DOMAIN-DRIVEN SERVICE IDENTIFICATION: FROM BOUNDED CONTEXTS TO WEB APIS OF QUALITY AND STYLE

Microservices Roundtable

Zürich, 26. Februar, 2018

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

Agenda

- **Context (recap)**
  - SOA principles and microservices tenets
  - Selected Domain-Driven Design DDD patterns
    - Strategic DDD
    - Tactic DDD
- **Web API Design and Evolution (WADE) project**
  - EuroPLOP 2017: Interface Representation Patterns (IRP), incl. Pagination
  - Service Responsibilities and Granularity (Business/Technical)
  - Quality (of Service), Evolution
  - Microservices API Patterns (MAP)
- **From DDD to WADE/IRP (and the REST of SOA)**
  - Mappings
  - Practice identification

What is SOA? (Source: OOPSLA Tutorials 2004-2008)

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

- A *set of services* that a business wants to expose to their customers and partners, or other portions of the organization.
- An architectural style which requires a *service provider*, a *service requestor* (consumer) and a *service contract* (a.k.a. client/server).
  - “A service is a component with a remote interface.” (M. Fowler)
- A set of architectural patterns such as *enterprise service bus*, *service composition*, and *service registry*, promoting principles such as *modularity*, *layering*, and *loose coupling* to achieve design goals such as separation of concerns, reuse, and flexibility.
  - Services have to be discovered
  - Service invocations have to be routed, transformed, adapted
  - Smaller services have to be stitched together to implement user needs
- A *programming and deployment model* realized by standards, tools and technologies such as Web services.

Business Domain Analyst

IT Architect

Developer, Administrator

Adapted from IBM SOA Solution Stack (S3) reference architecture and SOMA method, https://www-01.ibm.com/software/solutions/soa/

Microservices – An Early and Popular Definition (2014)

Reference: http://martinfowler.com/articles/microservices.html

- **J. Lewis and M. Fowler (L/F): "[…] an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies."**
- **IEEE Software Interview with J. Lewis, M. Amundsen, N. Josuttis:**

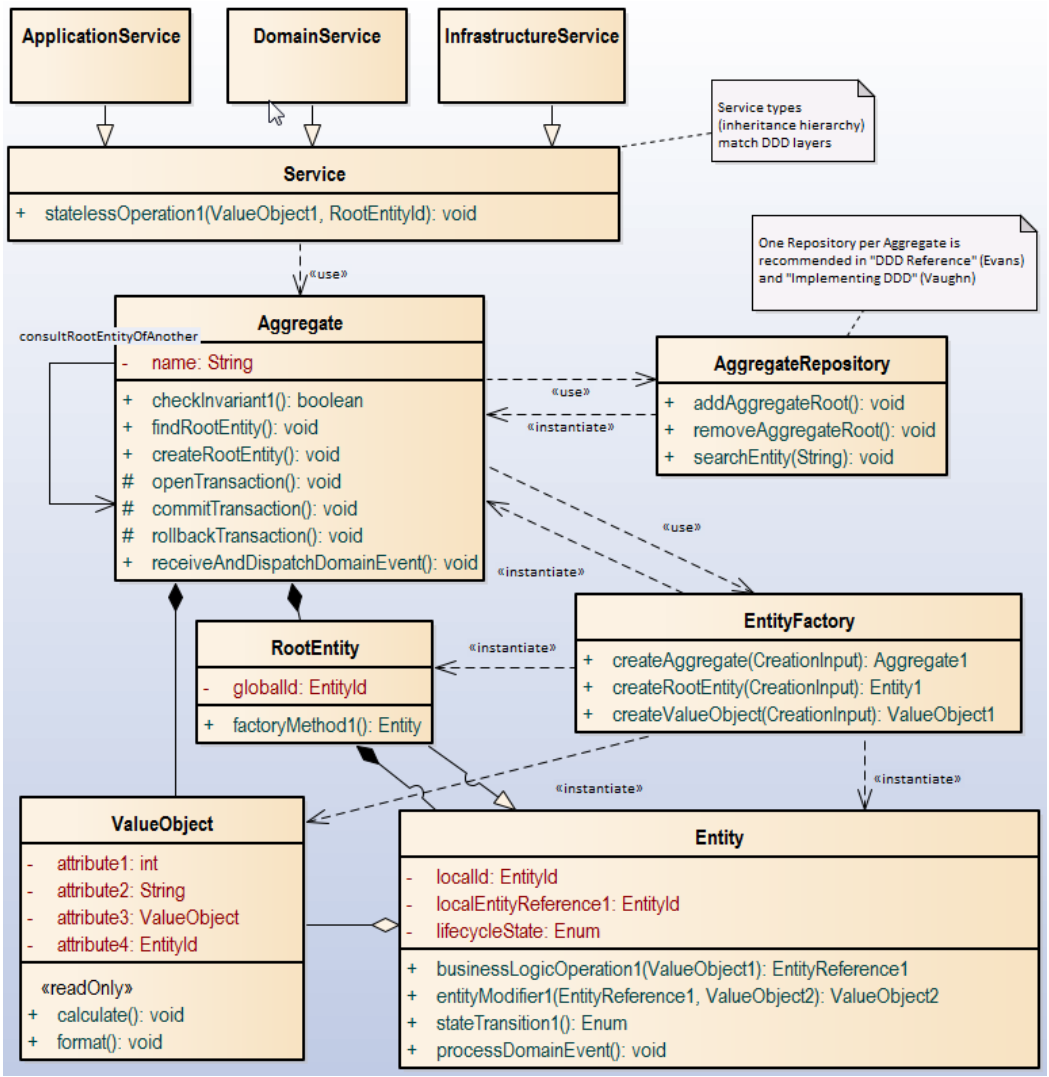
(screen captions are hyperlinks)

Seven Tenets for Microservices Approach to SOA (2016/2017)

1. ***Fine-grained interfaces*** **to single-responsibility units that encapsulate data and processing logic are exposed remotely to make them independently scalable, typically via RESTful HTTP resources or asynchronous message queues.**
2. **Business-driven development practices and pattern languages such as *Domain-Driven Design (DDD)* are employed to identify and conceptualize services.**
3. **Cloud-native application design principles are followed, e.g., as summarized in Isolated State, Distribution, Elasticity, Automated Management and Loose Coupling *(IDEAL*).**
4. **Multiple storage paradigms are leveraged (SQL and NoSQL) in a *polyglot persistence* strategy; each service implementation has its own data store.**
5. ***Lightweight containers*** **are used to deploy and scale services.**
6. ***Decentralized continuous delivery*** **is practiced during service development.**
7. **Lean, but holistic and largely automated approaches to configuration and fault management are employed within an overarching *DevOps* approach.**

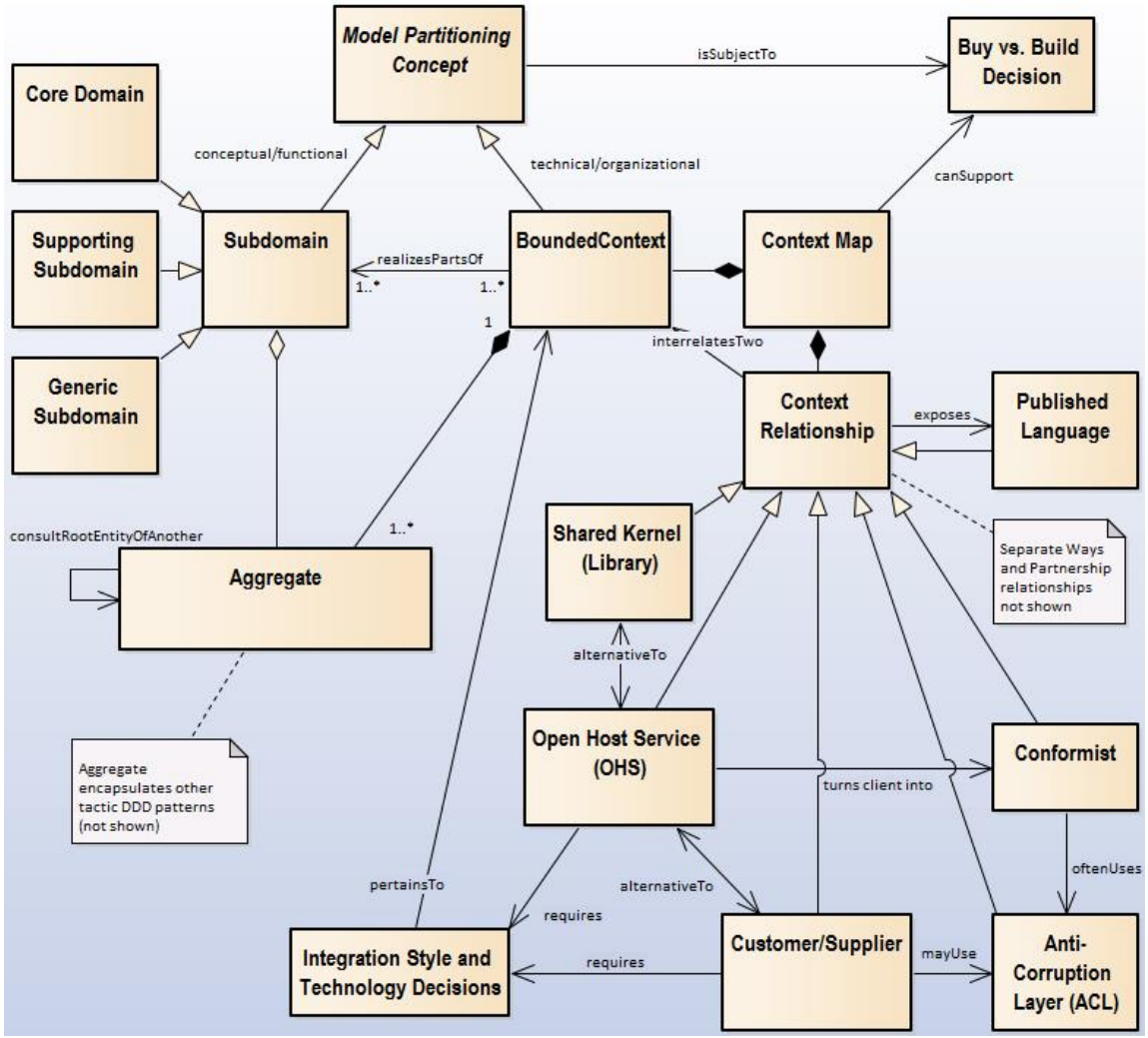
Reference: O. Zimmermann, Microservices Tenets – Agile Approach to Service Development and Deployment, Proc. Of SummerSoC 2016, Springer Computer Science – Research and Development, 2016 (CSR&D Paper).

Patterns for Tactic DDD: Meta Model (Source: ZIO)



- **Entity = "True OO"**
  - Has id
  - Has state
  - Has behavior
- **Root Entity**
  - Visible outside of Aggregate (by id)
- **Value Object**
  - No behavior
- **Aggregate**
  - Groups entities
  - Validates invariants (e.g., cross-entity business rules)

Strategic DDD Patterns: Meta Model (Source: ZIO)



- **Partitioning:**
  - Subdomain: top down
  - Bounded Context: bottom up
- **Context relationships**
  - Published Language (exposed by OHS etc.)
  - Local vs. remote?
  - Visibility?
  - (A)symmetry?
  - Amount of control and influence for client?
  - ACL as an option
- **Follow-on decisions**
  - Technology, style

SOA Principle and IDEAL Application Property: Loose Coupling

- **Practitioner heuristics (a.k.a. coupling criteria) in books, articles, blogs:**
  - SOA in Practice book by N. Josuttis, O'Reilly 2007
    - 11 types of (loose) coupling; emphasis on versioning and compatibility
  - IBM Redbook SG24-6346-00 on SOA and ESB (M. Keen et al.), IBM 2004
    - Coupled vs. decoupled continuum: semantic interface, (business) data model, QoS (e.g. transactional context, reliability), security
  - DZone, IBM developerWorks articles, InfoQ, MSDN, …
- **Academic contributions (research results):**
  - General software engineering/architecture literature since 1960s/1970s
    - Starting from D. Parnas (modularization, high cohesion/low coupling)
  - WWW 2009 presentation and paper by C. Pautasso and E. Wilde:
    - 12 facets used for a remoting technology comparison: discovery, state, granularity
  - ESOCC 2016 keynote by F. Leymann and PhD theses (e.g. C. Fehling):
    - Four types of *autonomy*: *reference (i.e., location), platform, time, format*

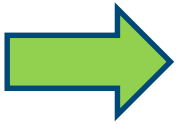
Agenda

- **Context (recap)**
  - SOA principles and microservices tenets
  - Selected Domain-Driven Design DDD patterns
    - Strategic DDD
    - Tactic DDD
- **Web API Design and Evolution (WADE) project**
  - EuroPLOP 2017: Interface Representation Patterns (IRP), incl. Pagination
  - Service Responsibilities and Granularity (Business/Technical)
  - Quality (of Service), Evolution
  - *Microservices API Patterns (MAP)*
- **From DDD to WADE/IRP (and the REST of SOA)**
  - Mappings
  - Practice identification

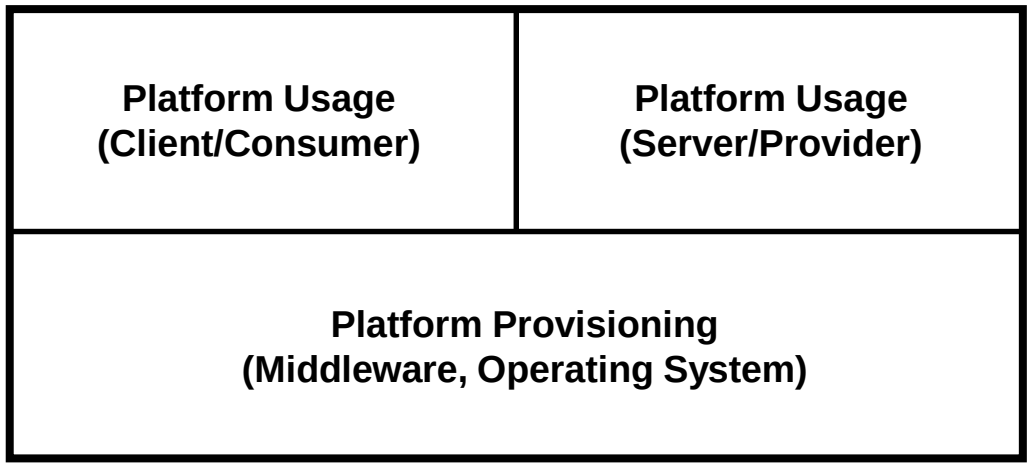
Three Perspectives on API Design: Builder(s) vs. Consumer

- **API infrastructure design is different from API creation and usage**
  - E.g. Eclipse framework team defines extension point concept
  - Plugins define extension points and use them
- **Same for SOA and REST stakeholders:**
  - Standards people and vendors vs. designers of providers and consumers

MAP (including its Interface Representation Patterns) targets platform users

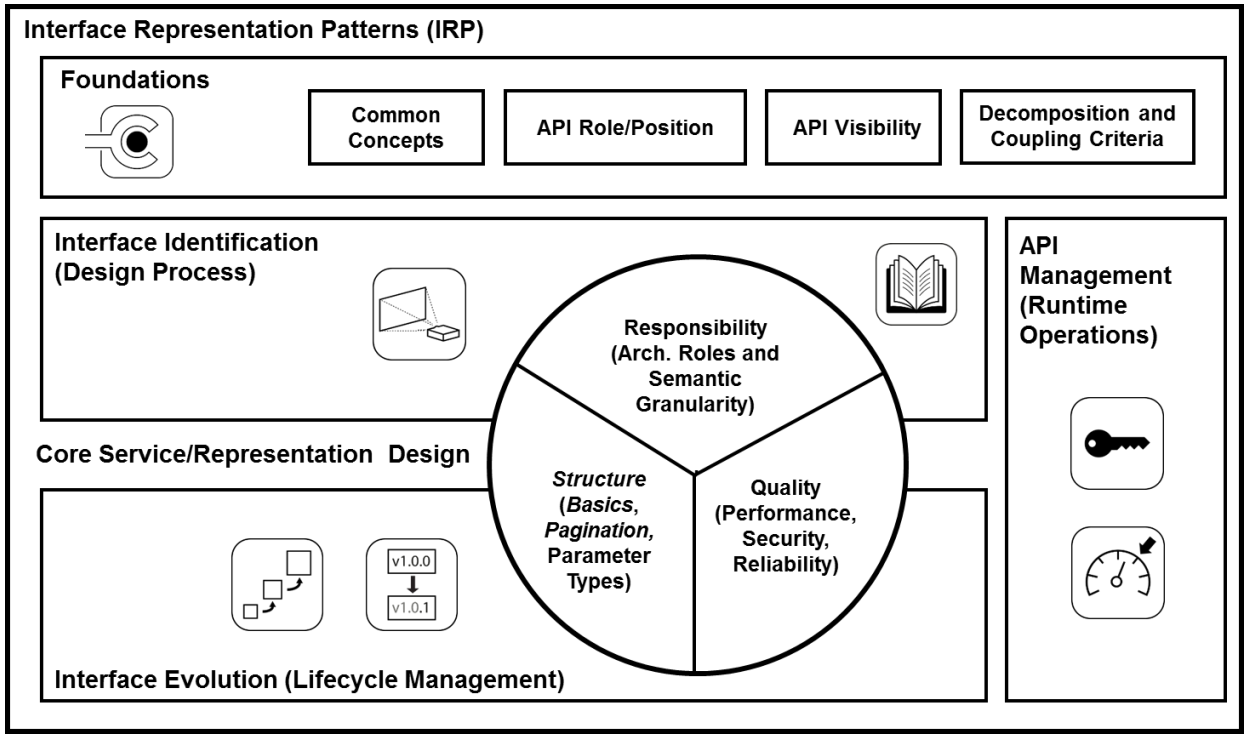


Most pattern languages describe platform design rather than platform usage (targeting platform designers rather than its users)



Towards an Microservices API Pattern Language (MAP)

- **Identfication (of API endpoints and calls a.k.a. services)**
- **Responsibilty, Structure, Quality (RSQ) patterns**
- **Evolution**

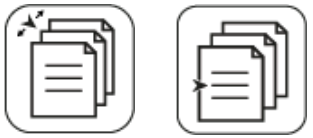


Reference: Zimmermann et al., *Interface Representation Patterns*, Proc. of EuroPLOP 2017

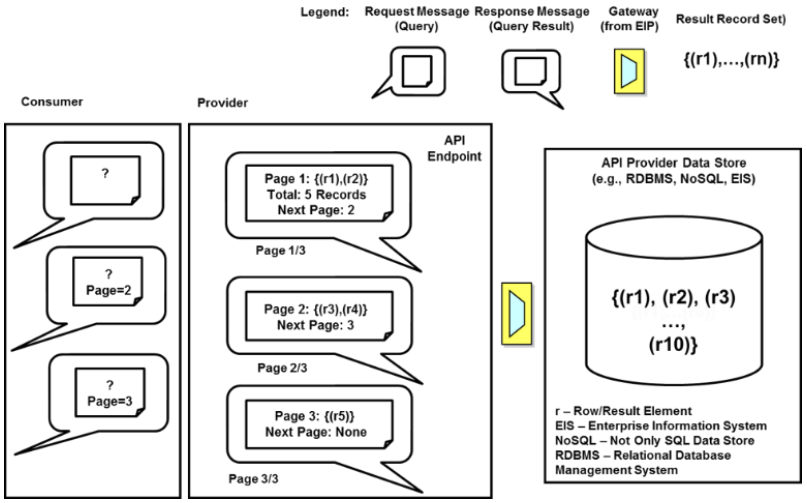
MAP Example: Pagination (1/2)

- **Context**
  - An API endpoint and its calls have been identified and specified.
- **Problem**
  - *How can a provider transmit large amounts of repetitive or inhomogeneous response data to a consumer that do not fit well in a single response message?*
- **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



Exercise: “Forces Jam”

- **Which Quality Attributes (QAs) and other requirements/constraints are the main decision drivers in microservices API design and consumption?**
  - Which *forces* should the WADE/MAP pattern language focus on?
  - What makes remote API design hard?
  - How would you justify your design decisions?

Task a): List your top three to five (optional: refine/structure a la SEI quality tree)

- **What are typical conflicts between these QAs/forces?**
  - Which tradeoffs should the MAP patern language discuss?

Task b): State at least two design time/build time QA conflicts (table/mind map)

Agenda

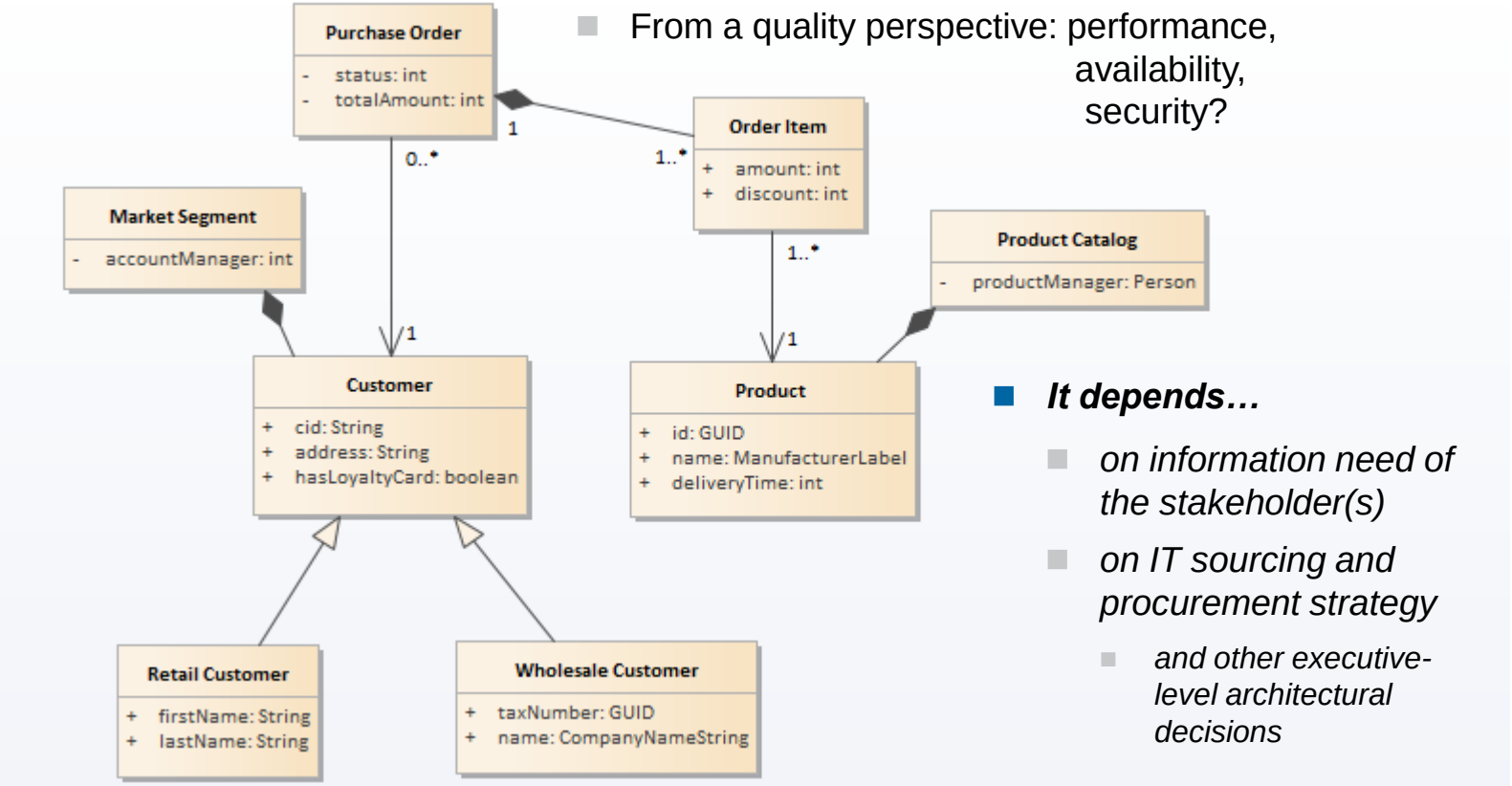
- **Context (recap)**
  - SOA principles and microservices tenets
  - Selected Domain-Driven Design DDD patterns
    - Strategic DDD
    - Tactic DDD
- **Web API Design and Evolution (WADE) project**
  - EuroPLOP 2017: Interface Representation Patterns (IRP), incl. Pagination
  - Service Responsibilities and Granularity (Business/Technical)
  - Quality (of Service), Evolution
  - Microservices API Patterns (MAP)
- **From DDD to WADE/IRP (and the REST of SOA)**
  - Mappings
  - Practice identification

Implementing Domain-Driven Design with APIs (IDD++)

- **Mentioned in IDDD book (and related blog posts and presentations):**
  - *Different layers, not 1:1 pass-through (interfaces vs. application/domain)*
  - Bounded Contexts (BCs) offered by API provider, one API endpoint and IDE project for each team/system BC (a.k.a. microservice)
  - Aggregates supply API resources or (responsibilities of) microservices
  - DDD 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 (source: ZIO, literature):**
  - Master data and transactional data go to different BCs/aggregates
  - Creation requests to Factories become POSTs
  - Entity modifiers become PUTs
  - Value Objects appear in the custom mime types representing resources

Online Shop/e-Commerce Scenario: How Many Services?

- **How loosely should the classes/services be coupled?**
  - From a functional point of view? By autonomy type?
  - From a quality perspective: performance, availability, security?

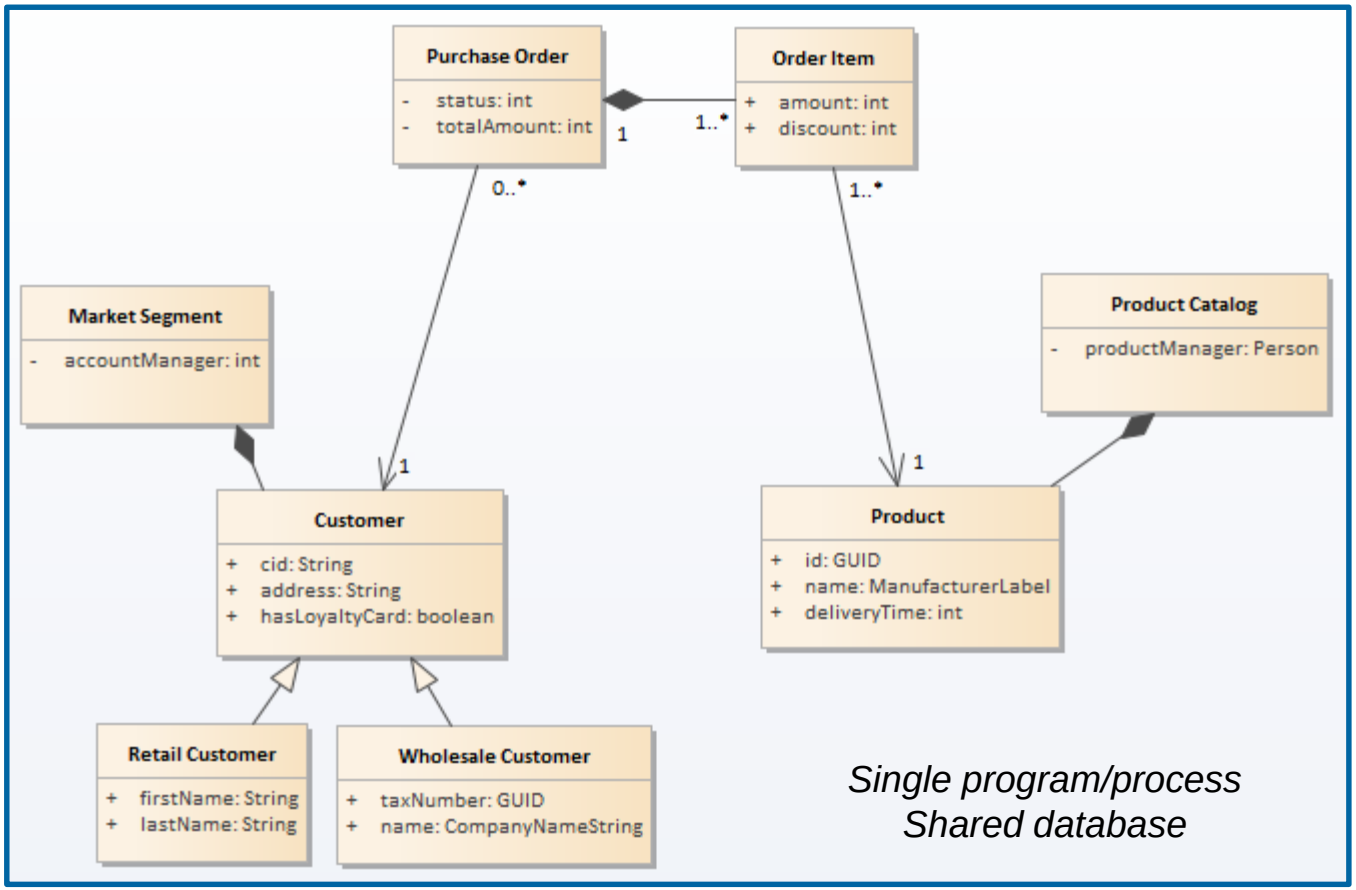


- ***It depends…***
  - *on information need of the stakeholder(s)*
  - *on IT sourcing and procurement strategy*
    - *and other executive-level architectural decisions*

Coupling Example in an Online Shop/e-Commerce (1/3)

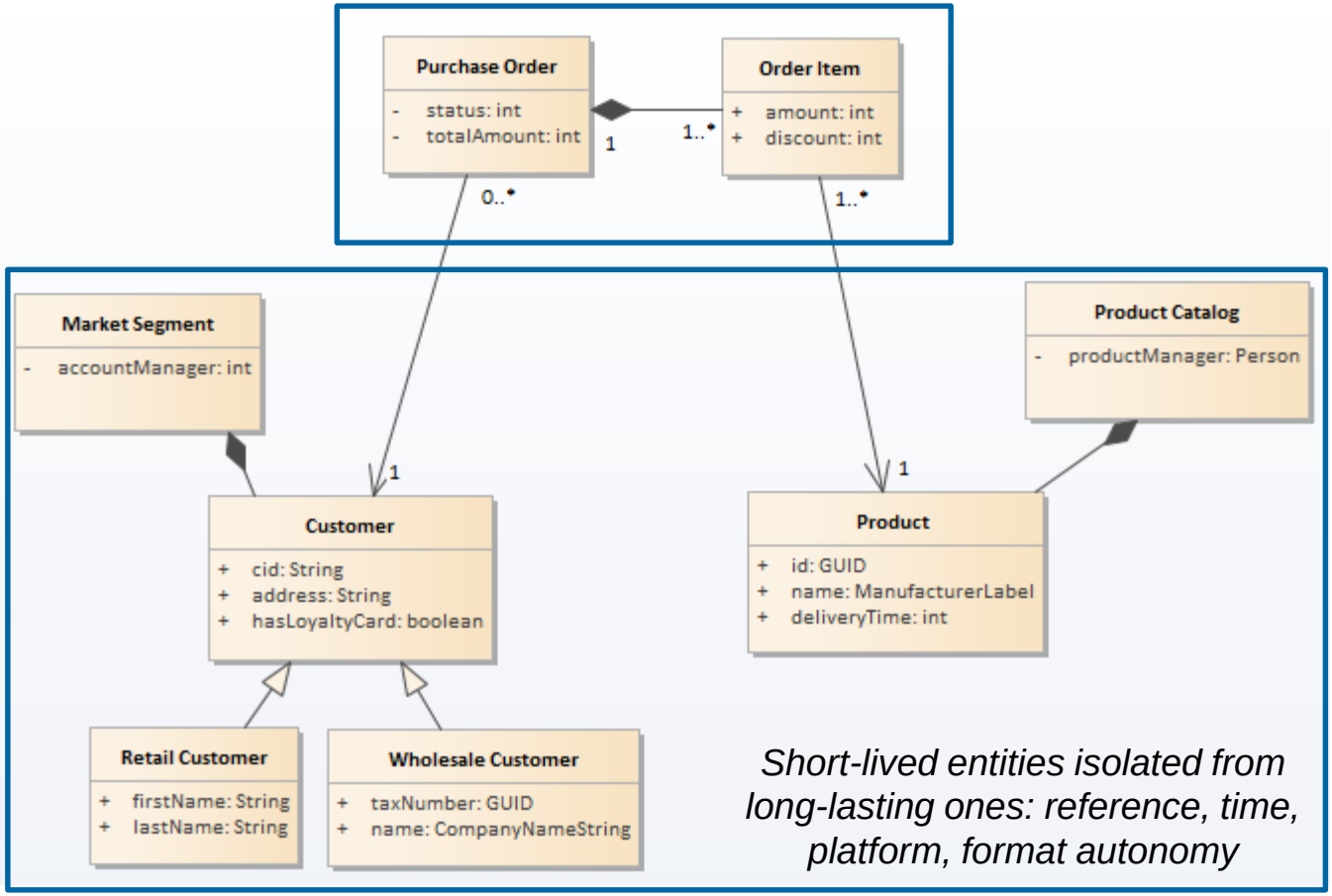
Service Boundary (Remote Interface)

- **Service Cut 0: e-commerce *monolith***



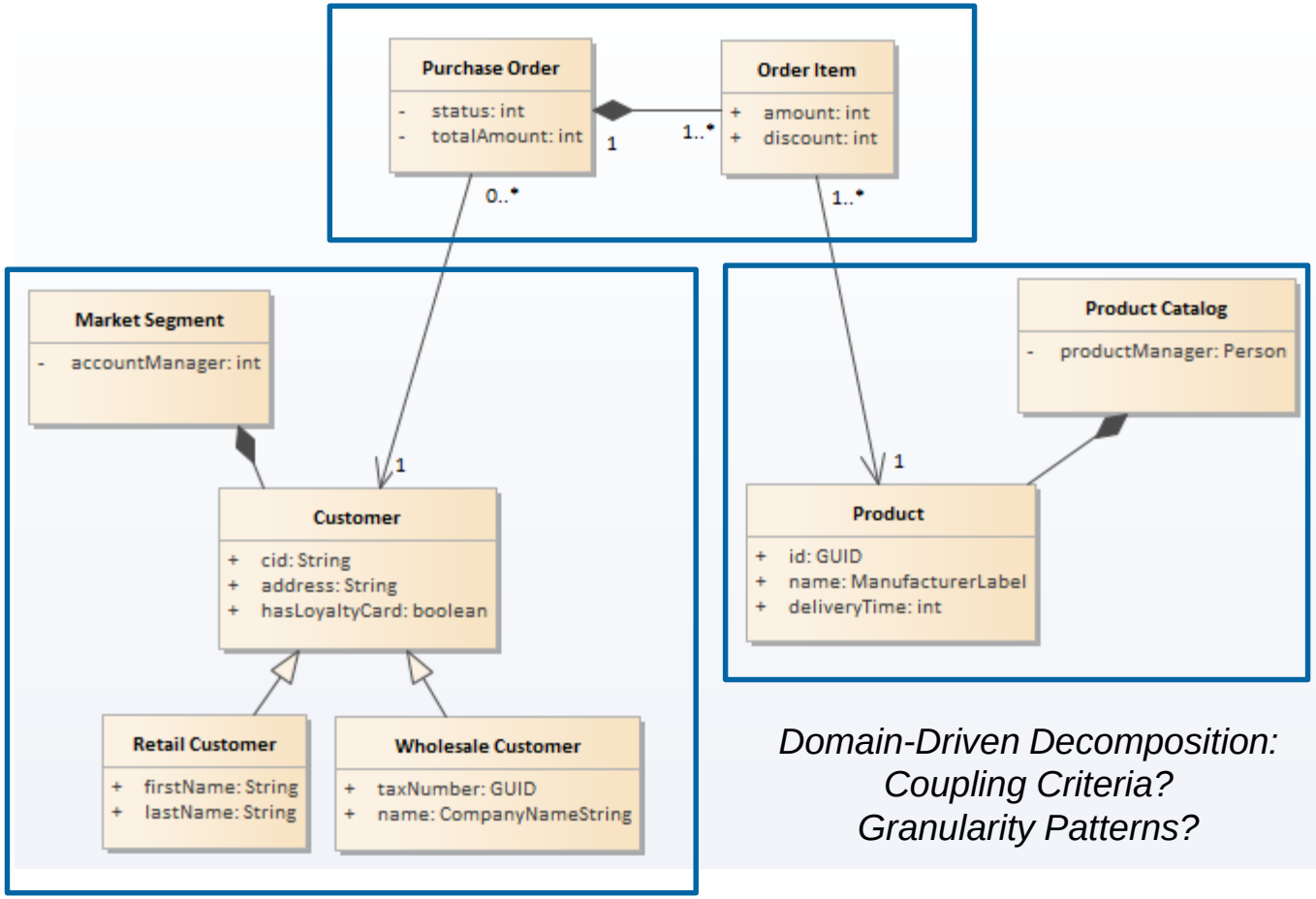
Coupling Example in an Online Shop/e-Commerce (2/3)

- **Service Cut 1: *Master Data Separation* (Order with Order Items versus Customer, Product)**



Coupling Example in an Online Shop/e-Commerce (3/3)

- **Service Cut 2: *Domain-Driven Design* Aggregates (Order, Customer, Product)**



Domain-Driven Decomposition:
Coupling Criteria?
Granularity Patterns?

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

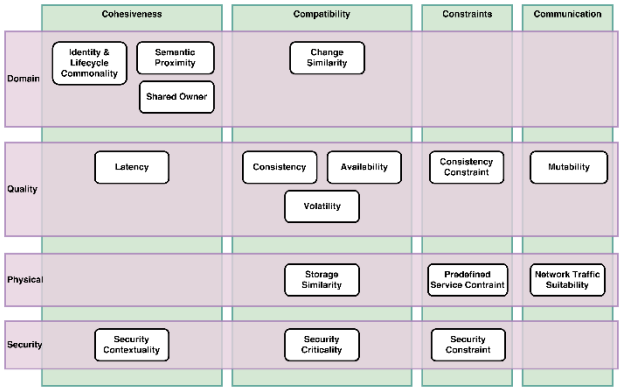
A Software Architect's Dilemma….

How do I split my system into services?

Step 1: Analyze System

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

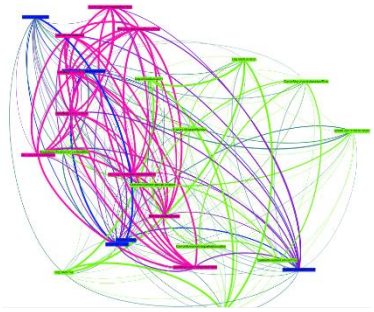
Coupling information is extracted from these artifacts.



The catalog of 16 coupling criteria

Step 2: Calculate Coupling

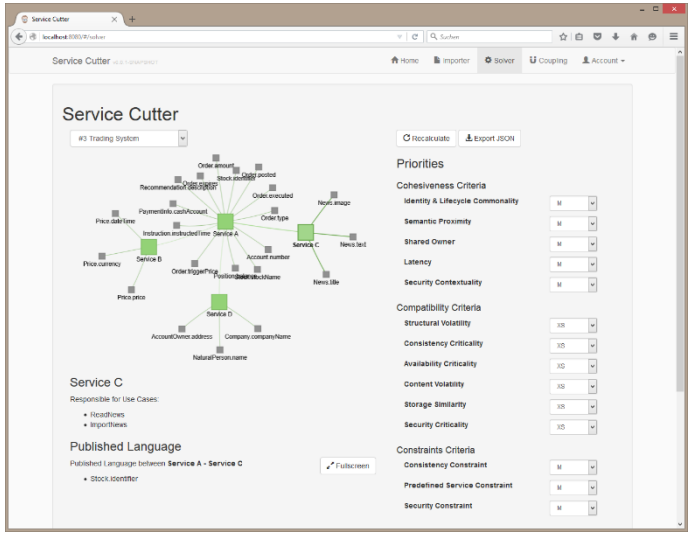
- Data fields, operations and artifacts are nodes.
- Edges are coupled data fields.
- Scoring system calculates edge weights.
- Two different graph clustering algorithms calculate candidate service cuts (=clusters).



A clustered (colors) graph.

Step 3: Visualize Service Cuts

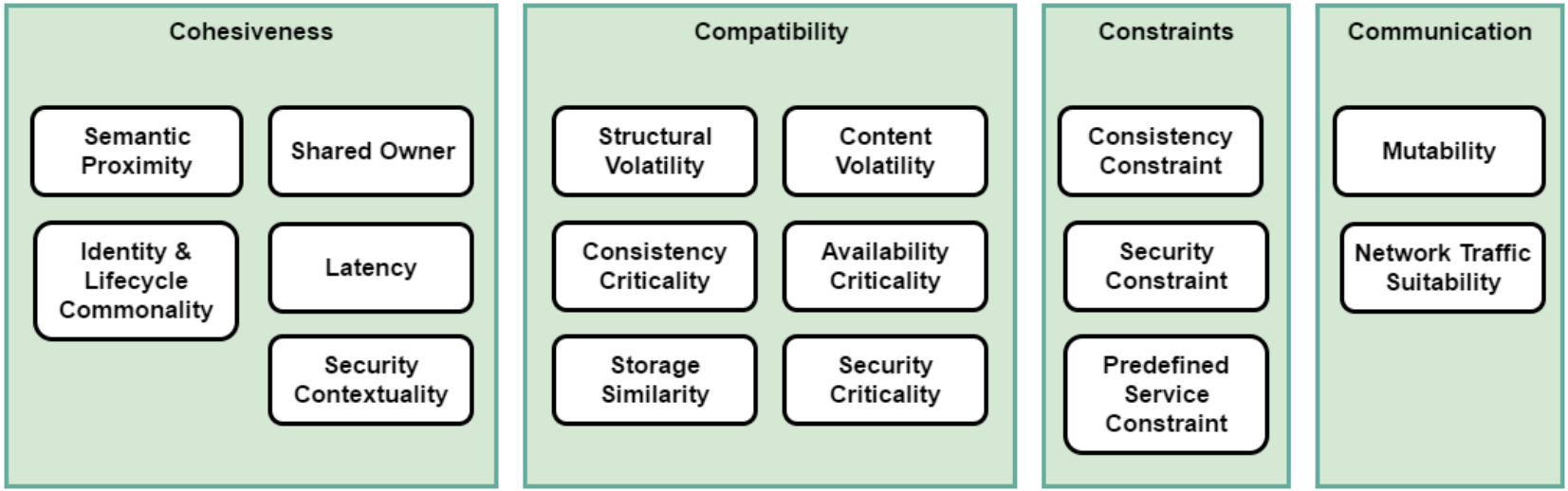
- Priorities are used to reflect the context.
- Published Language (DDD) and use case responsiblities are shown.



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)

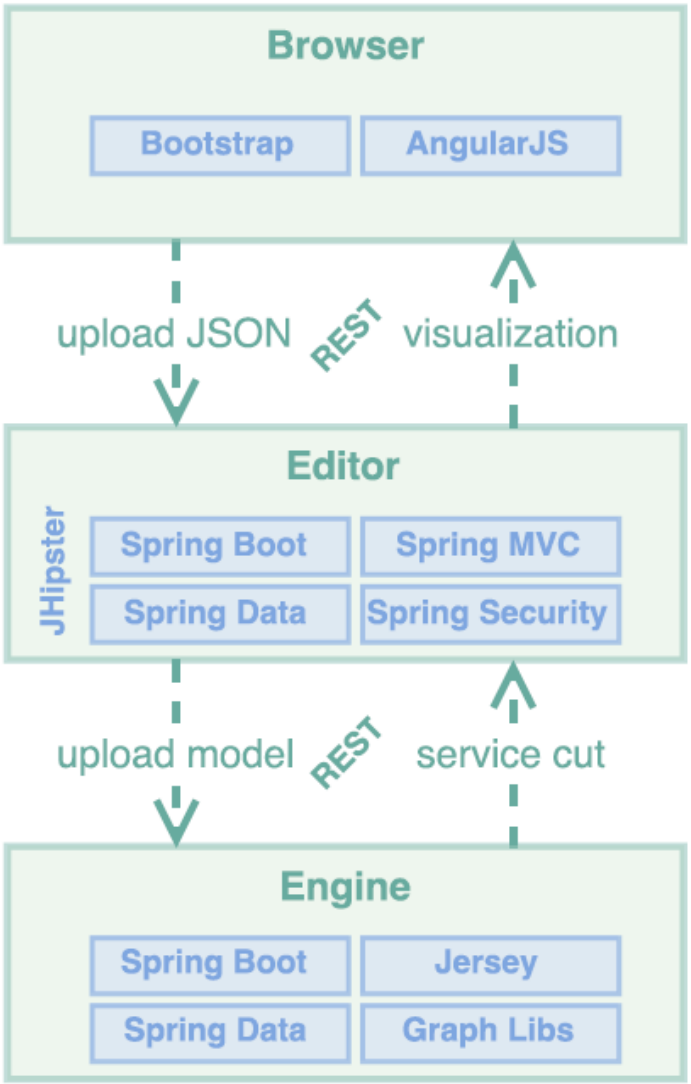
CC Card Template and Example

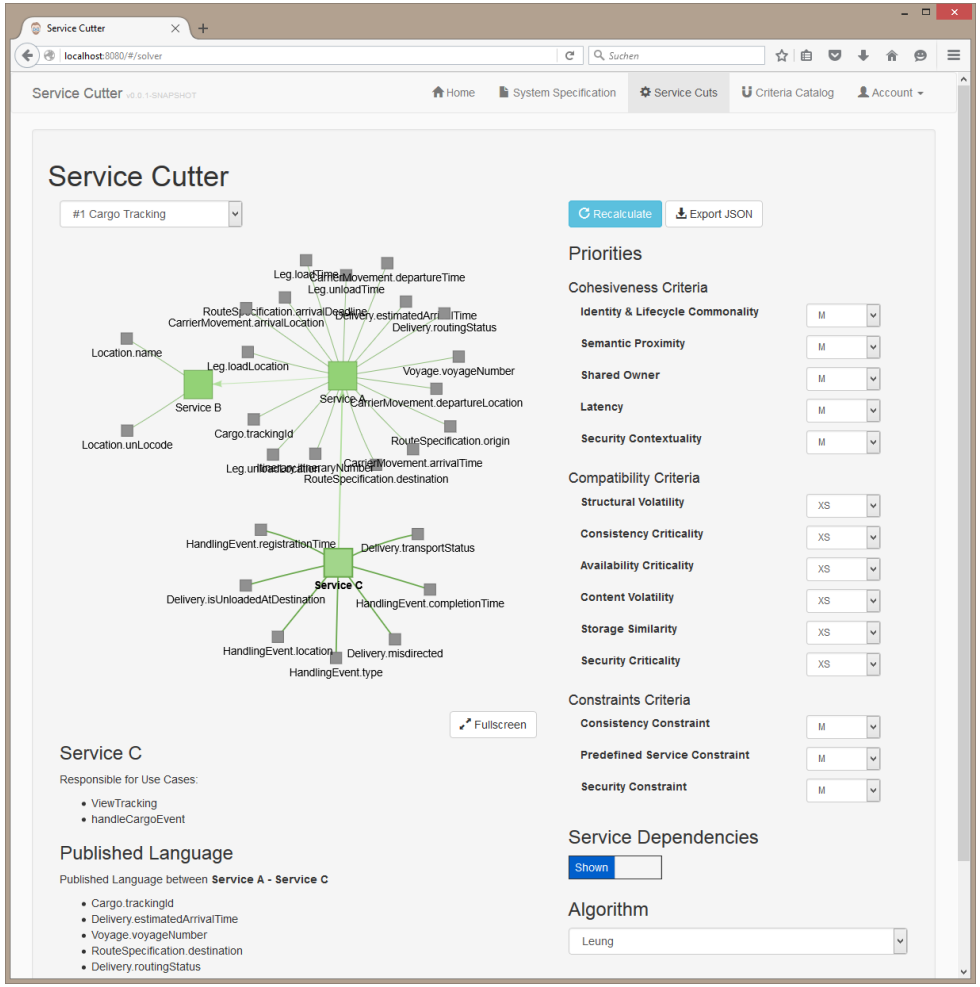
[Coupling Criteria Identifier and Name]	
Description	[A brief summary of the Coupling Criterion (CC) w.r.t. its impact on/usage of nanoentities.]
System Specification Artifacts (SSAs)	[Requirements engineering input and software architecture concepts/deliverables pertaining to this coupling criterion.]
Literature	[References to books, articles, and/or blog posts.]
Type	Cohesiveness | Compatibility | Constraint | Communication
	…es for this CC. Only applies to …, "critical", "normal", "low"].

CC-1 Identity & Lifecycle Commonality	
Description	Nanoentities that belong to the same identity and therefore share a common lifecycle.
User Representation	- UML class diagrams (Same Class, Composition, Inheritance) - Domain-D
Literature	Entity defin *Some object* *They repres* *and often a*
Type	Cohesivenes
Perspective	Domain
Characteristics	n/a

CC-2 Semantic Proximity	
Description	Two nanoentities are semantically proximate when they have a semantic connection given by the business domain. The strongest indicator for semantic proximity is coherent (joint) access of/to nanoentities within the same use case.
User Representations (SSAs)	– Coherent access to or updates of nanoentities in use cases. – Aggregation or association relationships in an entity-relationship model.

Service Cutter Tool – Architecture and User Interface





Presentation/Position Summary (ZIO)

- **Services are here to stay, but microservices do not constitute a new style**
  - Microservices evolved as an implementation approach to SOA that leverages recent advances in agile practices, cloud computing and DevOps
  - Microservices Architecture (MSA) constrains the SOA style to make services independently deployable and scalable (e.g., via decentralization)
  - Domain-Driven Design (DDD) is one of many ways to get to service and API design of quality and style
- **There is no single definite answer to the "what is the right granularity?" question, which has several context-specific dimensions and criteria**
  - Many forces apply, often conflicting
- **Platform-independent service design can benefit from patterns**
  - Interface Representation Patterns such as Pagination
  - Responsibility roles, quality improvement patterns such as Wish List
- **APIs should stick to their POINT when being EXPOSed**

Microservices – Literature and Resources

- **“Building Microservices”, S. Newman (O’Reilly 2016)**
  - Sample chapters available online (free of charge)
- **“Microservices” (auf deutsch), E. Wolf, dpunkt 2016**
  - http://dpunkt.de/a2016_downl/Microservices.pdf
- **InfoQ Microservices zone**
  - http://www.infoq.com/microservices
- **Microservices pattern languages (emerging):**
  - http://microservices.io/patterns/microservices.html
  - http://blog.arungupta.me/microservice-design-patterns/
  - http://samnewman.io/patterns/
- **SEI SATURN 2015 workshop**
  - https://github.com/michaelkeeling/SATURN2015-Microservices-Workshop