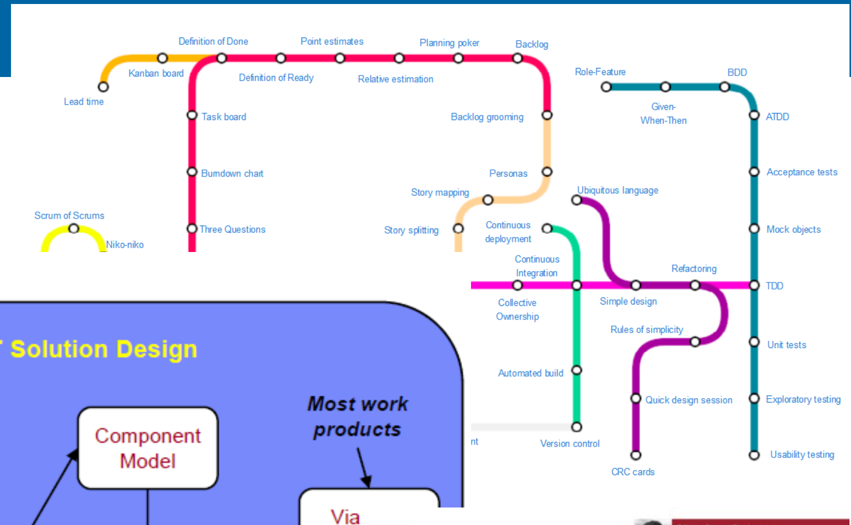


 © 2007 IBM Corporation

© Olaf Zimmermann, 2020.

Context Matters

Agile practices



Professional services methods

IT Solution Requirements Analysis
- Use Case Model
- System Context
- Non-functional requirements

Business Requirements Specification

Automation

Requirements on IT

IT Solution Design
- IT Standards
- Reference Architectures
- Architecture Overview Diagram
- Current IT Environment
- Component Model
- Deployment Units
- Operational Model
- Via Ana
- Service Char. An

Most work products

Structure & form

Development

WP WP WP Business Requirmnts Specificatn

WP WP WP IT Solution Requiremnts Analysis

WP WP WP IT Solution Design

WP WP WP Analysis & Development

WP WP WP Deploymnt & Operations

WP WP WP Test

INSIGHTS

Editor: Cesare Pautasso
University of Lugano
c.pautasso@ieee.org

Editor: Olaf Zimmerman
University of Applied Sciences of Eastern Switzerland, Rapperswil
ozimmerm@hsr.ch

Context Is King
What's Your Software's Operating Range?

Francisco Torres

Talking with users might change how you see the context of your software project, often in unexpected ways. Drawing from his experience on spacecraft operations software projects, Francisco Torres shares stories on how listening to users taught him to stop making assumptions and helped him define his software's operating range: the set of quality properties in which a software system can successfully run. —*Cesare Pautasso and Olaf Zimmerman*

THEY SAY THAT experience is what you get when you were expecting something else. This happened to me some years ago while I was studying human–machine interface concepts for a software system for spacecraft operations. One initial project activity involved analyzing the existing user interface to identify improvement areas and new concepts to prototype and explore. To that end, my colleagues and I designed a questionnaire on usability aspects, which a sample population of the users answered. Some of the answers were counterintuitive, to say the least. In time, and after giving it some thought, it suddenly dawned on me that I had lost sight of one key element: context. With context in mind, everything made more sense.

This lesson about context applies to not only GUI-related topics but also many other software-engineering areas, such as processes and tools. They all have a nominal operating range, so to speak. More on this later, but first I'll walk you through some of the GUI attributes our study explored. In each case, I'll brief you on the feedback we received through the questionnaire and explain why, despite not matching my expectations, the users' point of view ultimately made sense.

When the Stakes Are Too High

The questionnaire asked the respondents to assess how intuitive and user friendly the various displays were. We got some answers along the lines that some displays were "not user friendly," "far from intuitive," and "clumsy in the extreme." However, one theme also recurred in many of the answers. Learnability and intuitiveness were less of an issue because the users had been formally trained to operate the system, including rehearsals and simulations.

One respondent stated that it wasn't possible to use the system simply intuitively; some a priori knowledge was required. Another one even proposed to redesign some of the displays, trading intuitiveness for efficient screen real es-

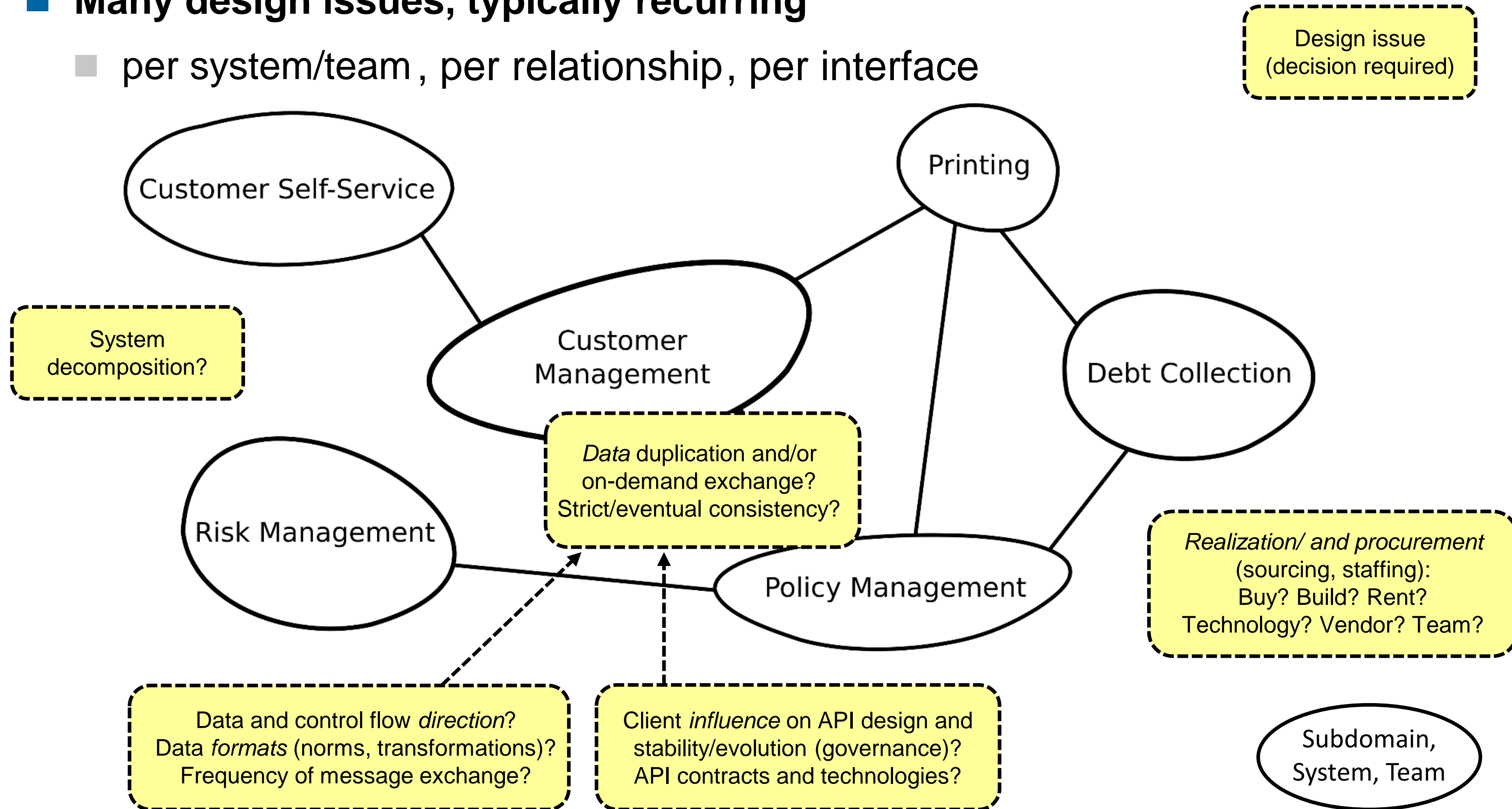
SEPTEMBER/OCTOBER 2015 | IEEE SOFTWARE 9

Experience reports

© Olaf Zimmermann, 2020.

“Fictitious” Insurance Application/Integration Landscape

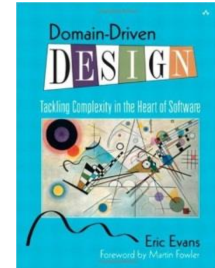
- **Many design issues, typically recurring**
  - per system/team, per relationship, per interface

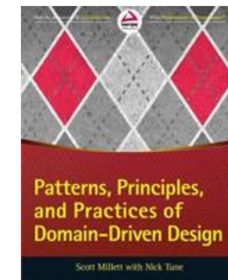


© Stefan Kapferer, Olaf Zimmermann, 2020.

Domain-Driven Design (DDD) Overview

- **Emphasizes need for modeling and communication**
  - Ubiquitous language (vocabulary) – the *domain model*
- ***Tactic DDD* – "Object-Oriented Analysis and Design (OOAD) done right"**
  - Emphasis on business logic in layered architecture
  - Decomposes Domain Model pattern from M. Fowler
  - Patterns for common roles, e.g. Entity, Value Object, Repository, Factory, Service; grouped into *Aggregates*
- ***Strategic DDD* – "agile Enterprise Architecture and/or Portfolio Management"**
  - Models have boundaries
  - Teams, systems and their relations shown in *Context Maps* of *Bounded Contexts*





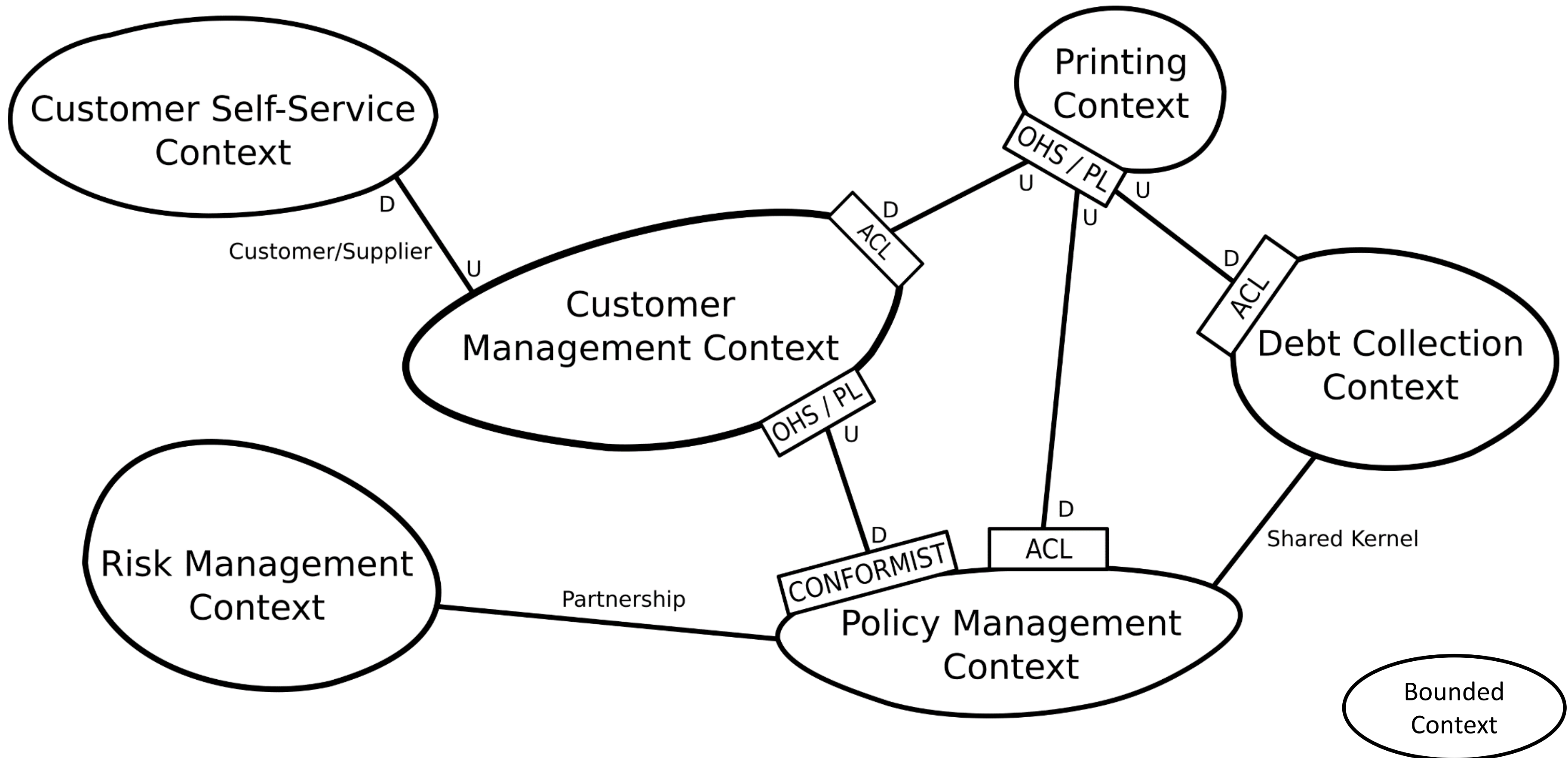
Books (Selection, Reverse Chronological Order)

- M. Ploed, Hands-on Domain-diven Design - by example, Leanpub
- Domain-Driven Design: The First 15 Years, Leanpub
- V. Vernon, DDD Distilled; a German translation is available: DDD Kompakt
- S. Millett with N. Tune, Patterns, Principles, and Practices of DDD, J. Wiley & Sons 2015
- V. Vaughn, Implementing DDD, Addison Wesley 2014
- F. Marinescu, Domain-Driven Design Quickly (InfoQ e-book, 2006)

© Olaf Zimmermann, 2020.

A Strategic DDD Context Map with Relationships

- **Insurance scenario, example model from https://contextmapper.org/**



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

© Stefan Kapferer, Olaf Zimmermann, 2020.

What is Context Mapper?

Context Mapper provides a DSL to create **Context Maps** based on strategic **Domain-driven Design (DDD)**. DDD with its Bounded Contexts offers an approach for **decomposing a domain or system** into multiple independently deployable (micro-)services. With our **Architectural Refactorings (ARs)** we provide transformation tools to refactor and decompose a system in an iterative way. The tool further allows you to generate **MDSL (micro-)service contracts** providing assistance regarding how your system can be implemented in an **(micro-)service-oriented architecture**. In addition, **PlantUML** diagrams can be generated to transform the Context Maps into a **graphical representation**. With **Service Cutter** you can generate suggestions for new services and Bounded Contexts.

CONTEXT MAPPER

```
ContextMap DDD_CargoSample_Map {
    type = SYSTEM_LANDSCAPE
    state = AS_IS

    contains CargoBookingContext
    contains VoyagePlanningContext
    contains LocationContext

    CargoBookingContext [SK]<->[SK] VoyagePlanningContext

    CargoBookingContext [D]<-[U,OHS,PL] LocationContext

    VoyagePlanningContext [D]<-[U,OHS,PL] LocationContext
}
```

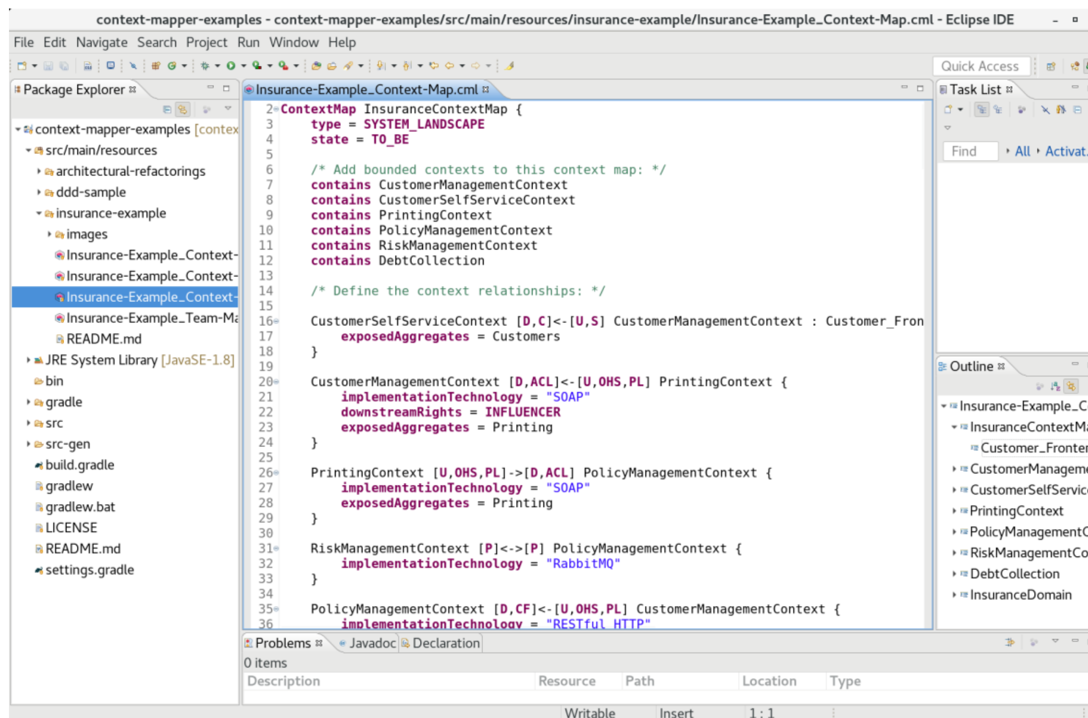
SK: Shared Kernel, **PL:** Published Language
D: Downstream, **U:** Upstream
ACL: Anti-Corruption Layer, **OHS:** Open Host Service

- **Eclipse plugin, based on:**
  - Xtext, ANTLR
  - Sculptor (tactic DDD DSL)
- **Creator: S. Kapferer**
  - Term projects and Master thesis @ HSR FHO

© Stefan Kapferer, Olaf Zimmermann, 2020.

Context Mapper: DSL implements Meta-Model and Semantics

- **A Domain-Specific Language (DSL) for DDD:**
  - Formal, machine-readable DDD Context Maps via *editors and validators*
  - Model/code *generators* to convert models into other representations
  - Model transformations for *refactorings* (e.g., “Split Bounded Context”)



Plugin update site: https://dl.bintray.com/contextmapper/context-mapping-dsl/updates/

© Stefan Kapferer, Olaf Zimmermann, 2020.

Context Mapper: Domain-Specific Language

```
ContextMap DDDSampleMap {
    contains CargoBookingContext
    contains VoyagePlanningContext
    contains LocationContext

    CargoBookingContext [SK]<->[SK] VoyagePlanningContext

    [U,OHS,PL] LocationContext -> [D] CargoBookingContext

    VoyagePlanningContext [D]<-[U,OHS,PL] LocationContext
}
```

Bounded Contexts (systems or teams)

DDD relationship patterns (role of endpoint)

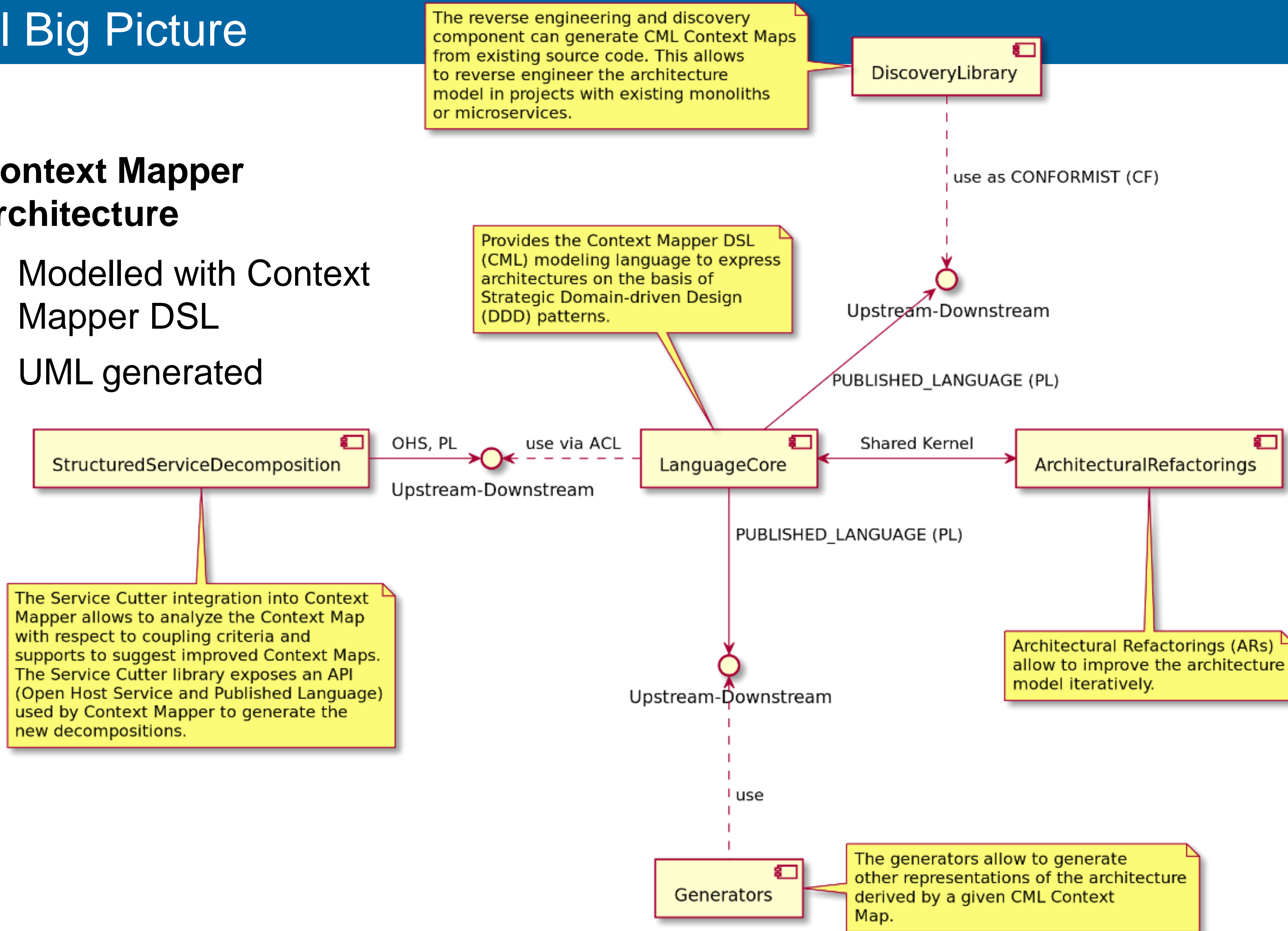
Influence/data flow direction: ->, <-> (upstream-downstream or symmetric)

SK: Shared Kernel, PL: Published Language
D: Downstream, U: Upstream
ACL: Anti-Corruption Layer, OHS: Open Host Service

© Stefan Kapferer, Olaf Zimmermann, 2020.

Tool Big Picture

- **Context Mapper architecture**
  - Modelled with Context Mapper DSL
  - UML generated



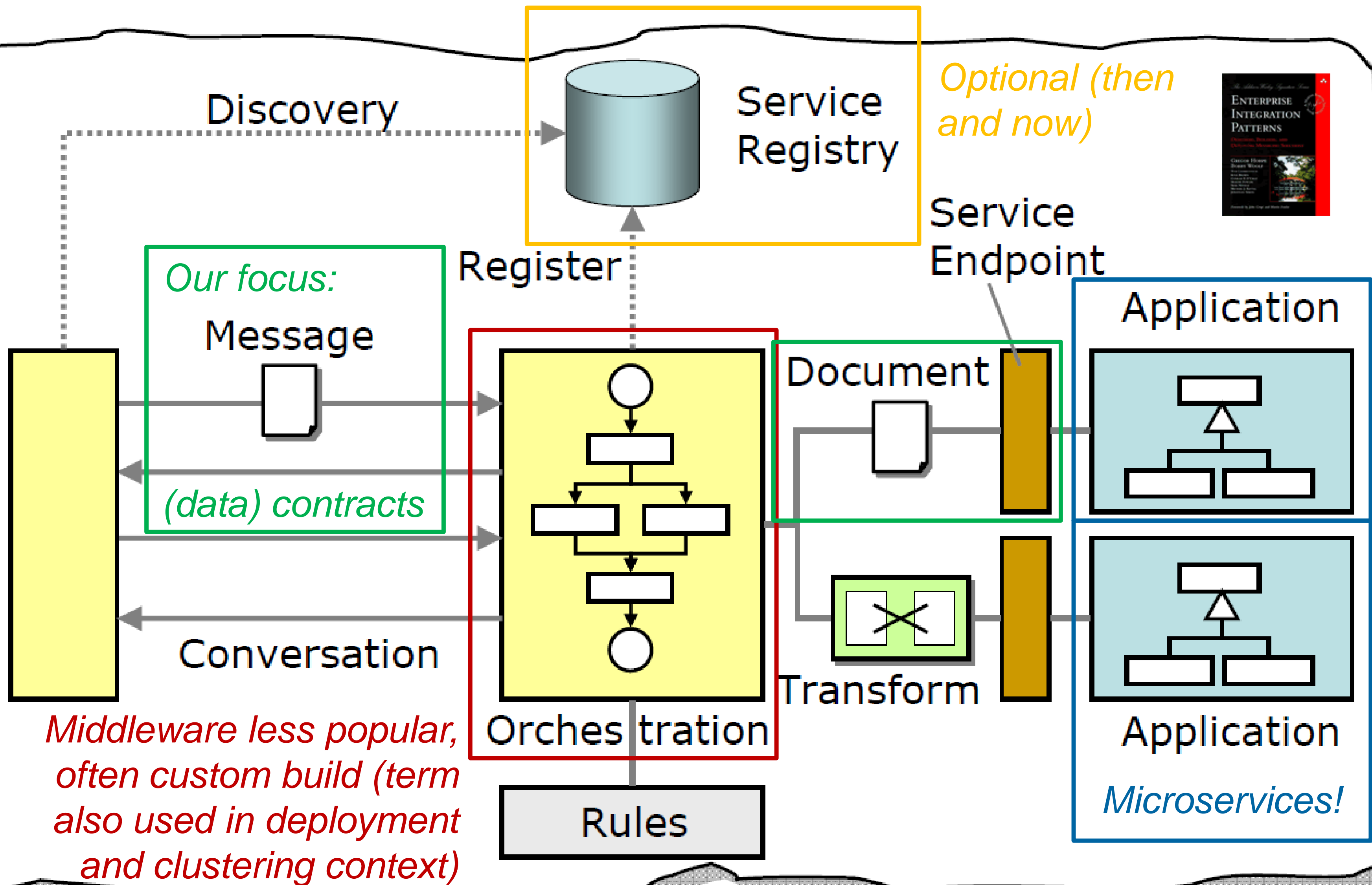
© Olaf Zimmermann, 2020.

Agenda Today (And Key Take Away Messages)

1. **Context matters**
   - One size does not fit all (top-level design heuristic: "it depends")
   - Strategic and tactic Domain-Driven Design (DDD)
   - Context Mapper DSL and tools
2. ***Contracts rule***
   - *Unified interfaces are great, but not enough*
   - *More SOA and microservices myth busting*
   - *Microservice Domain-Specific Language (MDSL)*
3. ***Components* contain (cost and risk)**
   - Towards a context-driven, contract-first service identification method
   - Microservice API Patterns (MAP) to structure the solution space
   - (time permitting) Industry trends and resulting research questions
     - Microfrontends, containerization, cloud-native 12-factor applications

© Olaf Zimmermann, 2020.

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



© Olaf Zimmermann, 2020.

Mythbusting (1/4): SOA 1.0 (2003/2004 to 2008/2009)

- ***Myth:* SOA and microservices solve different problems, not comparable**
  - Application boundaries blurred in the Web age
  - See Microservices Tenets article, see OOPSLA practitioner reports
- ***Myth:* Traditional SOA is "heavyweight" and requires centralization and enterprise-wide data normalization in an Enterprise Service Bus (ESB)**
  - What is heavyweight (definition)? Resource usage? Maintenance?
    - SOAP also uses HTTP by default; JSON not much lighter than "nice" XML
    - Have a look at the dependencies of services meshes (example: Istio)
  - Most practices recommended today already appeared in the (good) SOA tutorials in the 2000s
    - e.g. no canonical data model, no single point of failure, no business logic in ESB
    - Yes, poor SIA implementations did occur (but that also holds for microservices)
- ***Myth:* SOA and XML-based "Web" services are coupled with each other**
  - Actually, they are less related that REST and HTTP are
    - Although REST claims to be an architectural style (only implemented once)

© Olaf Zimmermann, 2020.

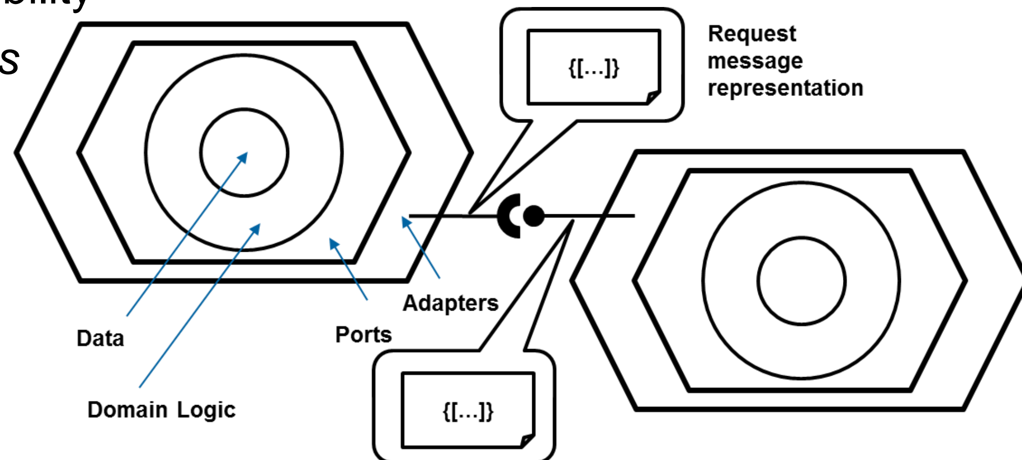
Mythbusting (2/4): Web Services and REST

- ***Myth:* REST is a protocol**
  - It is an architectural style defined by abstract constraints
  - So asking for a “REST API” is like asking for “Gothic window” (material?)
- ***Myth:* SOAP is a protocol**
  - It is a message exchange format, HTTP typically used for message transfer
    - Other protocols (theoretically) possible
- ***Myth:* REST and SOAP can be compared**
  - Can the Gothic style and concrete building materials/norms be compared?
- ***Myth:* Thought leaders are objective and independent**
  - There is an “industrial NN complex” (NN = Agile, REST, …)
    - To paraphrase M. Fowler at Agile Australia
  - Book authors and consultants do have commercial agendas (lie vendors)
    - And should not reference their own papers/books only (SOA Design Patterns?)

© Olaf Zimmermann, 2020.

A Consolidated Definition of Microservices

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



Detailed analysis: 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), IDEALly in a loosely coupled fashion
  - Facilitating *polyglot programming and persistence*
  - Leveraging DevOps practices including decentralized *continuous delivery* and *end-to-end monitoring* (for business agility and domain observability)

© Olaf Zimmermann, 2020.

Mythbusting (3/4): Microservices (since 2014)

- ***Myth:* Self-Contained Systems are new, different form MS(A) and "monolith"**
  - Evidence: e.g., S. Brown: Modular Monolith
- ***Myth:* Distributed service mesh sidecars are easier to create, configure, manage than SOA-days ESBs**
  - Evidence: notion of federated ESBs, EIP pattern mapping
  - Open source lock in replacing vendor lock in
- ***Myth:* RESTful HTTP is the only protocol that is required and permitted**
  - MOM and even RPC have their place
    - Evidence: Google gRPC, S. Newman first book on Microservices
- ***Myth:* Unified interface is sufficient as contract**
  - The success of Swagger/Open API Specification suggests that more elaborate API Descriptions are required
    - Data contract, pre- and postconditions, error handling, …

© Olaf Zimmermann, 2020.

OpenAPI Specification (OAS): An Interface Definition Language (IDL)

- **Wikipedia lists (only) 23 IDLs**
  - OAS is one of them
  - Bound to HTTP

TOOLS

SWAGGER UI

Use a Swagger specification to drive your API documentation. **Demo** and **Download**.

SWAGGER EDITOR

An editor for designing Swagger specifications from scratch, using a simple YAML structure. **Demo** and **Source**.

SDK GENERATORS

Turn an API spec into client SDKs or server-side code with **Swagger Codegen**.

```yaml
/pets/{petId}:
  get:
    summary: Info for a specific pet
    operationId: showPetById
    tags:
      - pets
    parameters:
      - name: petId
        in: path
        required: true
        description: The id of the pet to retrieve
        schema:
          type: string
    responses:
      '200':
        description: Expected response to a valid request
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Pet"
      default:
        description: unexpected error
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Error"
```


© Olaf Zimmermann, 2020.

Contracts in Microservice Domain-Specific Language (MDSL)

```
API description SpreadSheetExchangeAPI

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

endpoint type SpreadSheetExchangeEndpoint serves as TRANSFER_RESOURCE
exposes
  operation uploadSpreadheet with responsibility NOTIFICATION_OPERATION
    expecting payload CSVSpreadsheet
    delivering payload {"successFlag":D<bool>, ID}

  operation downloadCSVFile with responsibility RETRIEVAL_OPERATION
    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
- **API 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

© Olaf Zimmermann, 2020.

Mythbusting (4/4): (Micro-)Services Design

- ***Myth:* Services always must be small/fine-grained**
  - How to measure? How to observe?
  - What about dependencies? They increase.
- ***Myth:* A business capability has to be a function**
  - And Entity Service (always) are an anti pattern
  - Archive? Logbook? File share?
- ***Myth:* The DDD patterns fully solve the decomposition problem**
  - Process required (and related knowleddge/patterns), see here and here
  - Subdomains and Aggregates and Bounded Contexts (BCs) are as hard to find as services, so "turn BC into microservice" only delegates the problem
- ***Myth:* "Hello World" implementations are suited to demonstrate the value and price of microservices**
  - Domain model needs to have a certain size and complexity e.g., to see ramifications of replication, eventual consistency (see Lakeside Mutual)

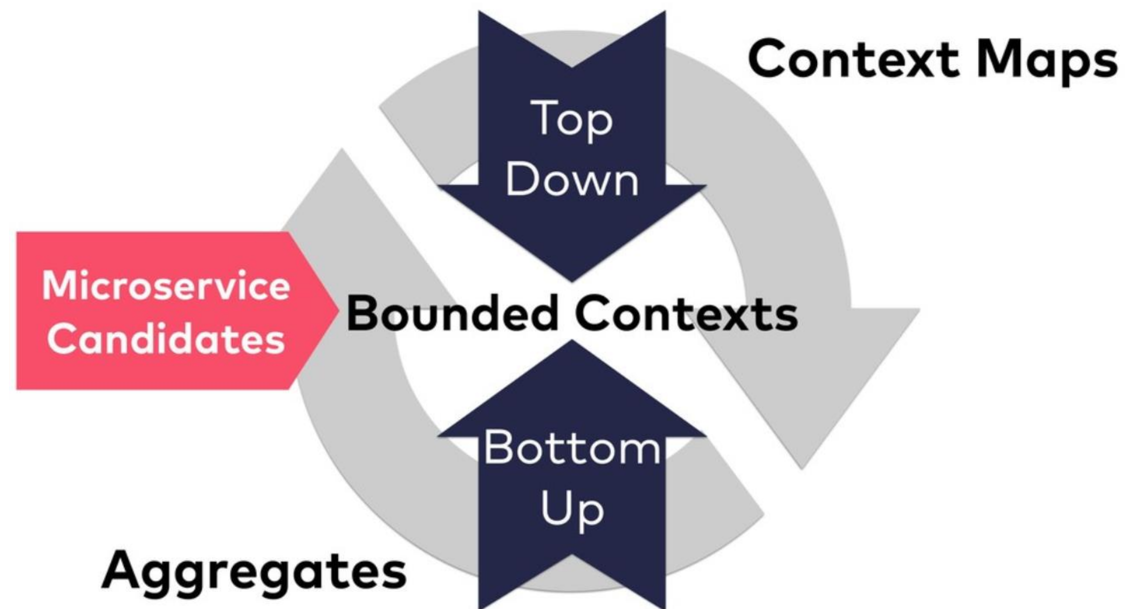
© Olaf Zimmermann, 2020.

Agenda Today (And Key Take Away Messages)

1. **Context matters**
   - One size does not fit all (top-level design heuristic: "it depends")
   - Strategic and tactic Domain-Driven Design (DDD)
   - Context Mapper DSL and tools
2. **Contracts rule**
   - Unified interfaces are great, but not enough
   - More SOA and microservices myth busting
   - Microservice Domain-Specific Language (MDSL)
3. ***Components contain (cost and risk)***
   - *Towards a context-driven, contract-first service identification method*
   - *Microservice API Patterns (MAP) to structure the solution space*
   - *(time permitting) Industry trends and resulting research questions*
     - *Microfrontends, containerization, cloud-native 12-factor applications*

© Olaf Zimmermann, 2020.

- **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

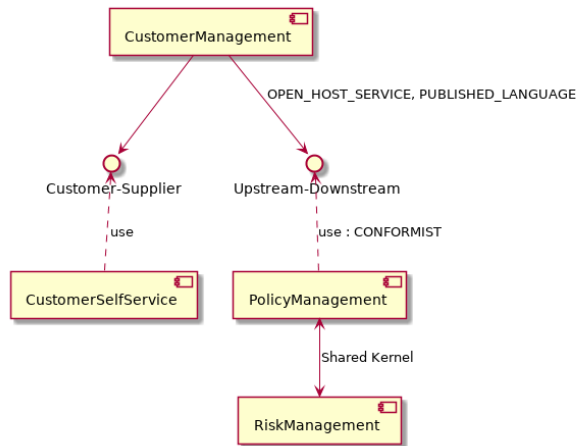


Reference: JUGS presentation, Bern/CH, Jan 9, 2020

© Olaf Zimmermann, 2020.

DDD and Service Identification/Design

https://preview.microservice-api-patterns.org/patterns/tutorials/tutorial2



Input: analysis model, NFRs

Step 0: Baseline (Starting Point)

Challenges (Tasks)

Step 1: Identification and Foundation Patterns

Step 2: Roles and Responsibilities (R)

Step 3: Basic and Composite Structures (S)

Step 4: Quality Enhancements (Q)

Step 5: Evolution Patterns

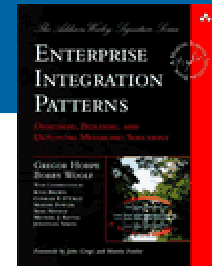
Tasks: Select pattern, refine design, refactor

Output: API contracts (here: MDSL)

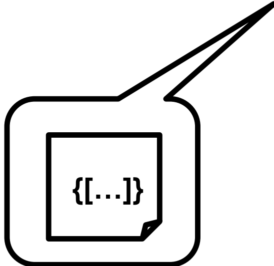
```
API description LakesideMutual
data type StatusInformation (V<bool>,L)
endpoint type CustomerManagement serves as INFORMATION_HOLDER_RESOURCE
exposes
  operation findCustomer with responsibility RETRIEVAL_OPERATION
    expecting payload V<void> // no payload
    delivering payload "customerIDList":ID*

  operation readCustomer with responsibility RETRIEVAL_OPERATION
    expecting payload "customerID":ID
    delivering payload "customerDTO":V?

  operation updateCustomer with responsibility EVENT_PROCESSOR
    expecting payload "customerDTO":V?
    delivering payload StatusInformation
```


© Olaf Zimmermann, 2020.

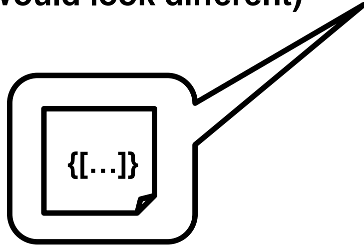
Calls to Service Operations are EIP-style Messages




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



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

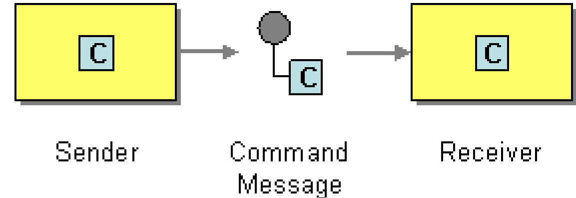


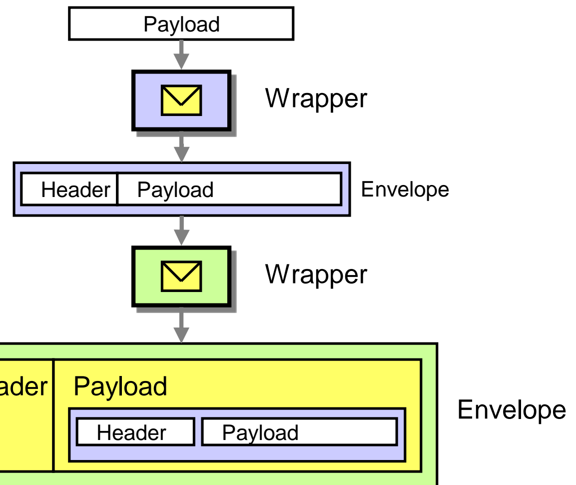
Response message structure


```
{
  "_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

© Olaf Zimmermann, 2020.

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)

© Olaf Zimmermann, 2020.

Microservices API Patterns (MAP): Pattern Index by Category

Responsibility

Endpoint Roles

- Processing Resource
- Information Holder Resource
- Transactional Data Holder
- Master Data Holder
- Static Data Holder

Structure

Representation Elements

- Atomic Parameter
- Atomic Parameter List
- Annotated Parameter Collection
- Context Representation
- Pagination

EuroPLoP 2017

Quality

Quality Management and Governance

- API Key
- Rate Limit

EuroPLoP 2018

Reference Management

- Embedded Entity
- Linked Information Holder

Evolution

- Version Identifier
- Semantic Versioning
- Two In Production
- Aggressive Obsolescence
- Experimental Preview
- Limited Lifetime Guarantee
- Eternal Lifetime Guarantee

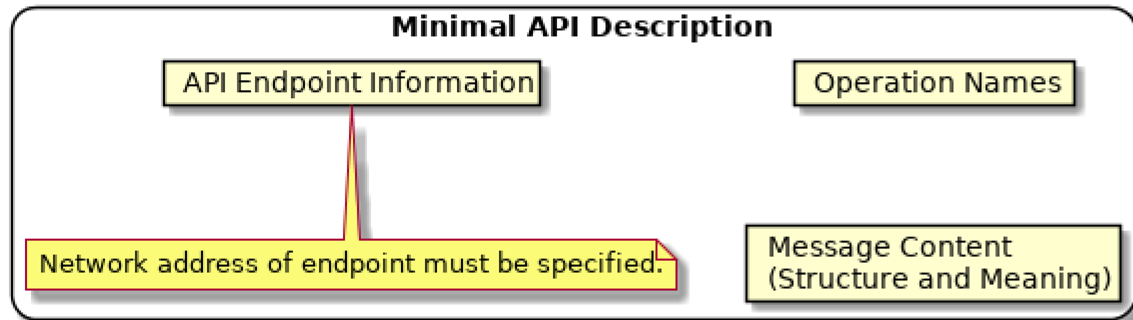
EuroPLoP 2019

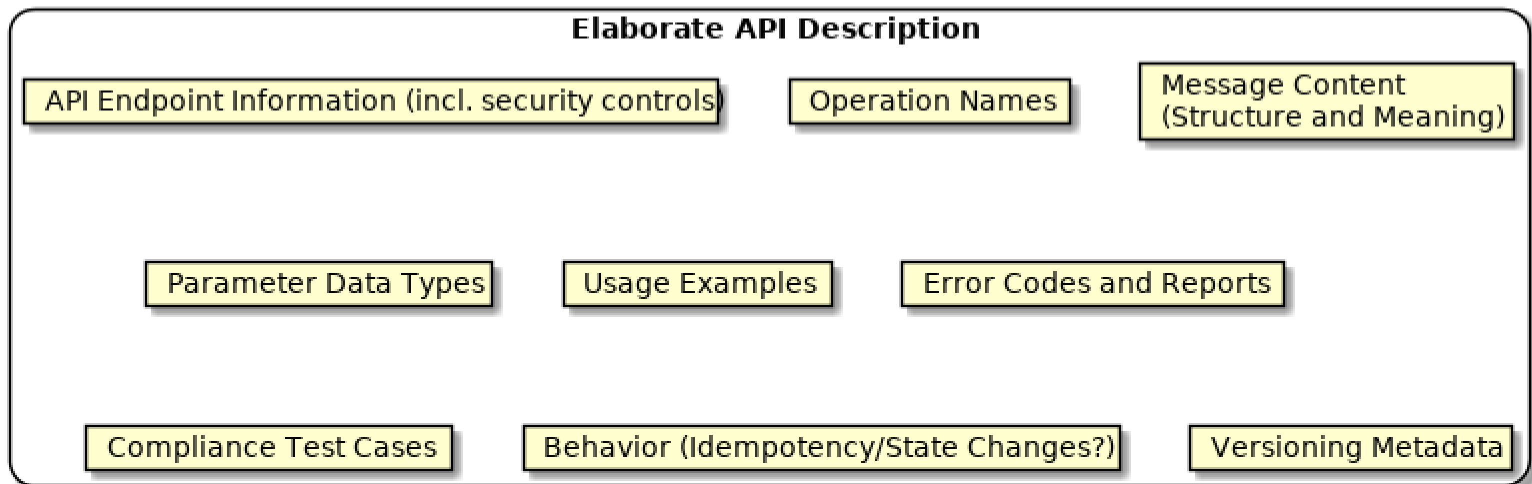
http://microservice-api-patterns.org

© Olaf Zimmermann, 2020.

API Description Pattern

- **Which knowledge should be shared between an API provider and its clients?**
- **How should this knowledge be documented?**





https://microservice-api-patterns.org/patterns/foundation/APIDescription.html

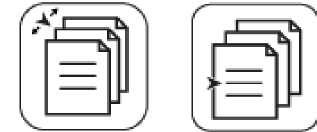
© Olaf Zimmermann, 2020.

- **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

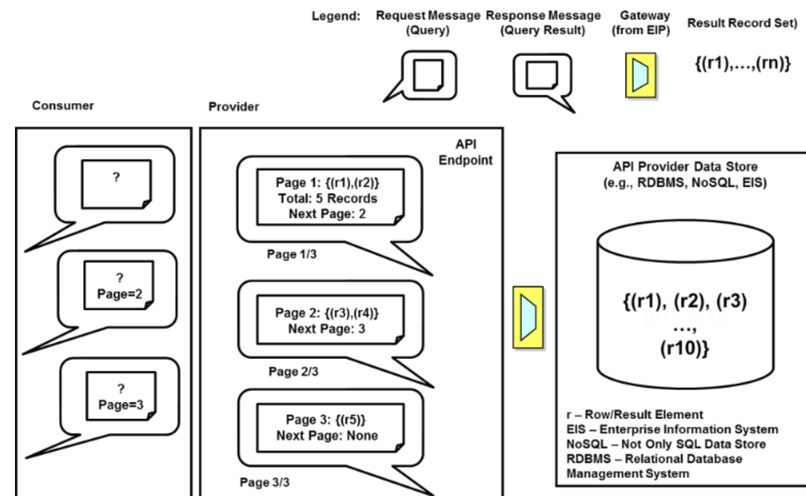
© Olaf Zimmermann, 2020.

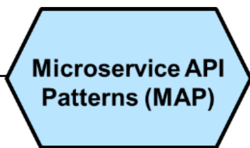
MAP Example: Pagination (2/2)

europlop



- **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





© Olaf Zimmermann, 2020.

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

Microservice API Patterns

HOME CATEGORIES PATTERN FILTERS PATTERN INDEX AUTHORS

Microservice API Patterns (MAP) take a broad view on API design and evolution, primarily focussing on message representations – the payloads exchanged when APIs are called. These payloads have *structure*. The representation elements in the payloads differ in their meanings as API endpoints and their operations have different architectural *responsibilities*. Furthermore, the chosen representation structures strongly influence the design time and runtime *qualities* of an API.

Our Microservice API Patterns capture proven solutions to design problems commonly encountered when specifying and implementing message-based APIs in terms of their structure, responsibilities, and quality.

Microservice API Patterns

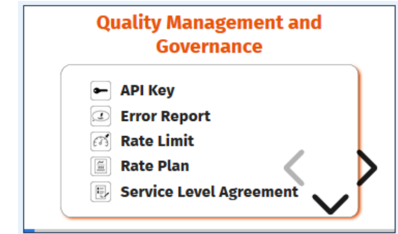
Olaf Zimmermann, Mirko Stocker, Uwe Zdun, Daniel Lübke, Cesare Pautasso

Open Overview Slide Show in New Window

© Olaf Zimmermann, 2020.

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



- ***Microservice API Patterns (MAP)* language/components**
  - Public MAP website now available in Version 1.2.1
  - 20+ patterns, sample implementation in public repo, supporting tools
- ***Microservices Domain-Specific Language (MDSL)***
  - Uses MAPs in service contracts (as decorators)
  - 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*
```

- ***Context Mapper* tool supporting strategic Domain-Driven Driven Design (DDD) and architectural refactoring**
  - Other tools emerging

- **Research areas (ZIO):**
  - Service modeling, identification, decomposition, refactoring

© Olaf Zimmermann, 2020.

Teaser Question Revisited

- **You had 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. A backend B2B channel uses message queues (RabbitMQ).**
- **What do you do (now)?**
  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 let libraries and frameworks finish the job.*
  c) *I install an API gateway product in Kubernetes and hire a sys admin, done.*
  d) *I design a service layer (Remote Facade with Data Transfer Objects) and publish an Open API Specification (f.k.a. Swagger) contract. I worry about message sizes, transaction boundaries, error handling and coupling criteria while implementing the contract. To resolve such issues, I create my own novel solutions. Writing infrastructure code and test cases is fun after all!*
  e) *I leverage Context Mapper, MDSL, MAP for API design and evolution ☺*

© Olaf Zimmermann, 2020.

FROM DOMAIN-DRIVEN DESIGN TO MICROSERVICE APIS OF QUALITY AND STYLE – BACKUP CHARTS

GI-Arbeitskreis Microservices und DevOps

Berlin, March 9, 2020

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

DDD Applied to (Micro-)Service Design ctd., Source:

TERM PROJECT

Service Decomposition as a Series of Architectural Refactorings

Author: Stefan KAPFERER

Supervisor: Prof. Dr. Olaf ZIMMERMANN

- **N. Tune and S. Millett:** Designing Autonomous Teams and Services
  - Describe how to coevolve organizational and technical boundaries to architect autonomous applications and teams based on DDD Bounded Contexts and (micro-)services.
- **O. Tigges:** How to break down a Domain to Bounded Contexts
  - Presents criteria to be used to identify Bounded Contexts.
- **R. Steinegger et al.:** Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications
  - Describes a development process to build MSA applications based on the DDD concepts, emphasizing the importance of decomposing a system in several iterations.
- **A. Brandolini:** Introducing Event Storming
  - Proposes a workshop-based technique to analyze a domain and discover bounded contexts, following events through the system/business process and detecting commands, entities (and more) along the way.

© Stefan Kapferer, Olaf Zimmermann, 2020.

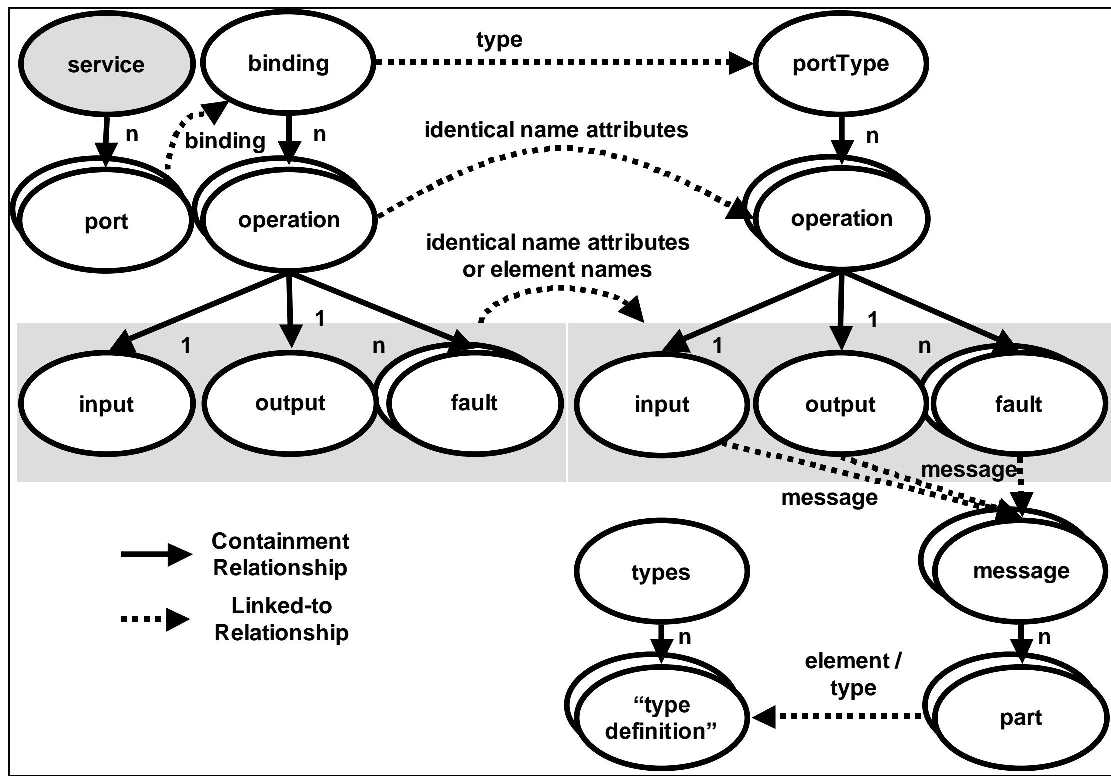
From DDD to RESTful HTTP APIs

- **“Implementing DDD” book by V. Vernon (and blog posts, presentations):**
  - *No 1:1 pass-through (interfaces vs. application/domain layer)*
  - Bounded Contexts (BCs) realized by API provider: one service API and IDE project for each team/system BC (a.k.a. microservice)
  - Aggregates supply API resources (or responsibilities) of service endpoints
  - Services donate top-level (home) resources in BC endpoint as well
  - The Root Entity, the Repository and the Factory in an Aggregate suggest top-level resources; contained entities yield sub-resources
  - Repository lookups as paginated queries (GET with search parameters)
- **Additional rules of thumb (from our experience and additional sources):**
  - Master data and transactional data go to different contexts/aggregates
  - Creation requests to Factories become POSTs
  - Entity modifiers become PUTs or PATCHes
  - Value Objects appear in the custom mime types representing resources

© Olaf Zimmermann, 2020.

SOA 1.0: WSDL (XML Language for Service Descriptions)

Web Services Description Language (WSDL)



Logical relationships between WSDL elements

- WSDL document elements
  - Type definitions and imports
  - Interface description (Port Type, Operations, Messages)
  - Extensible binding section
  - Implementation description (Ports)
- WSDL SOAP binding
  - Defines header and fault support
  - Extensibility element for addressing
- HTTP binding also defined

© Olaf Zimmermann, 2018.

Technical Service Contract in WSDL (DDD Sample Application)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://ws.handling
    <xs:element name="submitReport" type="tns:submitReport"/>
    <xs:element name="submitReportResponse" type="tns:submitReportResponse"/>
    <xs:complexType name="submitReport">
        <xs:sequence>
            <xs:element minOccurs="0" name="arg0" type="tns:handlingReport"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="handlingReport">
        <xs:sequence>
            <xs:element name="completionTime" type="xs:dateTime"/>
            <xs:element maxOccurs="unbounded" name="trackingIds" type="xs:string"/>
            <xs:element name="type" type="xs:string"/>
            <xs:element name="unLocode" type="xs:string"/>
            <xs:element minOccurs="0" name="voyageNumber" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="submitReportResponse">
        <xs:sequence/>
    </xs:complexType>
    <xs:element name="HandlingReportErrors" type="tns:HandlingReportErrors"/>
    <xs:complexType name="HandlingReportErrors">
        <xs:sequence/>
    </xs:complexType>
</xs:schema>
```

- **XML elements for operation parameters**
  - a.k.a. message parts
- **XML complex types for nontrivial DTOs**
- **XML basic types for scalar DTOs**

```
<wsdl:portType name="HandlingReportService">
  <wsdl:operation name="submitReport">
    <wsdl:input name="submitReport" message="tns:submitReport">
  </wsdl:input>
    <wsdl:output name="submitReportResponse" message="tns:submitReportResponse">
  </wsdl:output>
    <wsdl:fault name="HandlingReportErrors" message="tns:HandlingReportErrors">
  </wsdl:fault>
  </wsdl:operation>
```


© Olaf Zimmermann, 2018.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

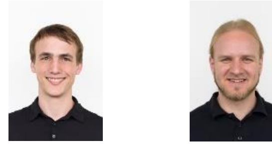
COMPUTER SCIENCE

Bachelor Thesis Fall Term 2015

Software

zühlke
empowering ideas

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….

Step 1: Analyze System

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

Coupling information is extracted from these artifacts.

	Cohesiveness	Compatibility	Constraints	Communication
Domain	Identity & Lifecycle Commonality, Semantic Proximity, Shared Owner	Change Similarity		
Quality	Latency	Consistency, Availability, Volatility	Consistency Constraint	Mutability
Physical		Storage Similarity	Predefined Service Contraint	Network Traffic Suitability
Security	Security Contextuality	Security Criticality	Security Constraint	

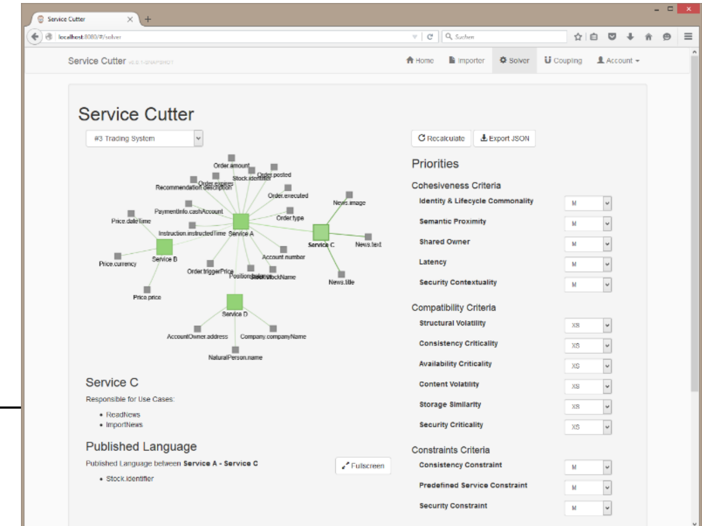
The catalog of 16 coupling criteria

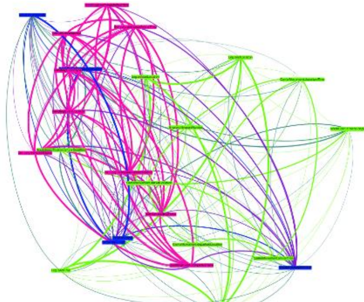
Step 2: Calculate Coupling

- 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).

Step 3: Visualize Service Cuts

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



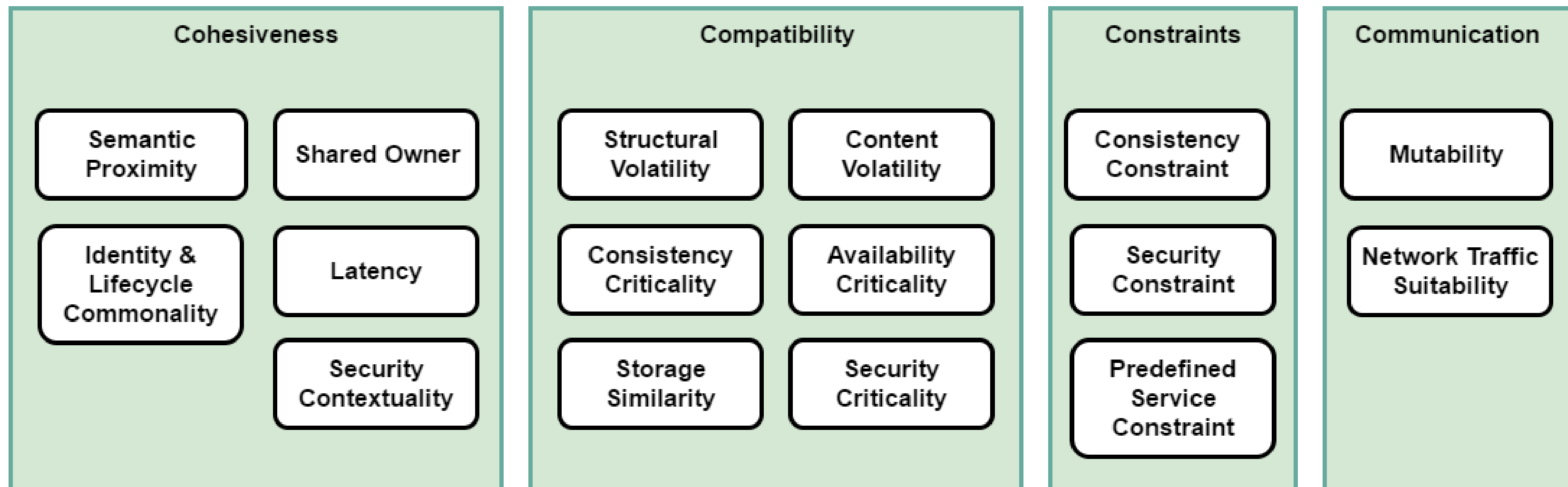


A clustered (colors) graph.

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)

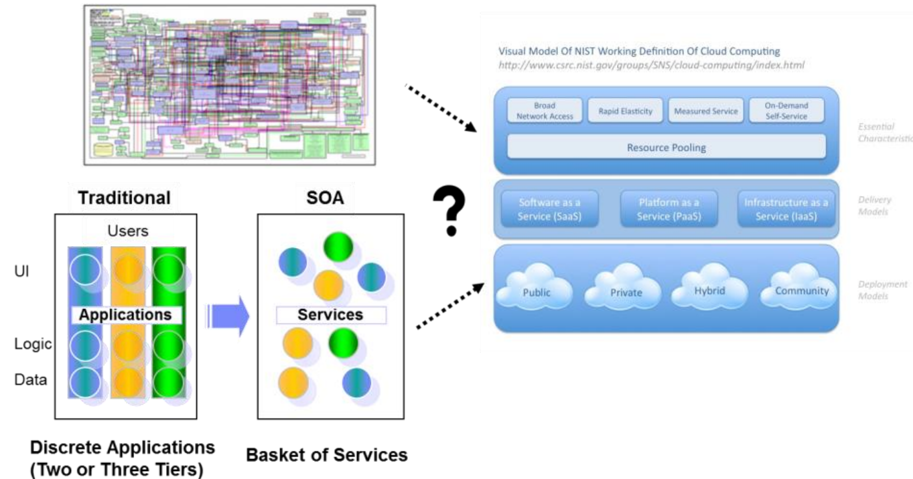


Full descriptions in CC card format: https://github.com/ServiceCutter/ServiceCutter/wiki/Coupling-Criteria

- **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)

© Olaf Zimmermann, 2020.

Open Research Problem: Refactoring to Microservices



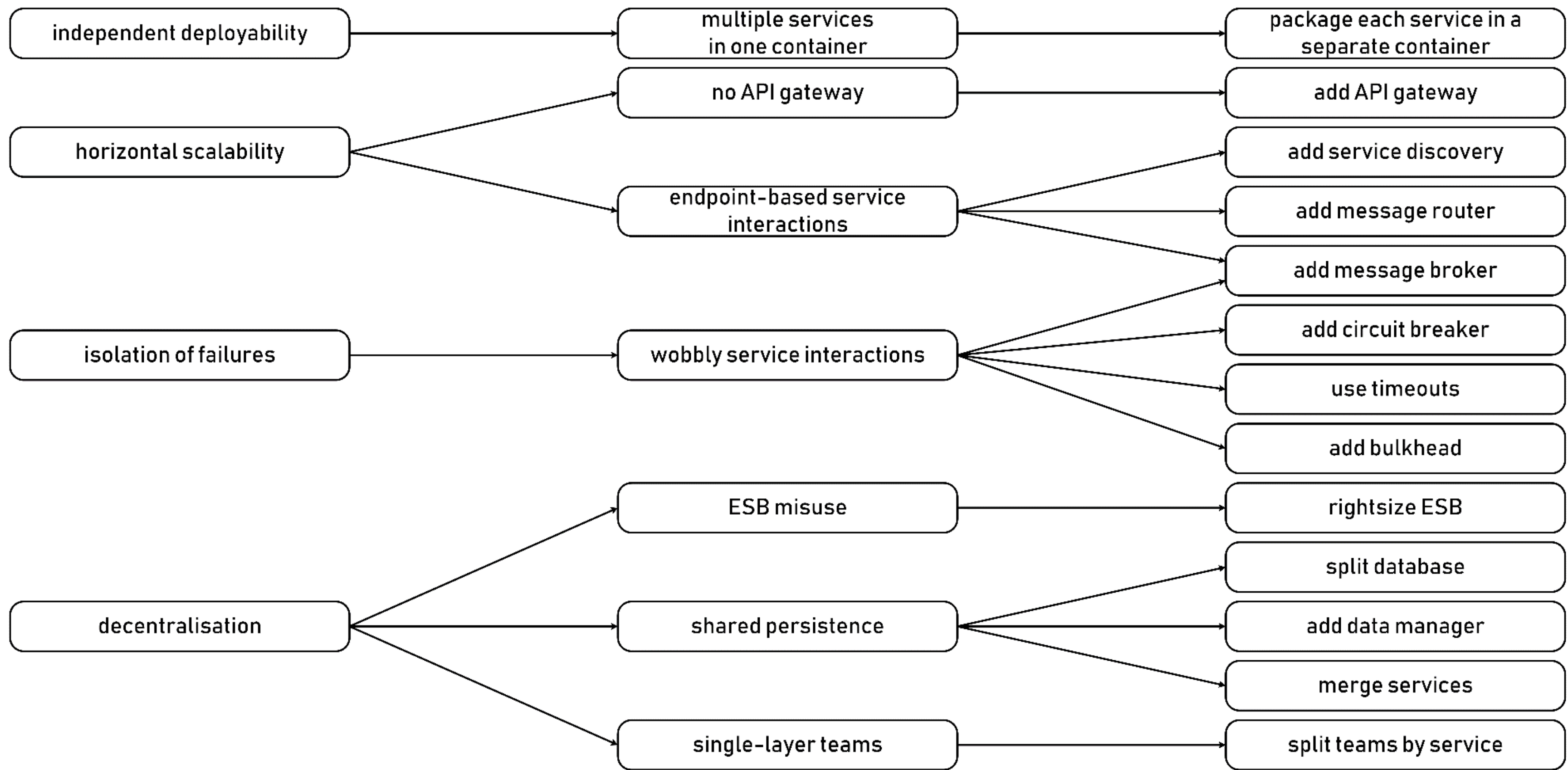
Research Questions

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?

© Olaf Zimmermann, 2020.

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)

© Olaf Zimmermann, 2020.