SERVICE DESIGN AS A SET OF RECURRING ARCHITECTURAL DECISIONS: PRINCIPLES, PATTERNS AND PROJECT LESSONS

4th Computer Science Conference for University of Bonn Students (CSCUBS)

Bonn, May 17, 2017

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

Abstract

- **Service-oriented computing is a key enabler for major trends such as cloud computing, Internet of things, and digital transformation. About a decade after the first wave of Service-Oriented Architecture (SOA) concepts reached a plateau of maturity, microservices are currently emerging as a state-of-the-art implementation approach to SOA that leverages recent advances in software engineering such as domain-driven design, continuous delivery and deployment automation.**
- **However, (micro-)service interface design remains a challenge due to the fallacies of distributed computing. Service designers seek design guidance and reusable architectural knowledge for this problem domain.**
- **This presentation first derives the principles and patterns defining the SOA style from an industrial case study. Next, it establishes seven corresponding microservices tenets. The presentation then reports on the ongoing compilation of a service design pattern catalog and discusses tool support for pattern selection and other architectural decisions. It concludes with a reflection on research challenges and problems in service-oriented computing, potential contributions from other fields, as well as general lessons learned from industrial and academic projects.**

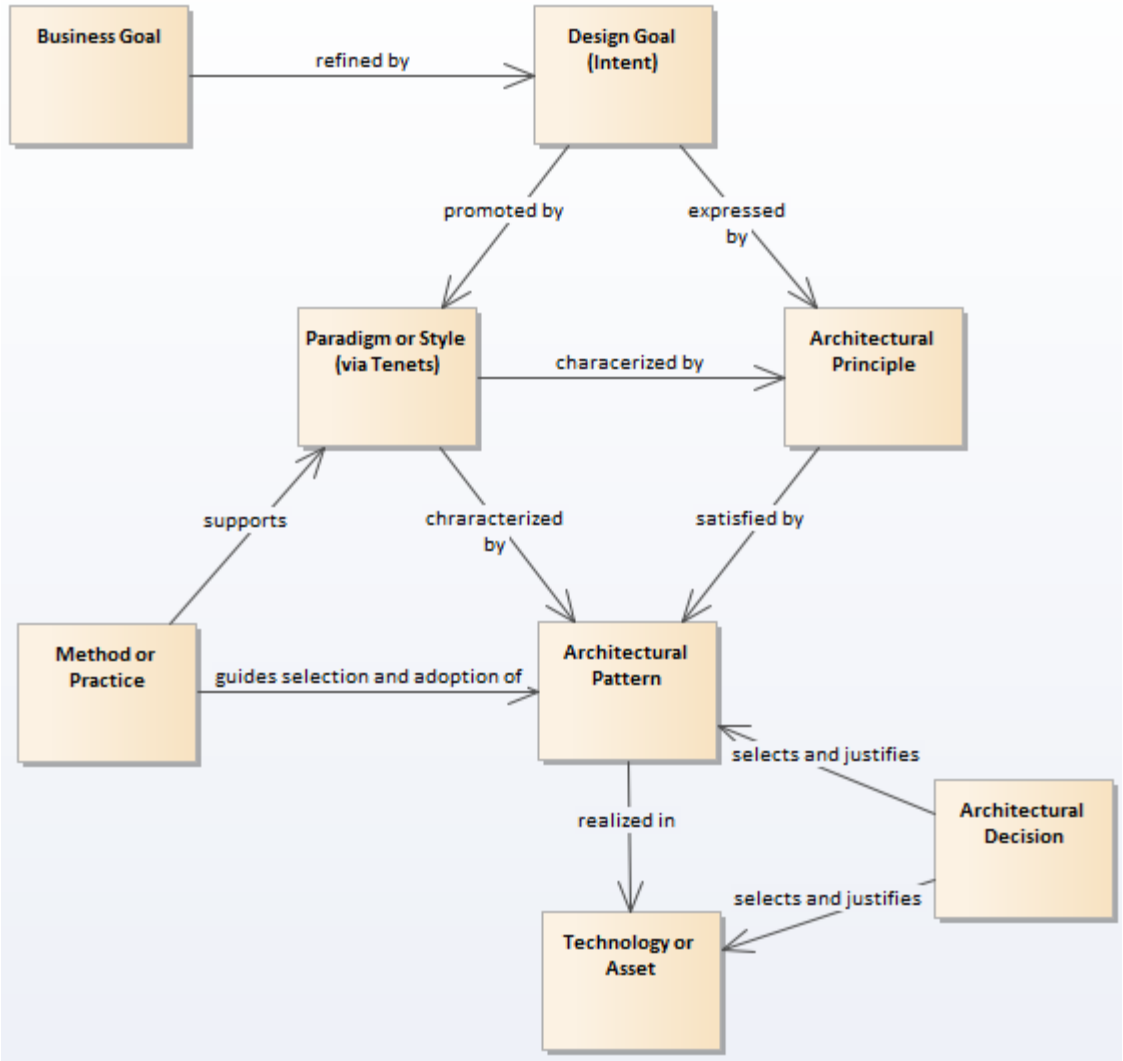
ZIO Past and Present

The Open Group Open Certified Distinguished Architect

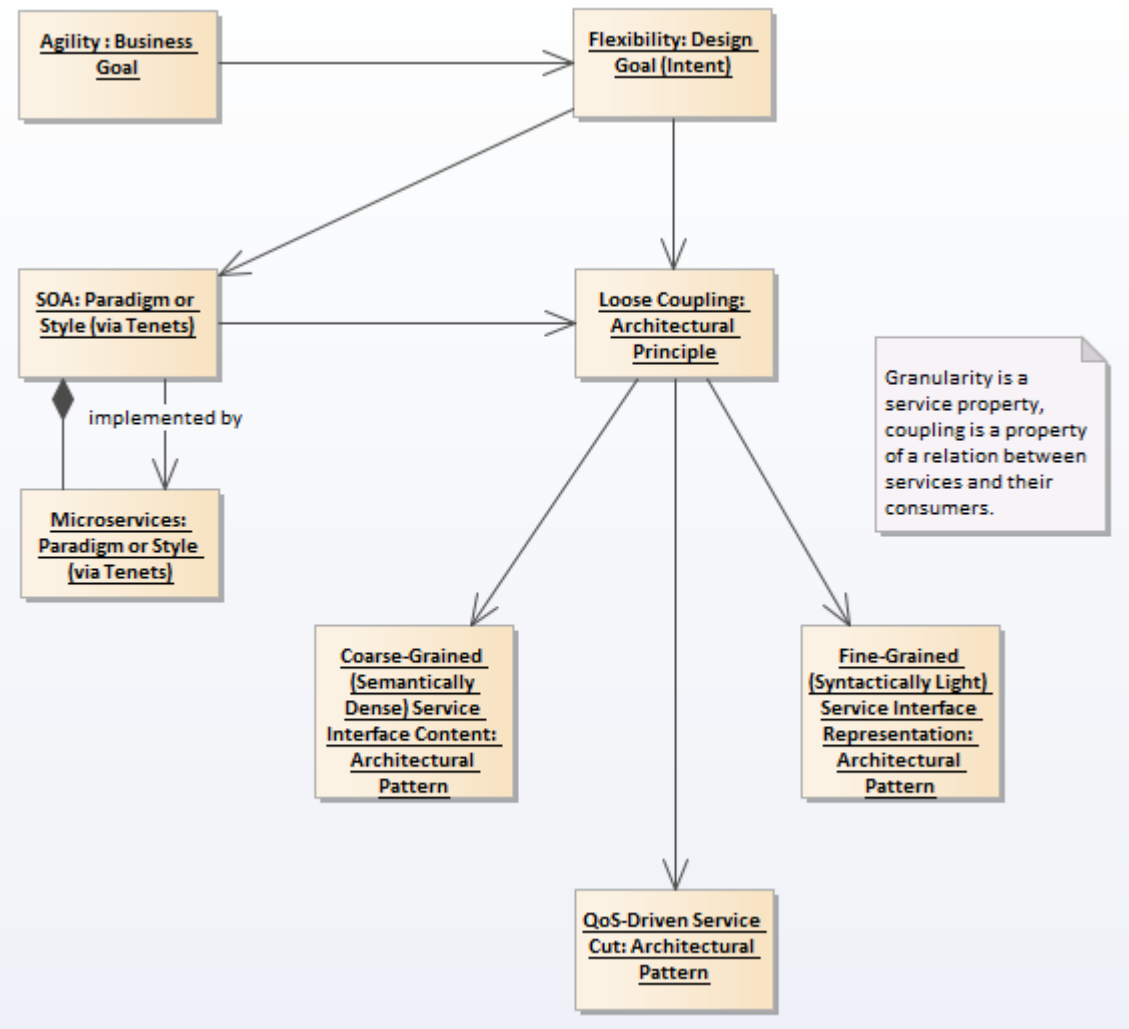
IT Architecture Chief/Lead IT Architect

- **Research & development and professional services since 1994**
  - em. IBM Solution Architect & Research Staff Member
    - Systems & Network Management, J2EE, Enterprise Application Integration/SOA
  - em. ABB Senior Principal Scientist
    - Enterprise Architecture Management/Legacy System Modernization/Remoting
- **Selected industry projects and coachings**
  - Product development and IT consulting (middleware, SOA, information systems, SE tools); first IBM Redbook on Eclipse/Web Services (2001)
  - Tutorials: UNIX/RDBMS, OOP/C++/J2EE, MDSE/MDA, Web Services/XML
- **Focus @ HSR: design of distributed/service-oriented systems**
  - Cloud computing, Web application development & integration (runtime)
  - Model-driven development, architectural decisions (build time)
  - (Co-)Editor, Insights column, IEEE Software
  - PC member, e.g., ECSA, ESOCC, WICSA, SATURN, SummerSoC

- ***Business goals and design goals***
- **Paradigms (defined by tenets)**
- **Principles**
- **Patterns**
- **Decisions**
- **Methods, practices, tools**



Summary of Key Messages (of Parts 1 to 4 of this Presentation)

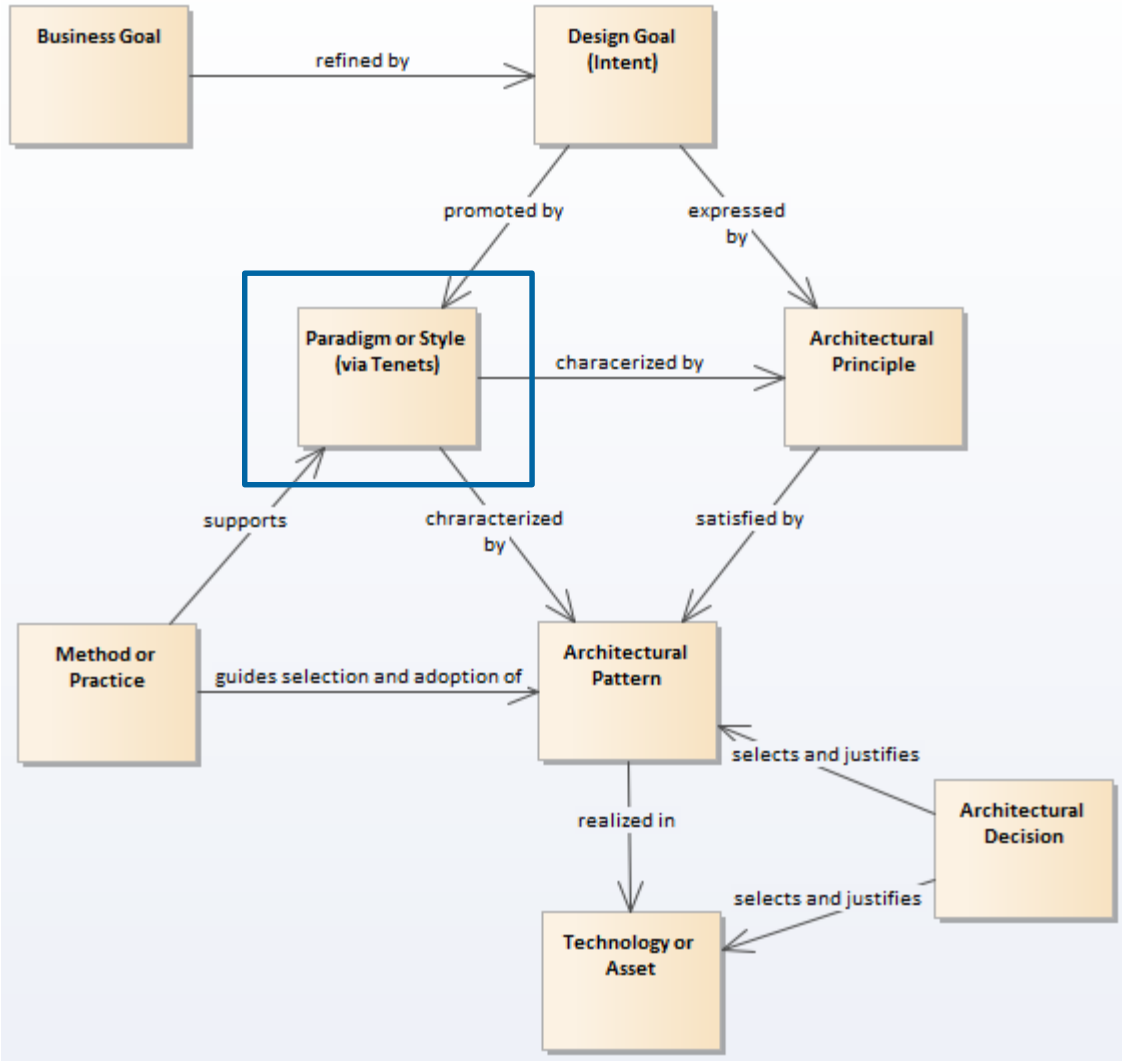


- **To follow:**
  - Industry case studies
  - SOA style definition
  - Microservices tenets
  - Loose coupling principle
    - 4 types
  - Granularity patterns
    - 3 dimensions
  - Architectural decisions
    - ADMentor tool

Agenda (“3P++”)

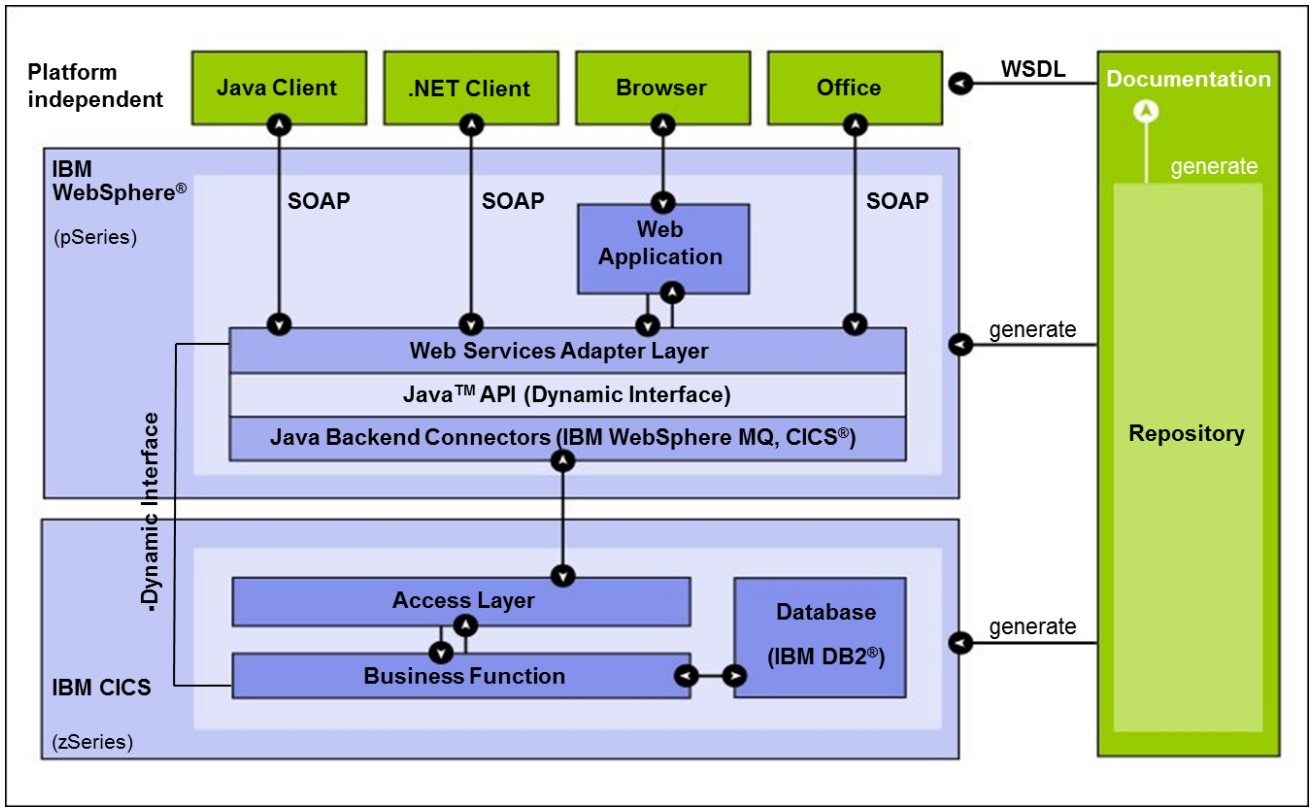
1. **Introduction to Service-Oriented Computing Paradigms**
   - Service-Oriented Architecture (SOA) style (deduction from examples)
   - Microservices tenets: agile approach to service realization
2. **Architectural Principles**
   - IDEAL cloud application architectures
   - Loose coupling, coupling criteria
3. **Interface Representation Patterns (IRP)**
   - Service Granularity (Business/Technical), Quality of Service
   - Pagination
4. **Architectural Decision Making, Capturing, and Sharing**
   - Y-statements, ADMentor tool
5. **Lessons learned from Projects in Industry and Academia**
   - Research challenges and vision

- **Business goals and design goals**
- ***Paradigms* (defined by tenets)**
- **Principles**
- **Patterns**
- **Decisions**
- **Methods, practices, tools**



Sample Information System: Financial Services (Retail Banks)

Reference: IBM, ACM OOPSLA 2004

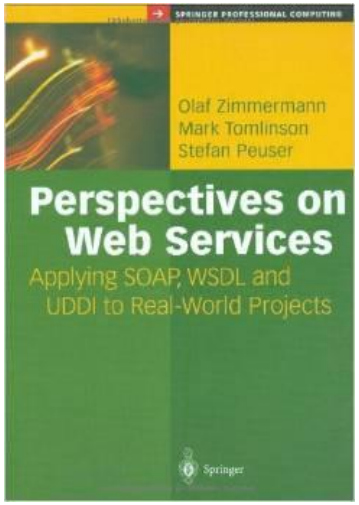


- **Information systems support – and partially automate – business processes (a.k.a. enterprise applications) to increase profit and cut cost**
  - E.g. in banking (assess credit risk), insurance (check claim), logistics, …

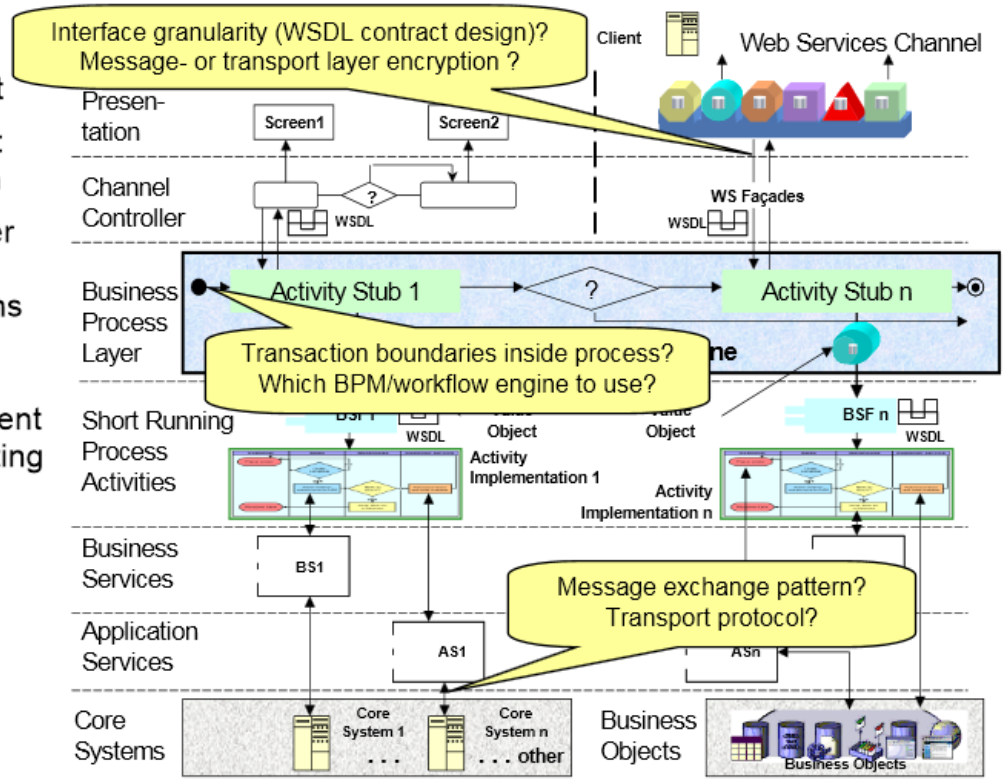
Enterprise Application in Telecommunications – IT Architect's View

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



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.

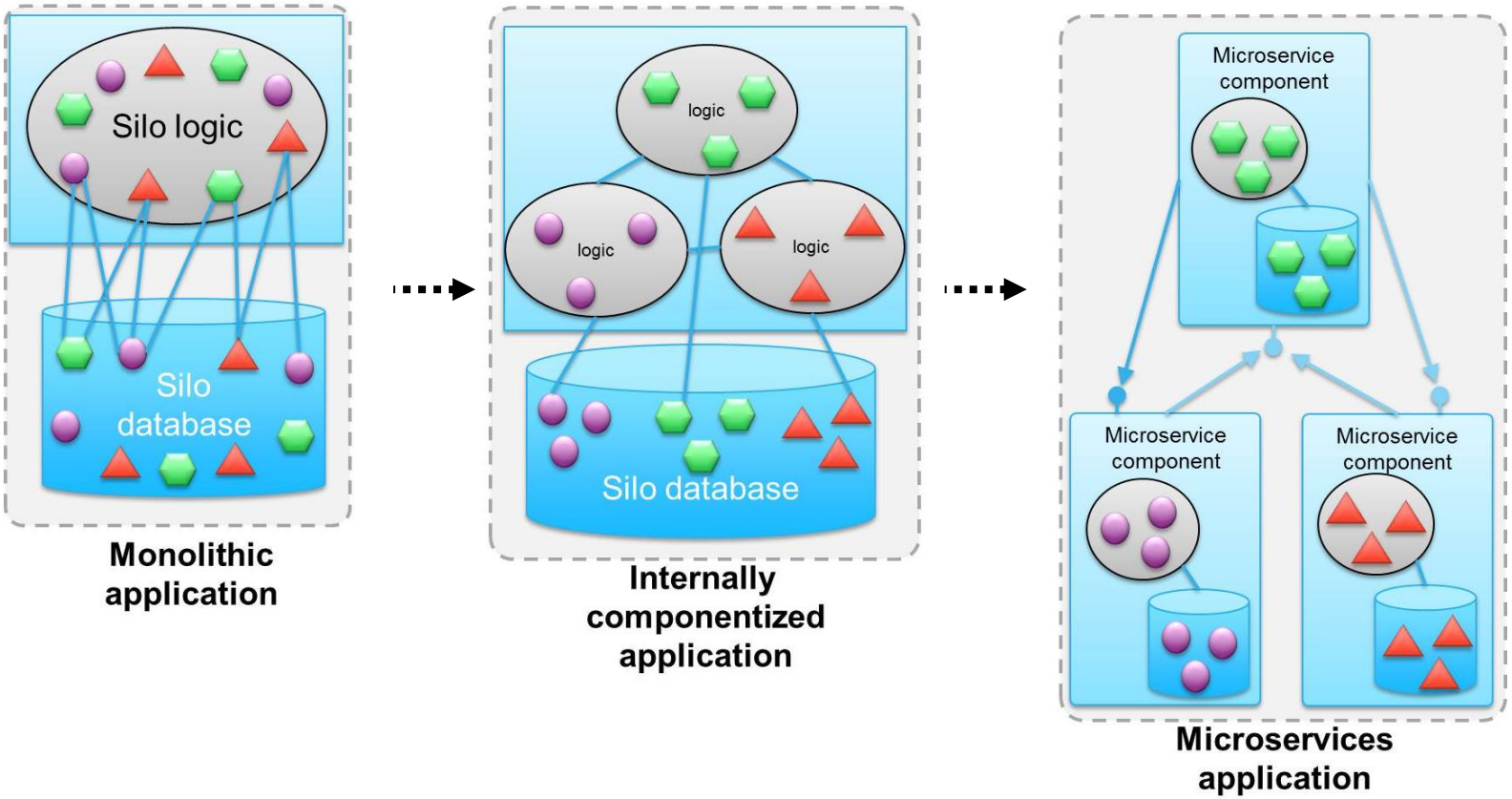
Business Domain Analyst

IT Architect

Developer, Administrator

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

From Monolith and Components to SOA and (Micro-)Services



Reference: IBM developerWorks – Microservices, SOA, and APIs: Friends or Enemies?
http://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

Microservices – An Early and Popular Definition (2014)

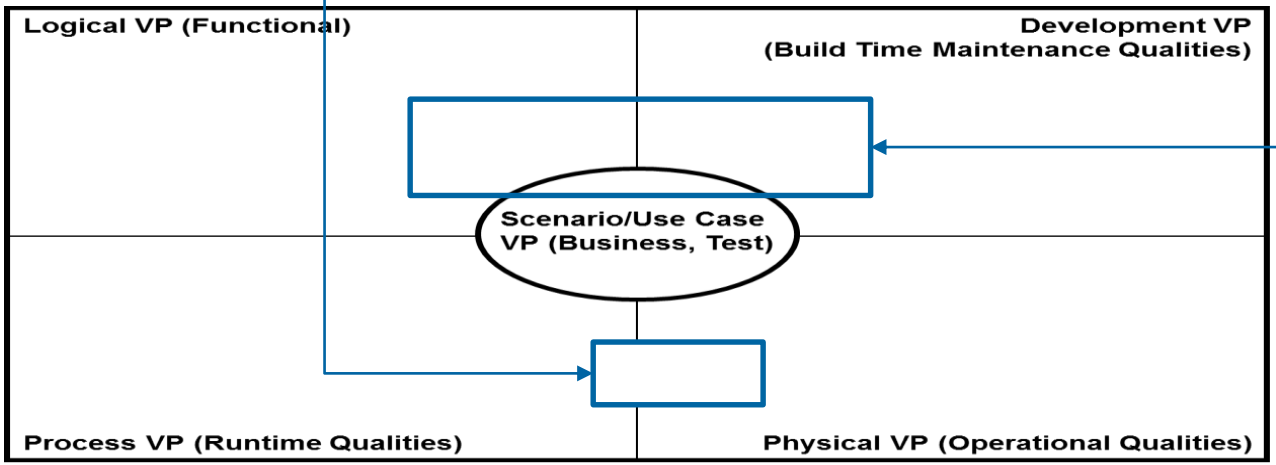
Reference: http://martinfowler.com/articles/microservices.html

- **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)

Microservices Definition: 4+1 Viewpoint Mapping (More: CSR&D Paper)

Application Component Property (Gartner/TMF)	Mapping to 4+1 Viewpoint Model (Kruchten 1995)	Mapping to ZIO Tenet	Novel or "Same Old Architecture"?
tightly scoped	Scenario/Use Case, Logical	1, 2	SOA
strongly encapsulated	Logical, Development	1	SOA
loosely coupled	Development, Process (Integr.)	1, 3	SOA
independently deployable	Process, Physical	1	novel
independently scalable	Process, Physical	1	novel



View model adapted from: P. Kruchten, 4+1 views on SWA, IEEE Software.

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.**

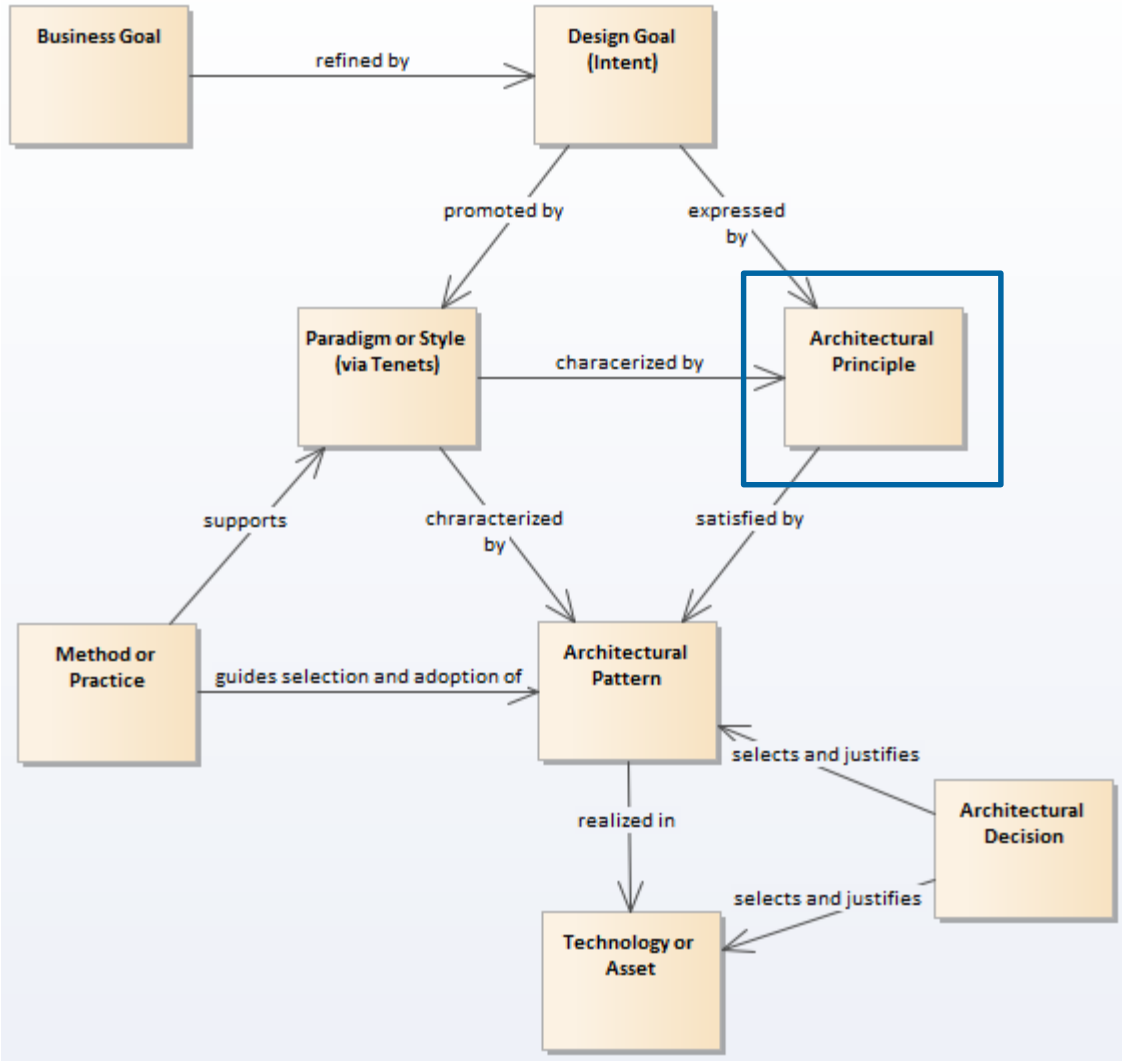
Reference: O. Zimmermann, Microservices Tenets – Agile Approach to Service Development and Deployment, Proc. Of SummerSoC 2016, Springer Computer Science – Research and Development, 2016 (CSR&D Paper).

Why SOA and Microservices?

- **Microservices are distributed application components and therefore IDEALly suited for a cloud deployment**
  - Isolated State and other IDEAL cloud application properties introduced next
- **Microservices work well with agile, self-organized teams that develop and operate their service(s)**
  - High velocity due to reduced communication with other teams
  - Some technological independence w.r.t. frameworks and programming Languages
  - Improved maintainability, at least in theory:
    - Microservices can easily be replaced
    - Architecture might be less prone to erosion over time because microservice boundaries are harder to overcome than in a single codebase.
    - But increases runtime complexity (when to decommission a service? versioning?).
- **A highly distributed and decentralized deployment and management approach has potential to increase robustness and resiliency**

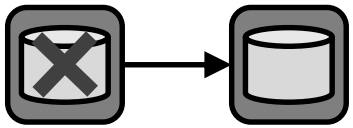
Architectural Principles define Architectural Styles and Paradigms

- **Business goals and design goals**
- **Paradigms (defined by tenets)**
- ***Principles***
- **Patterns**
- **Decisions**
- **Methods, practices, tools**

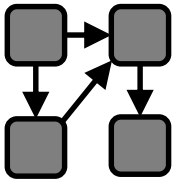


IDEAL Cloud Application Properties (Fehling, Leymann et al.)

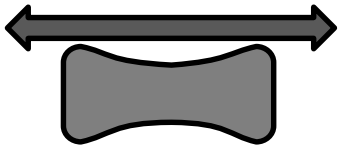
Reference: Cloud Computing Patterns, Springer 2014, http://cloudcomputingpatterns.org/



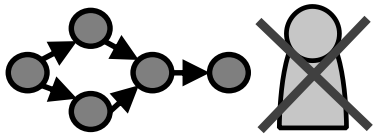
Isolated State: most of the application is *stateless* with respect to:
Session State: state of the communication with the application
Application State: data handled by the application



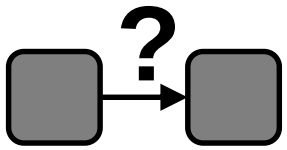
Distribution: applications are decomposed to…
… use multiple cloud resources
… support the fact that clouds are large globally distributed systems



Elasticity: applications can be scaled out dynamically
Scale out: performance increase through addition of resources
Scale up: performance increase by increasing resource capabilities



Automated Management: runtime tasks have to be handled quickly
Example: exploitation of pay-per-use by changing resource numbers
Example: resiliency by reacting to resource failures



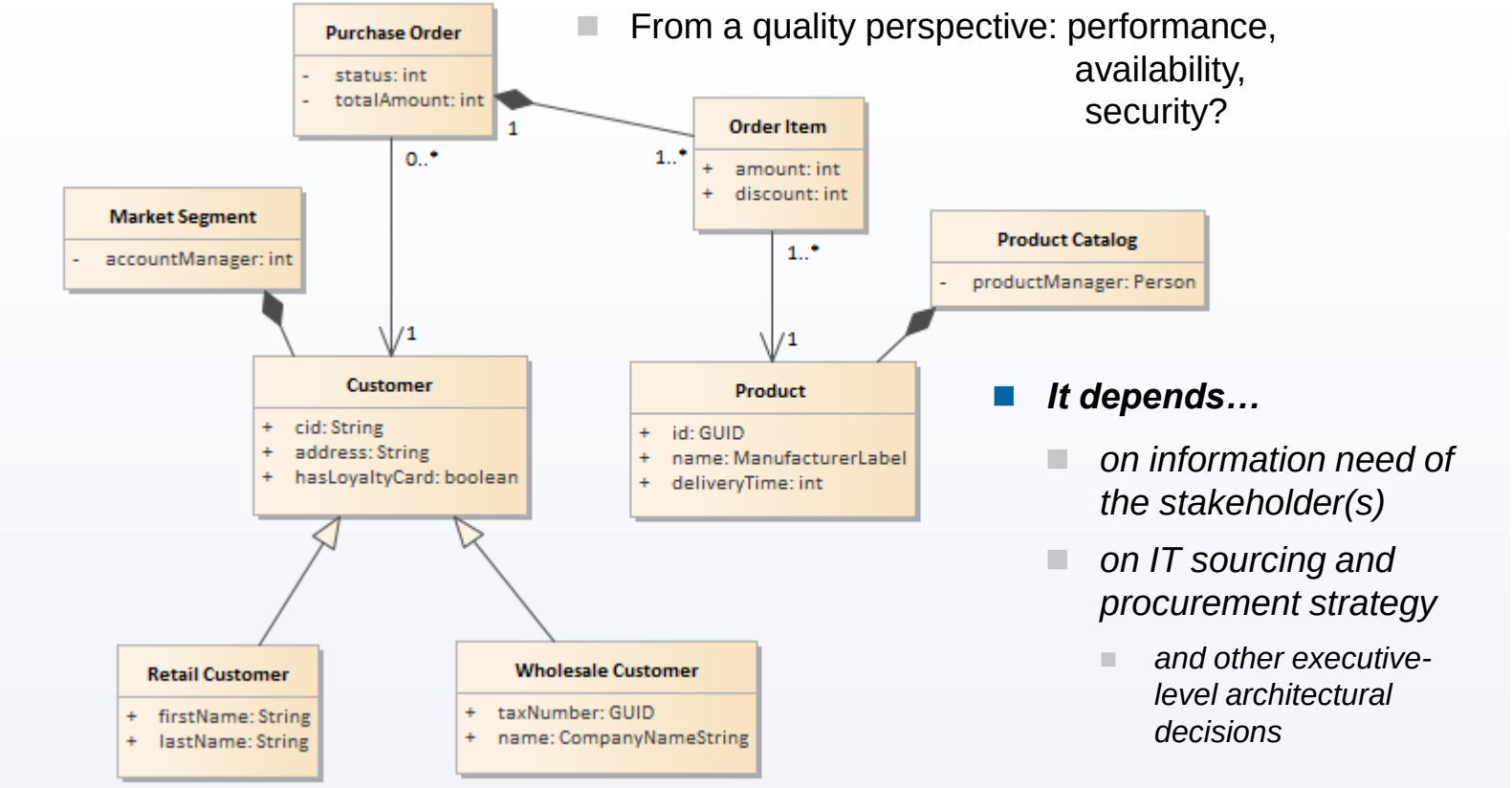
Loose Coupling: influence of application components is limited
Example: failures should not impact other components
Example: addition / removal of components is simplified

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*

Coupling Example in an Online Shop/e-Commerce (0/3)

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

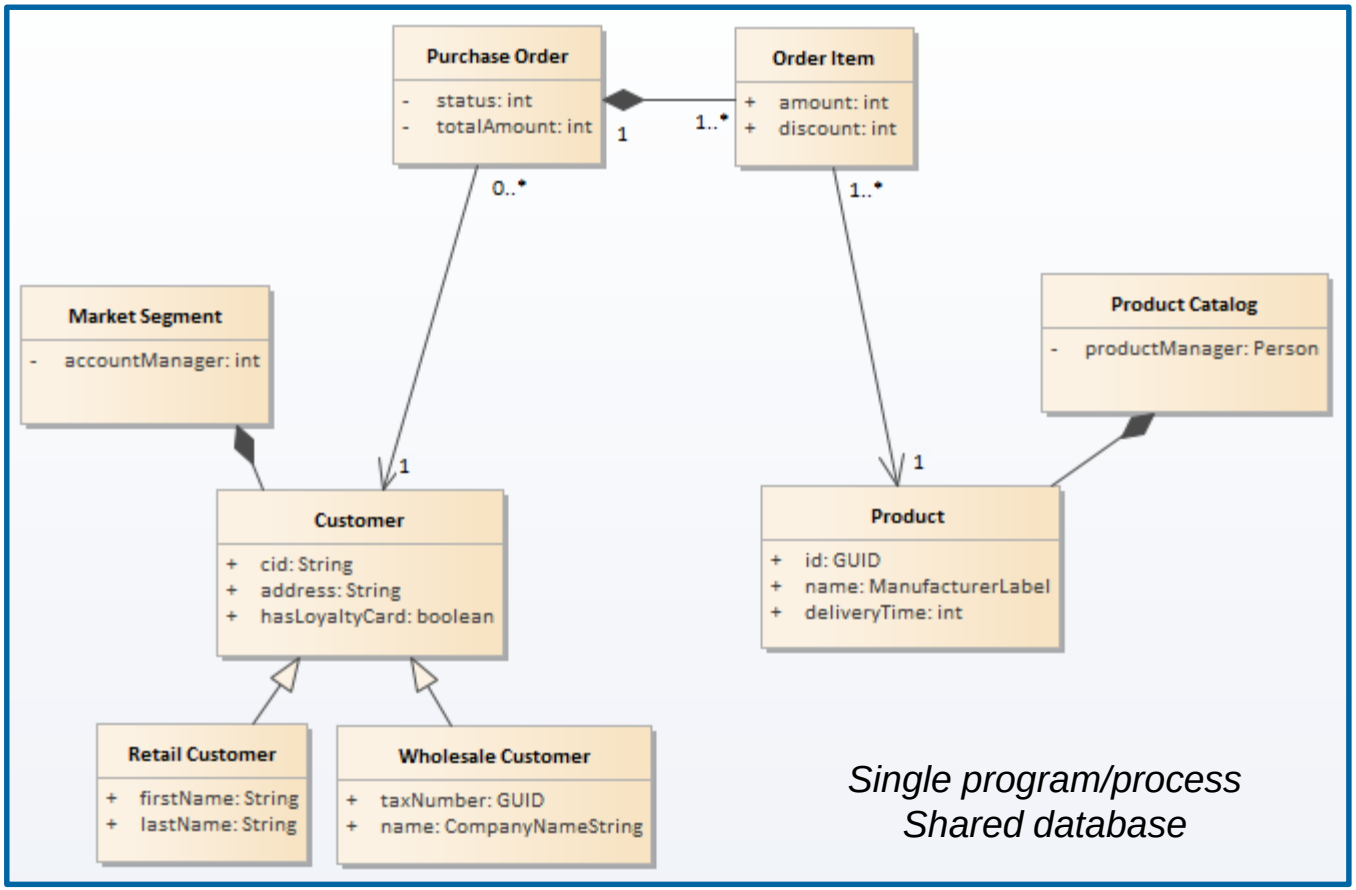


- ***It depends…***
  - *on information need of the stakeholder(s)*
  - *on IT sourcing and procurement strategy*
    - *and other executive-level architectural decisions*

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

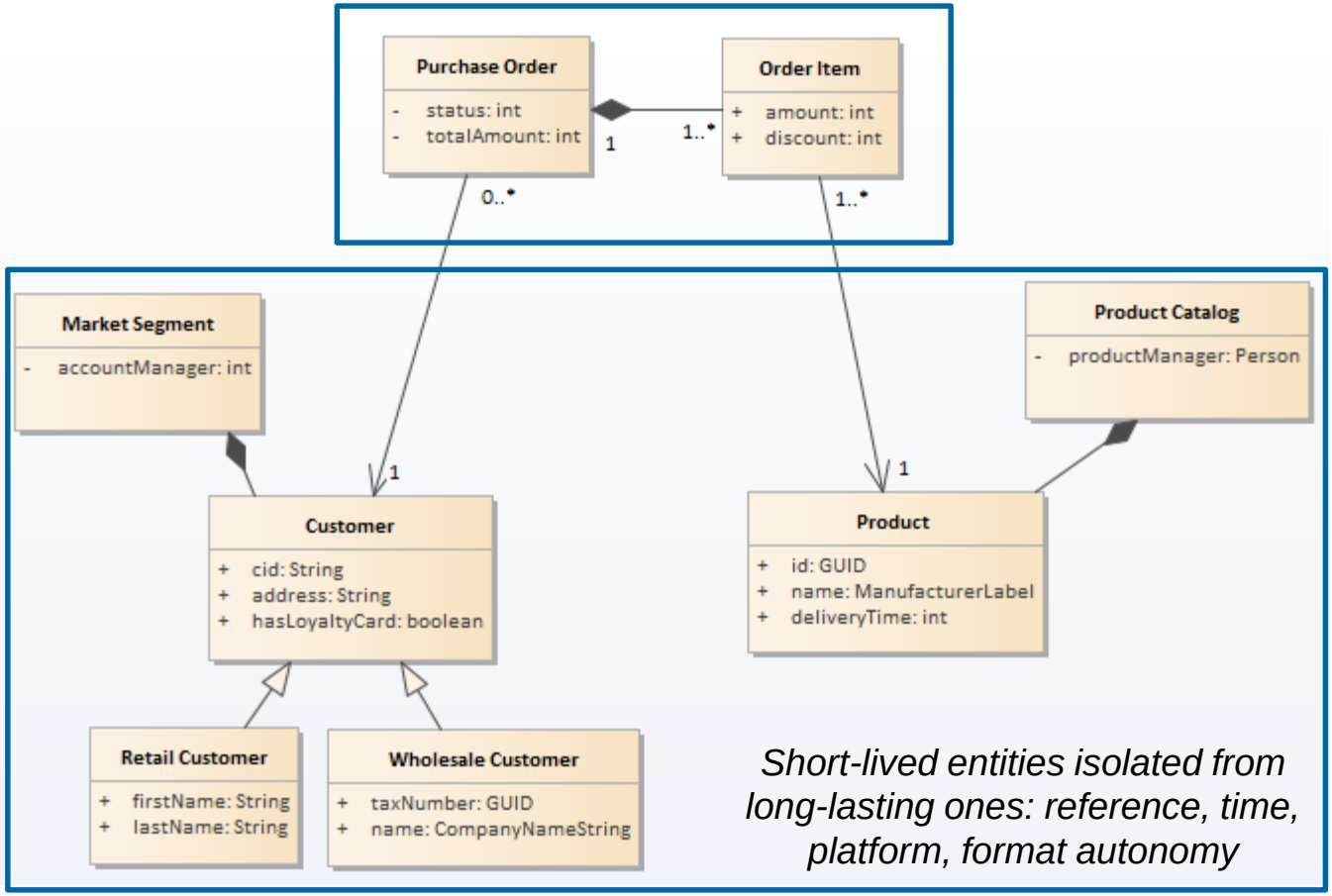
Service Boundary (Remote Interface)

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



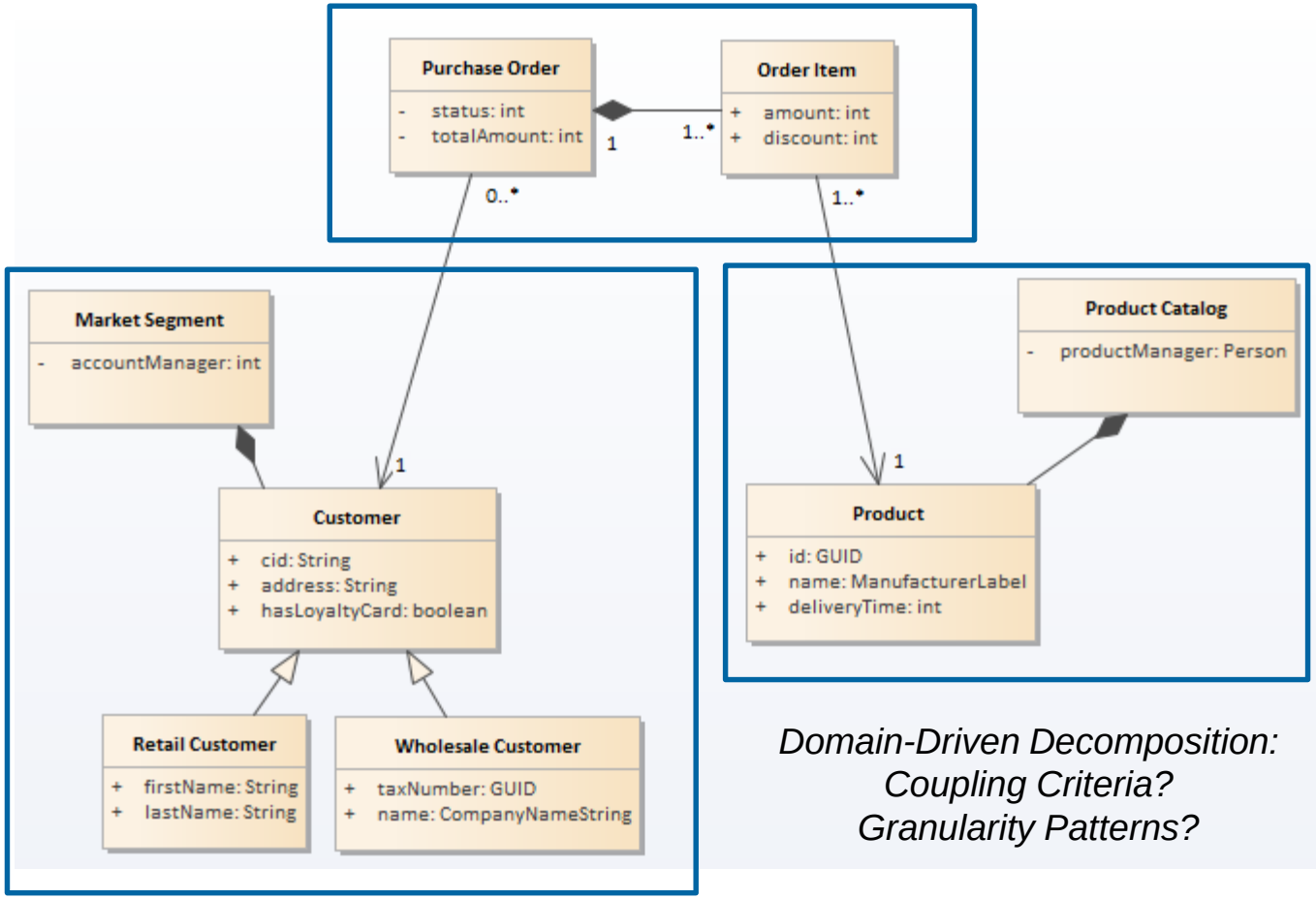
Coupling Example in an Online Shop/e-Commerce (2/3)

- **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

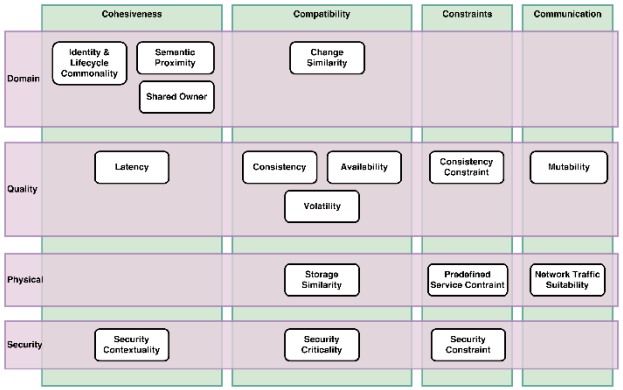
A Software Architect's Dilemma….

How do I split my system into services?

Step 1: Analyze System

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

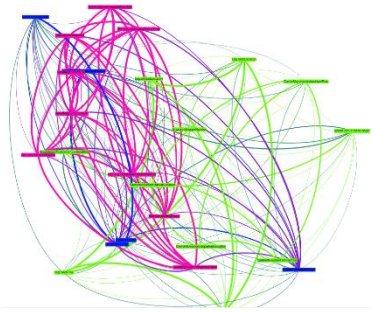
Coupling information is extracted from these artifacts.



The catalog of 16 coupling criteria

Step 2: Calculate Coupling

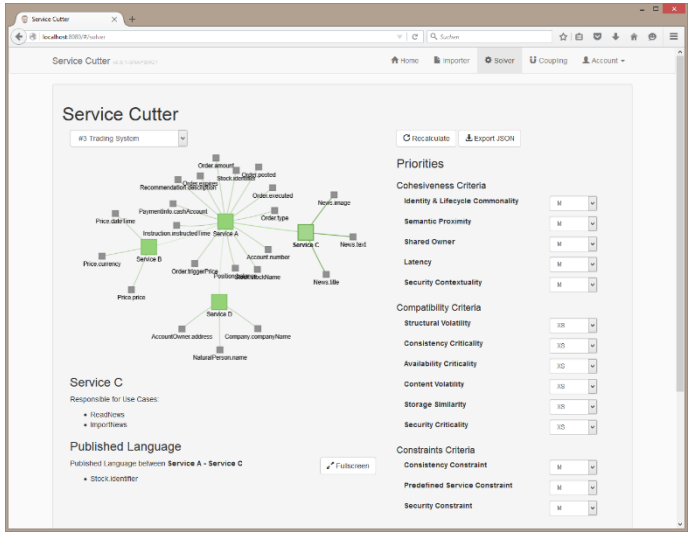
- Data fields, operations and artifacts are nodes.
- Edges are coupled data fields.
- Scoring system calculates edge weights.
- Two different graph clustering algorithms calculate candidate service cuts (=clusters).



A clustered (colors) graph.

Step 3: Visualize Service Cuts

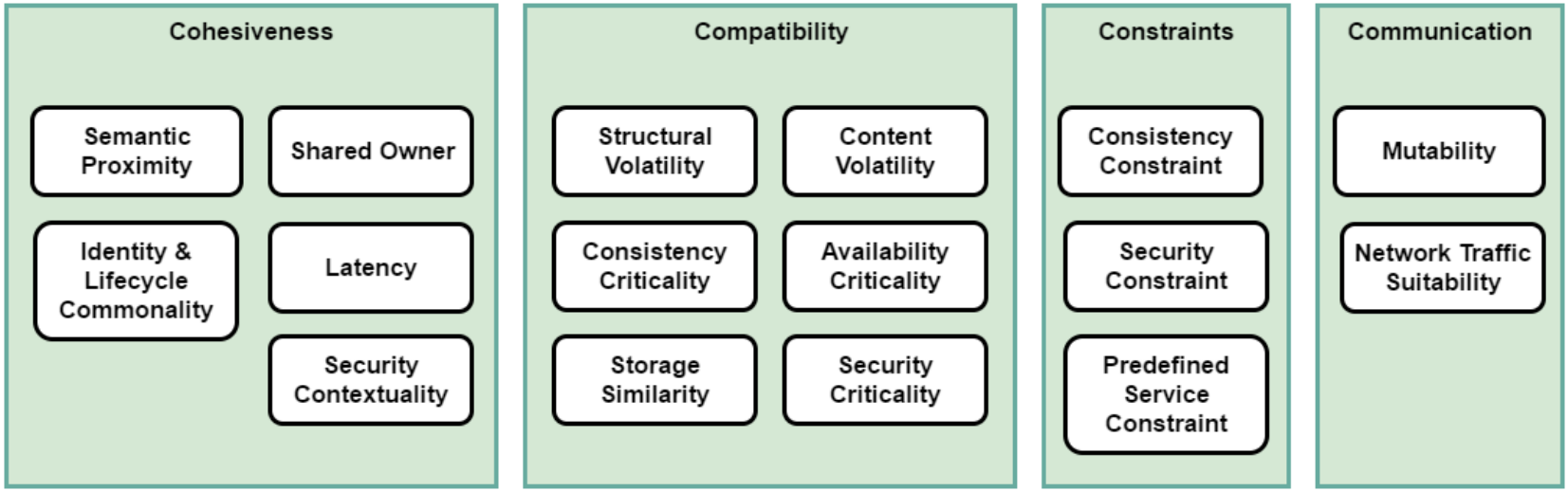
- Priorities are used to reflect the context.
- Published Language (DDD) and use case responsiblities are shown.



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)

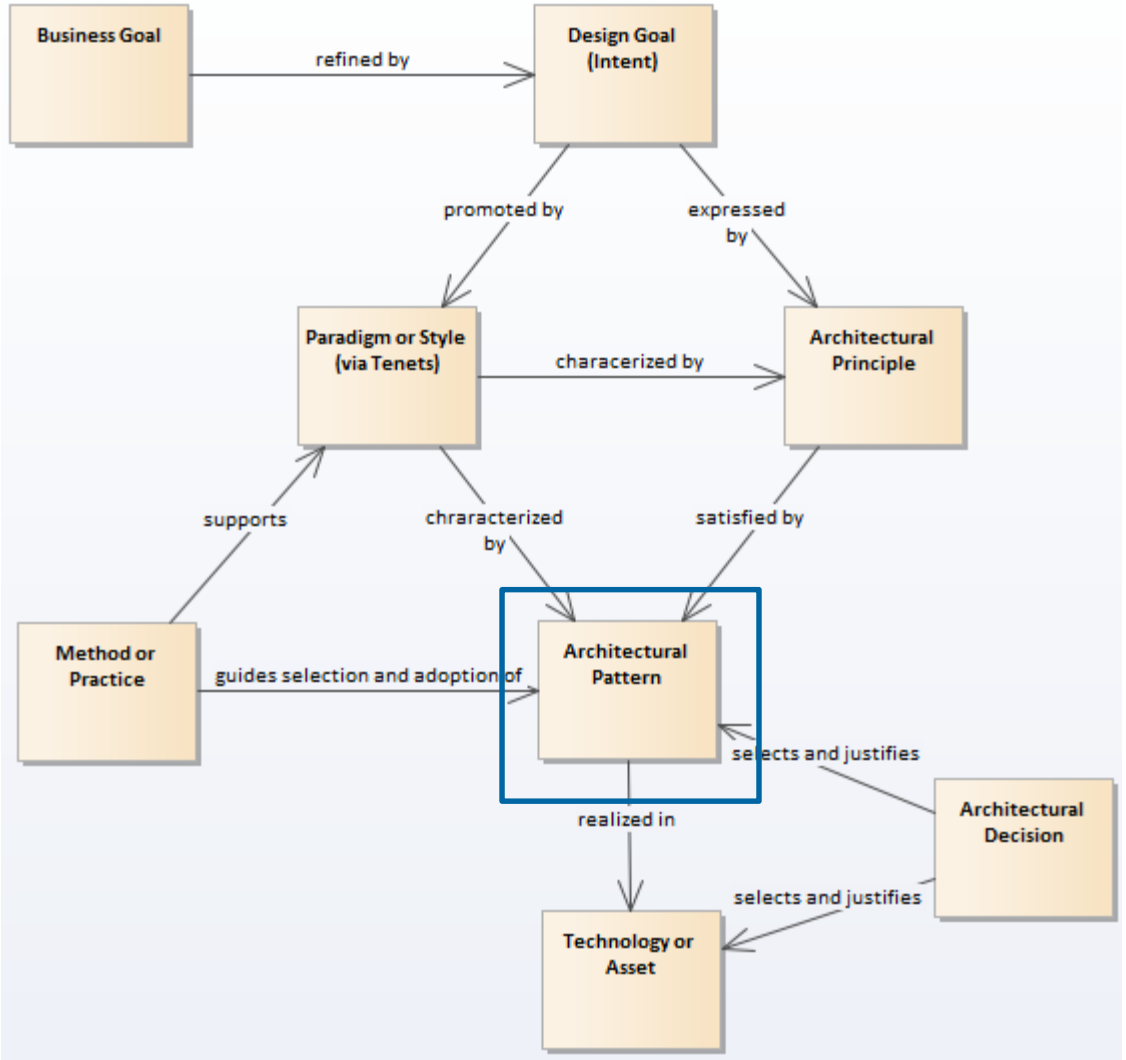


Full descriptions in CC card format: https://github.com/ServiceCutter/ServiceCutter/wiki/Coupling-Criteria

- **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)

From Style Tenets and Principles to (Architectural) Patterns

- **Business goals and design goals**
- **Paradigms (defined by tenets)**
- **Principles**
- ***Patterns***
- **Decisions**
- **Methods, practices, tools**



What is “Micro” a.k.a. How Small is (too) Small?

- **Judging from the name, the size of a microservice seems to be an important criterion – but how to define/measure it?**
  - Optimal size of a microservice is not measured in Lines of Code (LoC)
- **The size of a microservice should be chosen such that it can be**
  - Developed (and operated => DevOps?) by a single team
  - Fully understood by each developer on the team
  - Replaced by a new implementation if necessary
- **On the other hand, it should not be too small**
  - Communication and deployment overhead
  - Transactions spanning multiple microservices are hard to manage
  - The same is true for data consistency (consistency boundaries)

Jeff Bezos’s Two-Pizza Rule for optimal team size

What belongs in a Microservice?

- **A microservice should be large enough to contain the data it needs to operate – and loosely coupled with others**
- **New or changed business requirements should ideally lead to changes in just a single microservice (including the user interface)**
- **Example:**
  - An e-commerce order management service should also handle the order data. In addition, it will also need access to customer data and product information to fulfill its responsibilities.
- **Which data should the order management service own and control?**
  - Only *transactional data* such as order items, bill, delivery?
  - Or *master data* as well (customer, products)?

Be careful not to end up with a (distributed) monolith again!

Service Granularity Test (by Example)

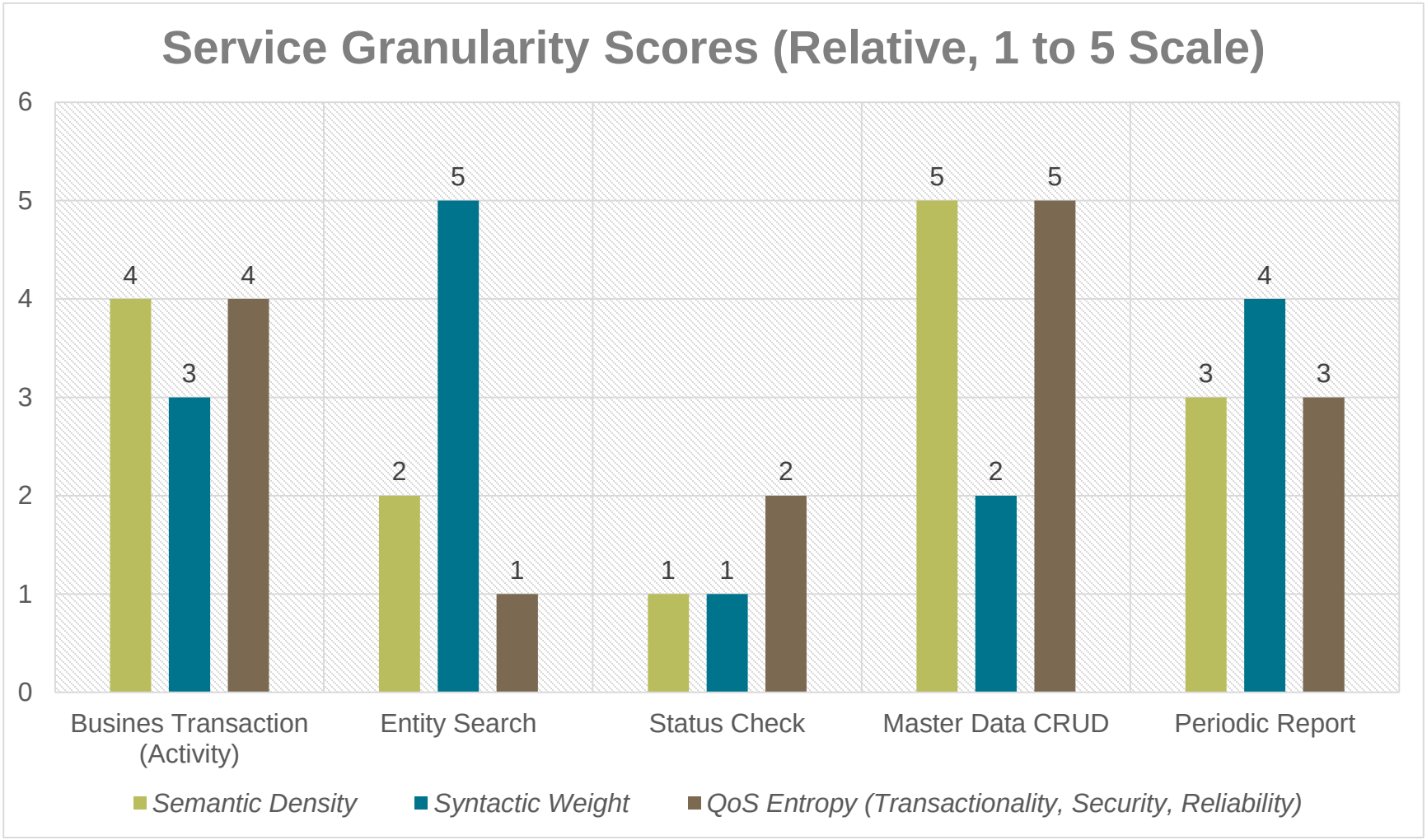
- **Test: Do the exemplary services qualify as microservices?**
  - “small” (Lewis/Fowler) and “fine grained” (Netflix, ZIO)?
  - “having a single responsibility” (R. Martin)?
  - “being maintainable by a 2-pizza team” (J. Bezos)?
  - supporting IDEAL principles such as loose coupling (Fehling et al, ZIO)?
- **Example A: *Exchange Rates* in YaaS/Hybris (SAP):**
  - https://devportal.yaas.io/services/exchangerates/latest/
- **Example B: *Create Goods and Activity Confirmations* (SAP B. by Design)**
  - https://help.sap.com/doc/saphelp_byd1702_en/2017.02/en-US/PUBLISHING/PSM_ISI_R_II_APGACFM_GOODS_CONF_IN.html
- **Example C: *Create an Outbound Delivery with a Reference to a Sales Order* (in ESA/Hana via SAP Business Hub)**
  - https://api.sap.com/#/catalog/a7a325f837df42f8a5c1083890e28801/II_SHP_OUTBOUNDDELIVERYCWRRC/SOAP

Service Granularity in Scientific Literature and Practice Reports

- **Business granularity (a.k.a. *semantic density*) has a major impact on agility and flexibility, as well as maintainability**
  - Position of service operation in business architecture, e.g., expressed in a Component Business Model (CBM) or enterprise architecture model
  - Amount of business process functionality covered
    - Entire process? Subprocess? Activity?
  - Number and type of analysis-level domain model entities touched
- **Technical granularity (a.k.a. *syntactic weight*) determines runtime characteristics such as performance and scalability, interoperability – but also maintainability and flexibility**
  - Number of operations in WSDL contract, number of REST resources
  - Structure of payload data in request and response messages
- ***QoS entropy* adds to the maintenance effort of the service component**
  - Backend system dependencies and their properties (e.g. consistency)
  - Security, reliability, consistency requirements (coupling criteria)

Granularity Scores by Service Pattern and Granularity Type

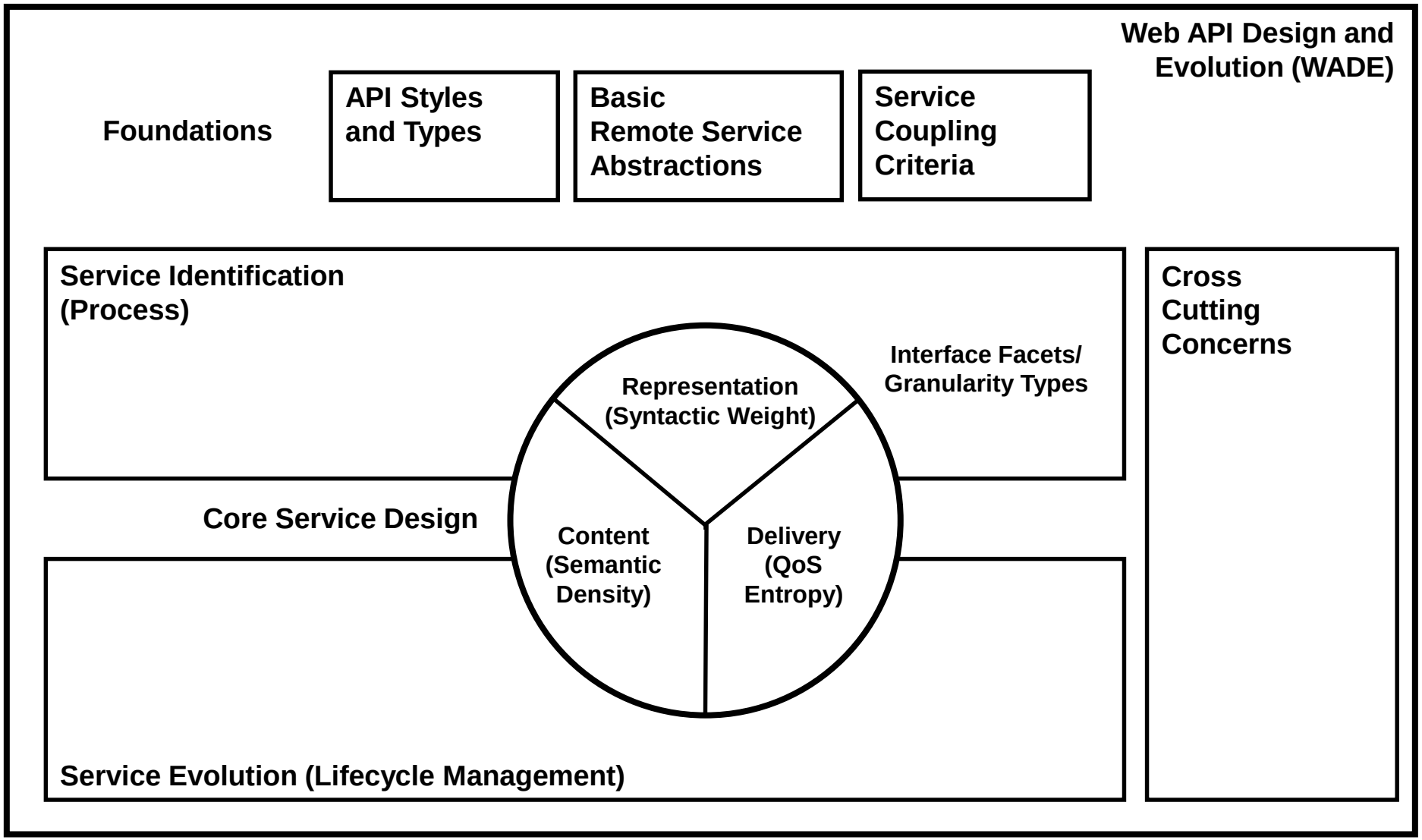
CRUD – Create, Read, Update, Delete; QoS – Quality of Service



Granularity Types and Criteria – Observations and Findings

- **Sometimes granularity is also seen as an architectural principle:**
  - https://en.wikipedia.org/wiki/Service_granularity_principle
- **Granularity is property of service contract exposed by a service provider**
  - Not an exact measure/metric, but a heuristic/an indicator of modularity and cohesion (on different levels of abstraction)
  - Business granularity vs. technical granularity (syntax, QoS)
- **Can't really tell the "right" size w/o use cases and (de)coupling criteria – "it depends" (again):**
  - Clients, contexts, concerns differ – for good reasons!
    - Service semantics, information need of consumer
    - Hidden complexity (backend, relations)
- **Conclusion: *A continuum of service granularity patterns exists***
  - There is no such thing as a "right" service size for all systems and service ecosystems – but the candidate *service cuts* can be captured as patterns

Towards an Interface Representation Pattern Language (IRP)



Candidate Patterns in IRP (Work in Progress)

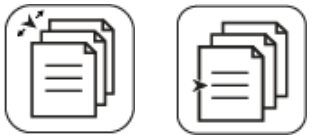
Category				
Foundations	Vertical Integration, Horizontal Integration	Public API	Community API	Solution-Internal API
Process	Contract First	Static Discovery	Dynamic Discovery	Service Model
Representation	AtomicParameter (Single Scalar, Dot)	Parameter Tree (Single Complex)	Atomic Parameter List (Multip. Scalars, Dotted Line)	ParameterComb (Multiple Complex)
	Pagination, Page	Query Parameter	Cursor	Offset
	Wish List	Request Deck	Metadata Parameter	Annotated Parameter List
Content Semantics	Command Service	Reporting Service	Status Check	Master Data Update
QoS	Service Contract, Context Object	SLA-SLO	API Key/Access Token	Rate Limit
Evolution	Semantic Versioning, Version Identifier	Two (Versions) in Production	Aggressive Deprecation	Liberal Receiver/ Conservative Sender

Reference: O. Zimmermann et al., Interface Representation Patterns, accepted for EuroPLOP 2017 (under shepherding)

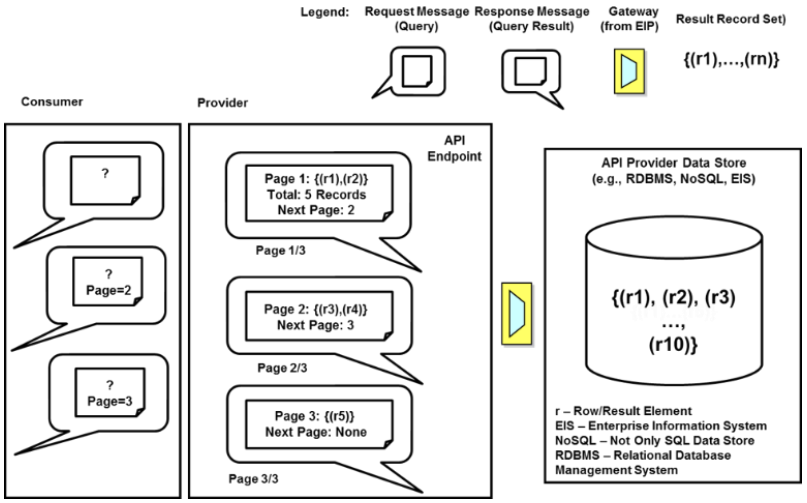
Example IRP: Pagination (1/2)

- **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

Example IRP: Pagination (2/2)

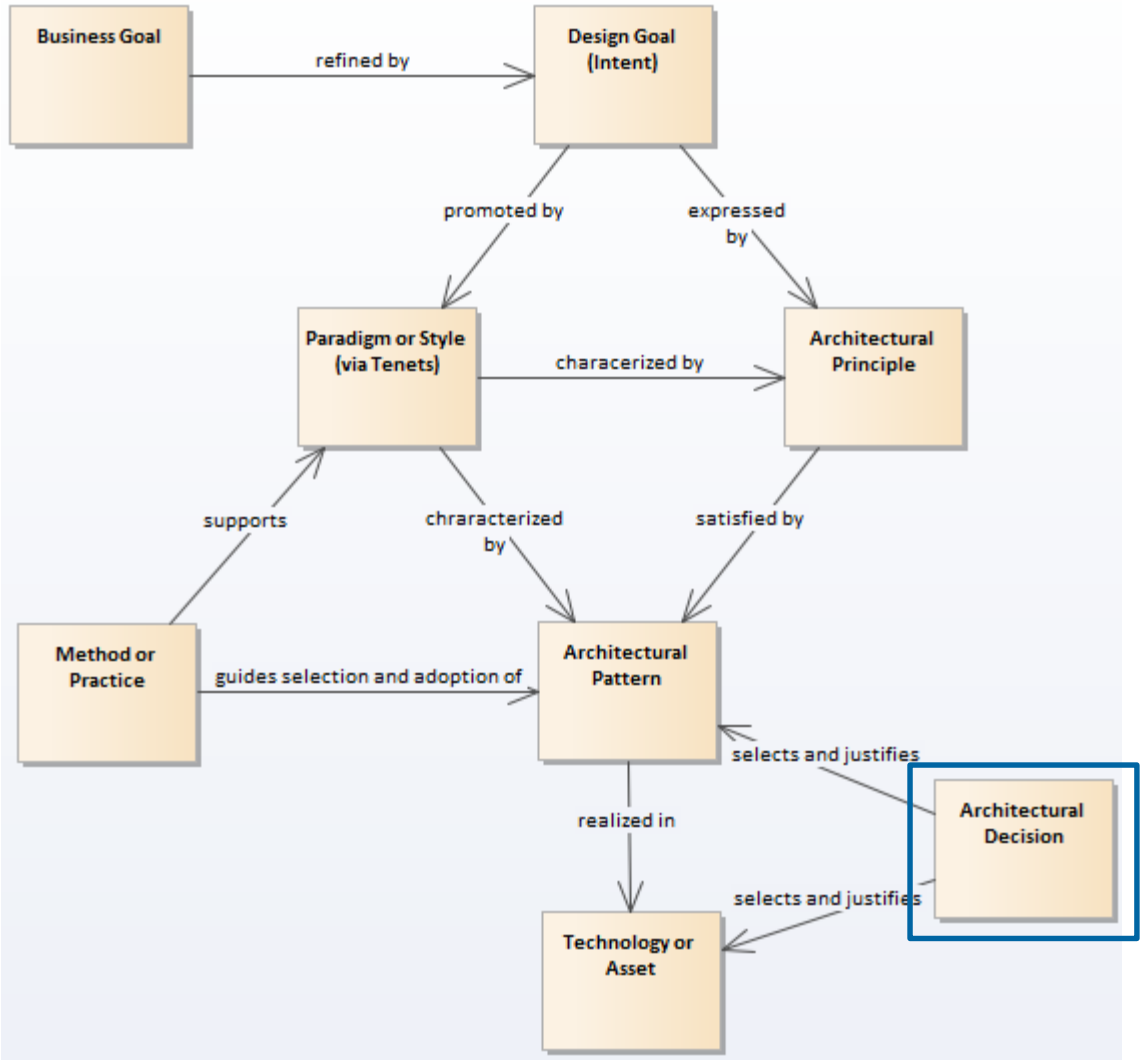


- **Solution**
  - Divide large response data sets into manageable and easy-to-transmit chunks.
  - *Send only partial results in the first response message and inform the consumer how additional results can be obtained/retrieved incrementally.*
  - Process some or all partial responses on the consumer 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



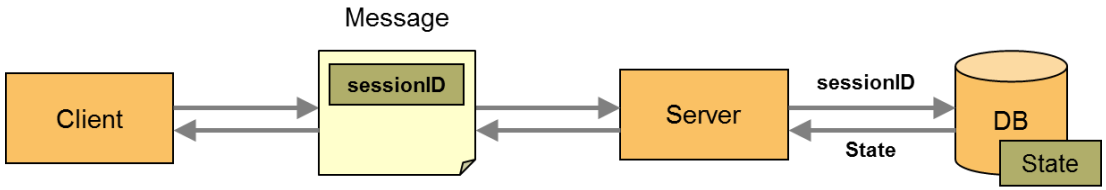
Selecting and Adopting Patterns Requires Decision Making

- **Business goals and design goals**
- **Paradigms (defined by tenets)**
- **Principles**
- **Patterns**
- ***Decisions***
- **Methods, practices, tools**



AD Modeling with Reuse – Context and Motivation (by Example)

- **AD capturing matters, e.g. ISO/IEC/IEEE 42010 has a rationale element**
  - But it remains an unpopular documentation task – particularly, but not only in agile communities
  - Effort vs. gain ("feeding the beast")?
- **Example (from cloud application design): Session State Management**
  - Shopping cart in online commerce SaaS (e.g., Amazon) has to be stored while user is logged in; three design options described in literature



"In the context of the Web shop service, *facing the need to* keep user session data consistent and current across shop instances, *we decided for* the Database Session State Pattern from the PoEAA book (and *against* Client Session State or Server Session State) to achieve ideal cloud properties such as elasticity, *accepting that* a session database needs to be designed, implemented, and replicated."

Reference: (WH)Y-template first presented at SEI SATURN 2012 and later published in IEEE Software and InfoQ, http://www.infoq.com/articles/sustainable-architectural-design-decisions
(inspired by decision part in George Fairbanks' Architecture Haiku, WICSA 2011 tutorial)

From Decisions Made to Decisions Required (Guidance)

- **Approach: Refactor decision capturing templates into problem-option-driver fragments and change tone, to separate concerns and to ease reuse**

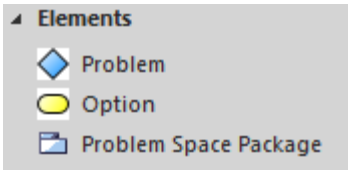
"In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State Pattern from the PoEAA book (and against Client Session State or Server Session State) to achieve cloud elasticity, accepting that a session database needs to be designed, implemented, and replicated."

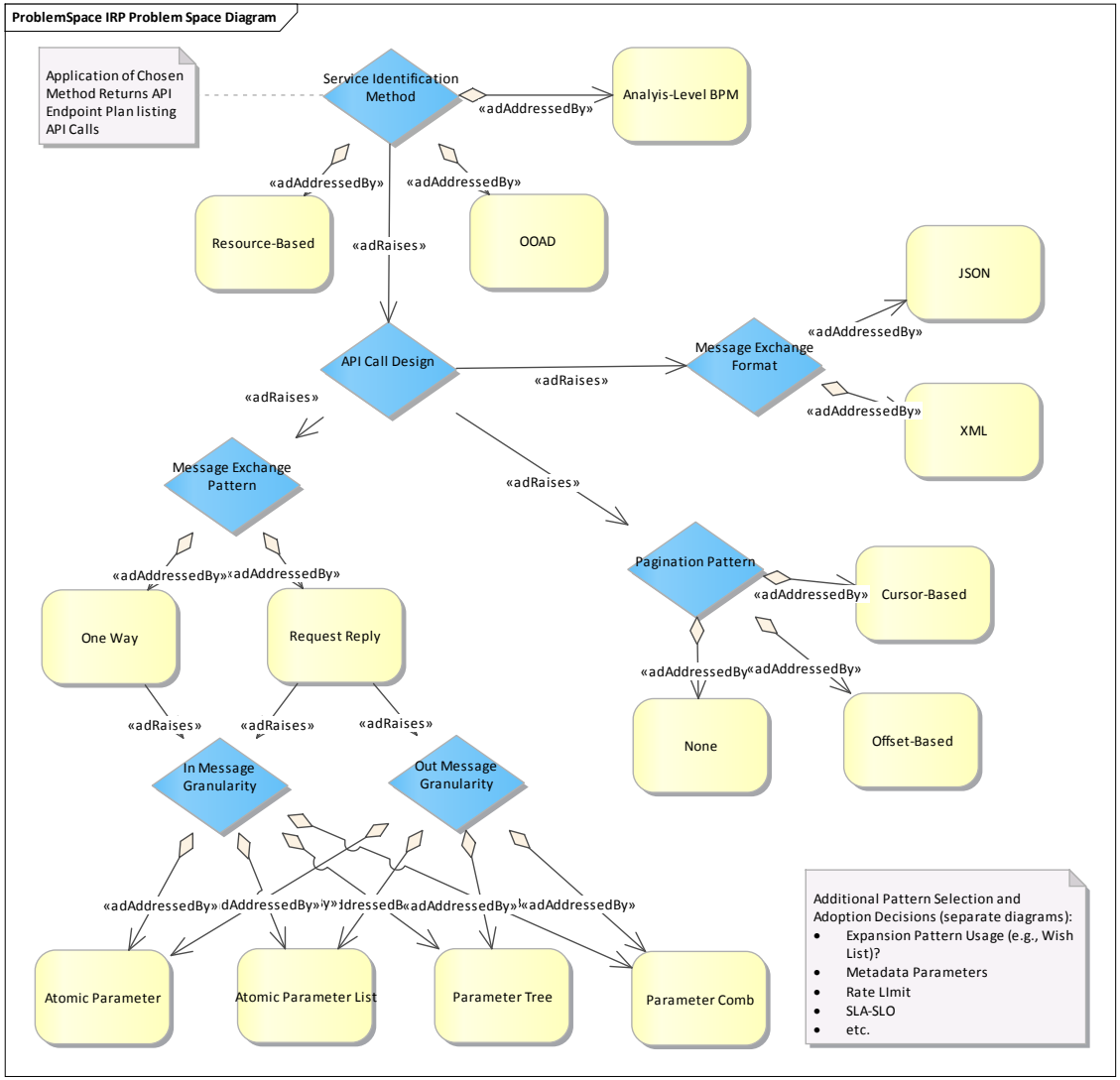
Curate {decision need, solutions, qualities} for reuse – but *not* the actual decision outcomes

- "When designing a stateful user conversation (for instance, a shopping basket in a Web shop), *you will have to* decide whether and how session state is persisted and managed." (question: is this a requirement or stakeholder concern?)
- "Your conceptual design options *will be* these patterns: Client Session State, Server Session State, and Database Session State."
  (question: are patterns the only types of options in AD making?)
- "The decision criteria *will include* development effort and cloud affinity."
  (question: what else influences the decision making?)

IRP Selections (a.k.a. Service Design Space) in ADMentor

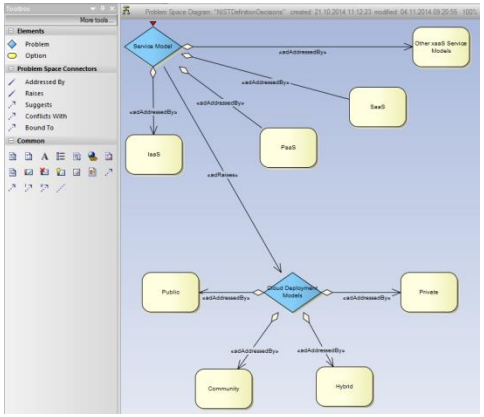
- **Patten selection and adoption qualifies as AD making**
  - Rationale to be captured: qualities, conformance with principles, etc.
- **Guidance through service design space via problem-option pair modeling**
  - In ADMentor





ADMentor Tool (AddIn to Sparx Enterprise Architect, “UML++”)

- **ADMentor is openly available at https://github.com/IFS-HSR/ADMentor**



Architectural Decision Guidance across Projects

Problem Space Modeling, Decision Backlog Management and Cloud Computing Knowledge

Olaf Zimmermann, Lukas Wegmann
Institute for Software
Hochschule für Technik (HSR FHO)
Rapperswil, Switzerland
{firstname.lastname}@hsr.ch

Heiko Koziolek, Thomas Goldschmidt
Research Area Software
ABB Corporate Research
Ladenburg, Germany
{firstname.lastname}@de.abb.com

(WH)Y?

- My version (the Y-approach):
  - *In the context of <use case/user story u>, facing <concern c>, we decided for <option o> to achieve <quality q>*
  - These Y-statements yield a bullet list of open/closed (design) issues (link to project management!)
  - Can go to appendix of software architecture document, notes attached to UML model elements, spreadsheet, team space, or wiki

ABB

- **Project website http://www.ifs.hsr.ch/index.php?id=13201&L=4**

Key Take Away Messages (Position Summary)

- **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)
- **Architectural principles and patterns characterize architectural styles**
  - e.g. loose coupling is a key SOA principle (multiple dimensions)
- **There is no single definite answer to the "what is the right granularity?" question, which has several context-specific dimensions and criteria**
  - Business granularity: semantic density (role in domain model and BPM)
  - Technical granularity: syntactic weight and QoS entropy
- **Platform-independent service design can benefit from Interface Representation Patterns such as Pagination, Wish List, Master Data CRUD**
- **Pattern-centric service design involves architectural decisions that recur**

Service Design Science – Towards a Research Roadmap

CS Field	Contribution Type(s)
Software engineering, SoC	Design by contract, MDSE, value networks
Databases, Information Systems	Representation modeling, query languages
Networking	Protocol design (conversations), contract verification (Interoperability. conformance testing)
Business Process Management and Modeling (BPM)	Service identification in static and dynamic business models, composition middleware
Distributed Systems, Telecommunication Networks	Event-driven, reactive, adaptive architectures, service discovery, metering and billing
Internet Technologies, Web Engineering	Semantic (micro-)service linking (not matchmaking)
Theoretical Computer Science	Formal definitions: SOA/MSA, service, MEP, etc.

- **My take on future trend in SoC/service design:**
  - Overarching knowledge question: *How to adopt existing and new computer science research results for the context of agile Web/service engineering?*
  - "Long live services – of various kinds and granularities" (ZIO, 2016)

SOA/Microservices and Semantic Big Data Management

- **REST maturity level 3 makes HATEOAS mandatory for any Web API that claims to be RESTful, which requires typed links**
  - Original vision of the Semantic Web by Tim Berners-Lee
  - HTTP API or Web API vs. RESTful HTTP API or Hypermedia API
- **Domain-Driven Design is about modeling the business domain the microservices and end user applications target**
  - Can be seen as a “poor man’s ontology”
- **Automation of provisioning etc. requires an understanding of the configuration scripts etc.**
  - Which is understandable for humans and machines
- **DevOps produces large amounts of distributed monitoring data**
  - Containers, network, integration middleware, databases, etc.
- **Complex event processing and adaptive systems as advanced usage scenarios with built in dynamism (“on demand”, runtime decisions)**
  - Auto scaling in the cloud; ad hoc service discovery and matchmaking (?)

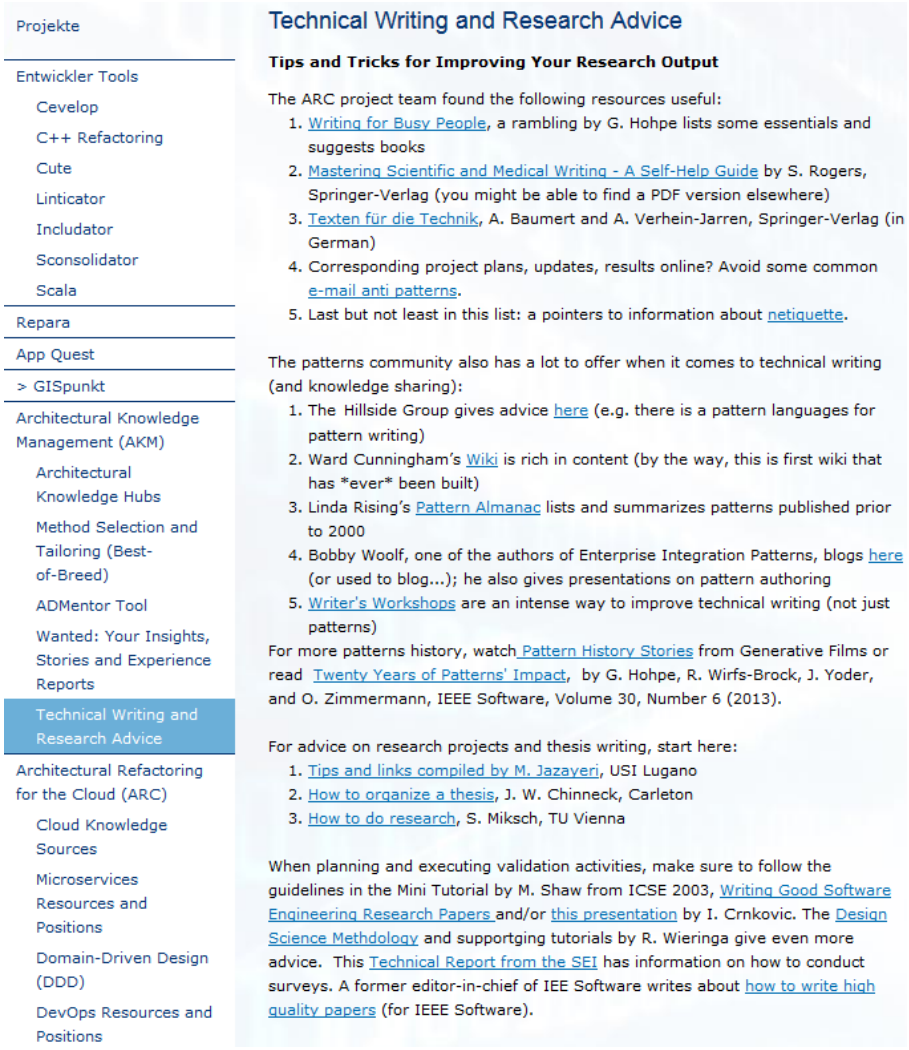
Agenda (“3P++”)

1. **Introduction to Service-Oriented Computing Paradigms**
   - Service-Oriented Architecture (SOA) style
   - Microservices tenets: agile approach to service realization
2. **Architectural Principles**
   - IDEAL cloud application architectures
   - Loose coupling, coupling criteria
3. **Interface Representation Patterns (IRP)**
   - Service Granularity (Business/Technical), Quality of Service
   - Pagination
4. **Architectural Decision Making, Capturing, and Sharing**
   - Y-statements, ADMentor tool
5. ***Lessons learned from Projects in Industry and Academia***
   - *Research challenges and vision*

Lessons Learned: Academia (Paper and Thesis Writing)

- **Follow a recognized research method**
  - E.g. Empirical
  - E,g. Design Science Methodology (DSM)
  - Action research and other validation forms
- **Take a look at other papers/theses**
  - Same advisor
  - Same 2nd advisor
  - PC chairs/members in target community

(screen caption is hyperlink)

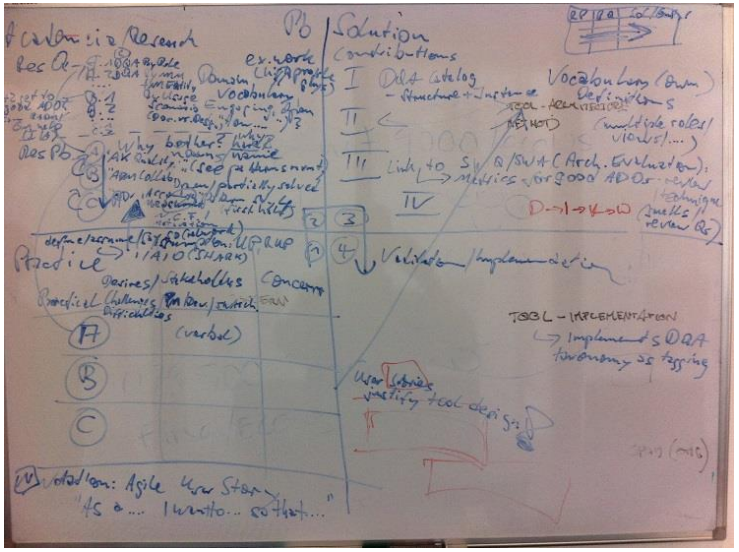


Generalization of Practical Problems into Research Problems

- **Abstract from practice, solve, instantiate**
  - Validation type to be picked wisely
    - Iterative and incremental approach ok
- **Finding good names matters… and is hard (iterate!)**
  - Research problem: noun (like pattern), research questions
  - Solution building block (contribution): noun (like a component in an architecture)
- **Research contribution spectrum:**
  - New problem and solution vs. new solution to existing problem (more efficient, more elegant, improvements in other quality attributes)

Problem vs. Solution

Practice vs. Research



Scoping Applied Research – Patterns and Anti Patterns

- **Use-case or user story driven vs. “solution seeking problem”**
- **Interdisciplinary work (“über den Tellerrand schauen”) vs. trend surfing**
- **Solving a conceptually hard problem vs. making problem look hard**
- **Dedication to quality vs. “just a prototype” excuse for bugs and lack of usability**
- **Apply your own research results during your research**

- **Recognized research methods (for design science):**
  - Design Science Methodology (DSM) by R. Wieringa (e.g. problem statement template, knowledge questions)
  - Writing good software engineering research papers by M. Shaw
  - Empirical approaches

Some Questions to Expected (from Advisor and Peer Reviewers)

- **During thesis projects, you will be asked a lot of questions like:**
  - "Why do we need X, and why do we need it here"?
  - "Why do you call it X and not Y (a little earlier you called it X')?
  - "How does X relate to X-1, X-2, …, to X+1, X+2, …, and to Y?"
  - "How do you know that X is correct, and where do you show that?"
  - "Where does X come from, your contribution or literature"?
  - "Is X complete or are there any X+1, X+2"?
  - "Is X on right level of abstraction or do you mix X, sub-X, super-X"?
  - …
- **So far, so good…**
  - … the problem is that X, X', Y is element of {{word}, {sentence}, {bullet list}, {figure}, {table}, {paragraph}, {section}, {chapter}} in papers and thesis ☹ ☺
    - So X can be text snippet – and concepts too

Some “Hot Buttons”…

- **Quality over quantity**
  - E.g. page quota: n pages or m words (opinions vary)
- **Don’t structure thesis too deeply – 3 to 4 levels of headings at most**
- **Everything that applies for papers is still valid**
  - Structure: Context/problem/solution/why a solution/why better than everybody else’s
  - Intellectual Property Rights (IPR)/copyright ownership, research ethics
- **Keep figures simple and consistent, and explain them in surrounding text**
  - Few colors/shadings, if any
  - Arrow semantics (solid line vs. dashed line)
  - Name the standard notation that you use, or provide a legend for IRPs
- **Colons and parenthesis are good to tell reader what is coming**
- **Avoid any editorial sloppiness – typos, inconsistencies, gaps**

… More “Hot Buttons”

- **Purity and clarity over verbosity (in language)**
  - No filler adjectives/adverbs (“works in principle”, “more or less”)
  - No exaggerations (“how high is highly positive”)?
  - But keep reader interested, indicate logical flow of text by keywords
  - Eloquence is appreciated (e.g. “application genre”)
- **One message/one thought at a time (high cohesion/low coupling like in software design)**
  - One message per sentence
  - One aspect/topic per paragraph
  - Order matters (there is no unordered list/no set in technical writing)
- **Avoid Wikipedia citations, or Web portals like IBM developerWorks**
  - Apart from that, quality matters more than source (which is still relevant)
    - Journal, conference, workshop hierarchy; known names, seminal works
    - Try to be broad in terms of communities, age, etc.
    - Cite what supervisors cite; respect current style at your university/institute/group

… Even More “Hot Buttons”

- **Provide rationale to demonstrate maturity (“Durchdringungsgrad”)**
  - Why this criterion and no other? Why this design?
  - If you claim something, does that mean everything else is wrong?
  - If you declare something to be out of scope, say why, and/or where done (you/others)
- **Show purpose and value of individual parts of your work**
  - What does the reader do with the information you just provided?
  - How is it used later in the thesis?
  - How does it change the world (value), see e.g. DSM template
  - Provide the “big picture” – how do thesis parts work together?
- **Pick your vocabulary consciously**
  - Shows that you are in command of the literature
  - As many terms as needed, but not more; simple, unambiguous names
  - Use consistently, avoid synonyms and homonyms
  - Avoid forward references if possible

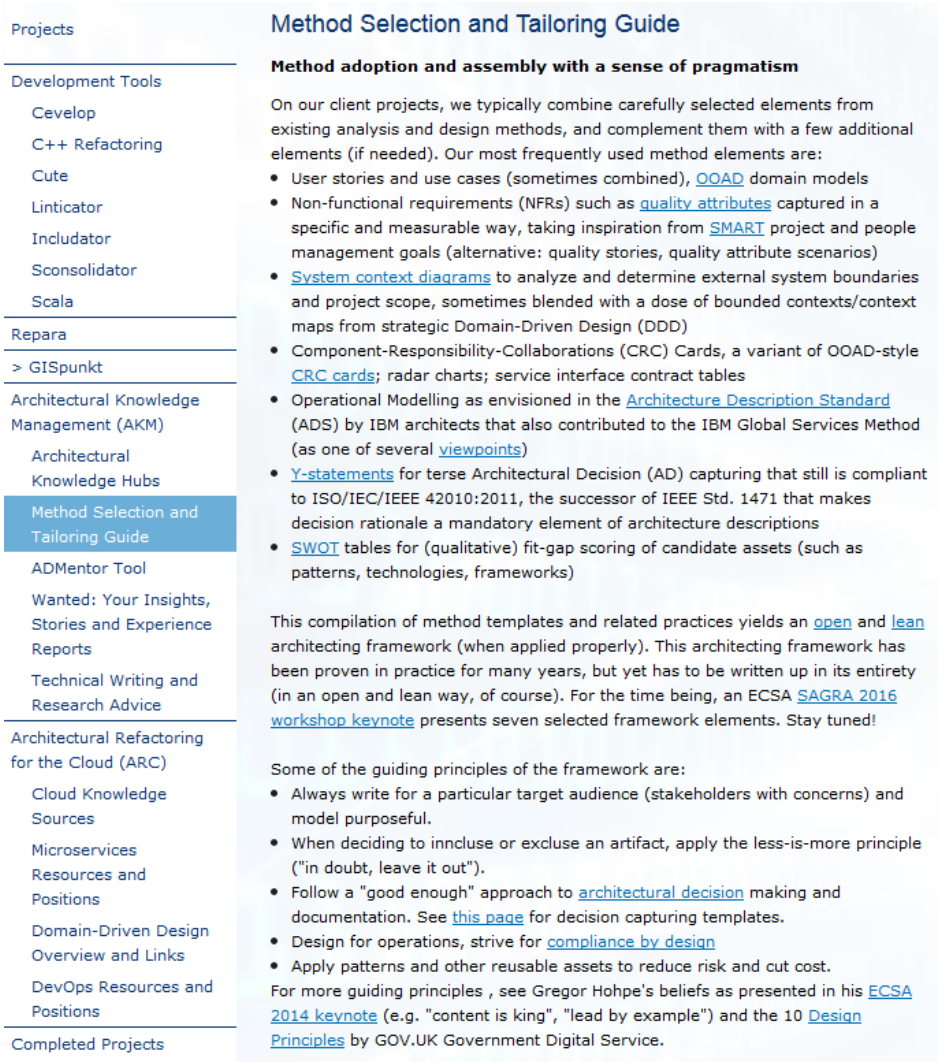
… Finally Some of ZIO's Ones (Valid for any Publication)

- **Tell your story five times:**
  - TOC/structure, text highlights (definitions, bullet lists), examples, figures/tables, full text – but in a consistent manner (semantic references)
- **Let intermediate drafts sit for a while to refresh your perspective**
  - Helps you to read from reader's perspective (e.g., abstract, intro and summary only)
  - Read the TOC only – it must tell the full story (exec. summary, speed readers)
- **Order bullet lists and other enumerations consciously and consistently**
  - E.g. by application time, by project phase, by importance, by dependency
  - Phrase all bullets or other elements in the same way (verb, noun, -ing form)
- **Don't underestimate the copy editing – tedious, but worth the effort**
  - Be peculiar… any bug you find will not annoy your supervisor and other readers ☺
  - Tackle in phases – figure captions only, indentation only, index only, etc.

Lessons Learned: Industry (Five Cs and Counting)

- **Context matters**
- **Client wants and needs to be distinguished**
- **Stakeholders concerns to be elicited**
- **Common sense to be applied**
- **Collaboration is essential**

(screen caption is hyperlink)



Skills and Traits of Consultants

- **Ability to listen**
  - Active, multiple times, …
- **Ability to ask**
  - Ask the right questions, and ask them right
- **Ability to say no**
  - In a constructive way – almost everything can be built if budget is there

- **Ability to deal with incomplete and conflicting information**
- **Curiosity (domains, people, business models, …)**
- **Get-the-job-done mentality (due date, bug fix, political turmoil)**
- **Ability to travel (schedule, location issues?)**
- **Humor, flexibility, helper attitude; other social skills**

IT Consultant Tenets and Code of Conduct (ZIO Top 5)*

- **Marketing Opens Doors – *Technical Excellence* Creates Opportunities**
  - Conferences, articles, academic degrees, continued education
- ***Responsiveness* Expected by Client**
  - "Blitz chess" metaphor: when the client has been active, you get active too
- ***Context* Matters and *Wants vs. Needs* Differ**
  - Kruchten's Octopus dimensions: do not blindly transfer "best" practices
  - Articulated requirements do not always equal actual requirements
- **End-to-End *Systems Thinking* Required**
  - DevOps, maintenance team, education needs when new tech. is used
- ***Trust* is Foundation for Long Term Success**
  - Establish early and sustain it (needed for critical project situations)
  - Transfer knowledge (to client, to peers), do not hide it

* Inspired by Gregor Hohpe's Beliefs (presented in his ECSA 2014 keynote)

Software Architecture and Software Engineering Resources

IEEE Software

ABOUT BACK ISSUES WRITE FOR US SUBSCRIBE SE-RADIO BLOG JOIN

CURRENT ISSUE: THE ROLE OF THE SOFTWARE ARCHITECT

(screen captions is hyperlink)

The Software Architect's Role in the Digital Age

Gregor Hohpe, Allianz SE

Ipek Ozkaya, Carnegie Mellon Software Engineering Institute

Uwe Zdun, University of Vienna

Olaf Zimmermann, University of Applied Sciences of Eastern Switzerland, Rapperswil

Projects

- Architectural Refactoring for the Cloud (ARC)
- Cloud Knowledge Sources
- Microservices Resources and Positions
- Domain-Driven Design Overview and Links
- DevOps Resources and Positions

Architectural Knowledge Hubs

Online Resources for Software Architects

The November/December 2016 Theme Issue of IEEE Software on the Role of the Software Architect in the Diigtal Age is a good starting point (Guest Editor's Introduction to Theme Issue as PDF).

Websites by thought leaders that we frequently consult (among many others) are:

1. Martin Fowler's Bliki
2. Gregor Hohpe's Ramblings
3. Philippe Kruchten's Weblog
4. Eoin Wood's website and blog at Artechra
5. Michael Stal's software architecture blog
6. The Software Architecture Handbook website by Grady Booch
7. Personal page of Gernot Starke (mostly in German) - arc42, aim42, IT architect profession
8. Technical Reports and other publications in the Digital Library of the Software Engineering Institute (SEI)
9. The Open Group website - IT Architect Certification, TOGAF, ArchiMate, XA
10. Object Management Group (OMG) - UML, SPEM, MDA, CORBA, ADM, KDM
11. IEEE Software, as well as SWEBOK and the very readable standard for architecture descriptions, ISO/IEC/IEEE 42010
12. Academic conferences (software architecture research): WICSA, QoSA, ECSA and online archives: ACM Digital Library, IEEE Xplore and ScienceDirect.

The following conferences have a practitioner focus on all things software architecture are (most of the presentations are available online and can be accessed from the conference websites):

1. SEI SATURN, e.g. SATURN 2013
2. Industry Day at CompArch/WICSA 2011
3. ECSA 2014 also had an Industry Day
4. OOP (most talks in German, presentations not available online by default)
5. SPLASH and OOPSLA (e.g. practitioners reports program at OOPSLA 2008)

If you are new to the field, you can get started by reviewing the arc42 site (in German) or look for architectural guidance and practices in OpenUP. If you have a little more time to study, many excellent books on the topic are available to you, including (but of course not limited to):

1. Software Systems Architecture (Second Edition) by Nick Rozanski and Eoin Woods introduces core architecture concepts, as well as a viewpoint- and perspective-based architecture framework.

SERVICE DESIGN AS A SET OF RECURRING ARCHITECTURAL DECISIONS: PARADIGMS, PRINCIPLES, PATTERNS

Service Design and Service Granularity – BACKGROUND INFORMATION

May 2017

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

Characteristics from L/F Definition Analyzed and Compared

Characteristic	Viewpoint/Qualities/Benefit	SOA Pendant
Componentization via services	Logical Viewpoint (VP): separation of concerns improves modifiability	Service provider, consumer, contract (same concept)
Organized around business capabilities	Scenario VP: OOAD domain model and DDD ubiquitous language make code understandable and easy to maintain	Part of SOA definition in books and articles since 200x (e.g. Lublinsky/Rosen)
Products not projects	n/a (not technical but process-related)	(enterprise SOA programs)
Smart endpoints and dumb pipes	Process Viewpoint (VP): information hiding improves scalability and modifiability	Same best practice design rule exists for SOA/ESB (see e.g. here)
Decentralized governance	n/a (not technical but process-related)	SOA governance (might be more centralized, but does not have to; “it depends”)
Infrastructure automation	Development/Physical VP: speed	No direct pendant (not style-specific, recent advances)
Design for failure	All VPs: robustness	Key concern for distributed systems, SOA or other
Evolutionary design	n/a (not technical but process-related): improves replaceability, upgradeability	Service design methods, Backward compatible contracts

SOA Principles and Patterns vs. Microservices Tenets

Aspect/Capability	SOA Principles and Patterns	Microservices Tenets and Patterns
Core metaphor	(Web) Service, Service Contract	Fine-grained interfaces, RESTful resources
Method	OOAD/UP; SOMA and others	Domain-Driven Design, agile Practices
Architectural principles	Layering, loose coupling, flow independence, modularity	IDEAL Cloud Architectural Principles
Data storage	Information Services (RDB, File)	Polyglot Persistence (NoSQL, NewSQL)
Deployment and hosting	Virtual machines, JEE, SCA; Application Hosting/Outsourcing	Lightweight Containers (e.g., Docker, Dropwizard); Cloud Computing
Build tool chain	n/a (proprietary vendor approaches, custom developed in-house assets, ITIL and other management frameworks)	Decentralized Continuous Delivery
Operations (FCAPS)		Lean but Comprehensive System Management (a.k.a. DevOps)
Message routing, transformation, adaption	Enterprise Service Bus (ESB)	API Gateway, lightweight messaging systems (e.g., RabbitMQ)
Service composition	Service Composition DSLs, POPL	Plain Old Programming Language (POPL)
Lookup	Service Registry	Service Discovery

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

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