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

12th Annual Symposium on Future Trends in Service-Oriented Computing, HPI Research School

Potsdam, April 27, 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) patterns and platforms reached a plateau of maturity and market saturation, microservices are currently emerging as a state-of-the-art implementation approach to SOA that leverages recent advances in software engineering and agile practices such as domain-driven design, continuous delivery and deployment automation. Due to the invariant intricacies and fallacies pertaining to distributed systems, service interface design remains a wicked problem irrespective of currently trending service decomposition paradigms and other market dynamics. Hence, service designers and API managers seek design guidance and reusable architectural knowledge for this problem domain.**
- **This presentation first recapitulates selected SOA principles and establishes seven corresponding microservices tenets. It then reports on the ongoing compilation of a service design pattern catalog that complements previous such approaches, and discusses related tool support. It concludes with a reflection on open research challenges and problems.**

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

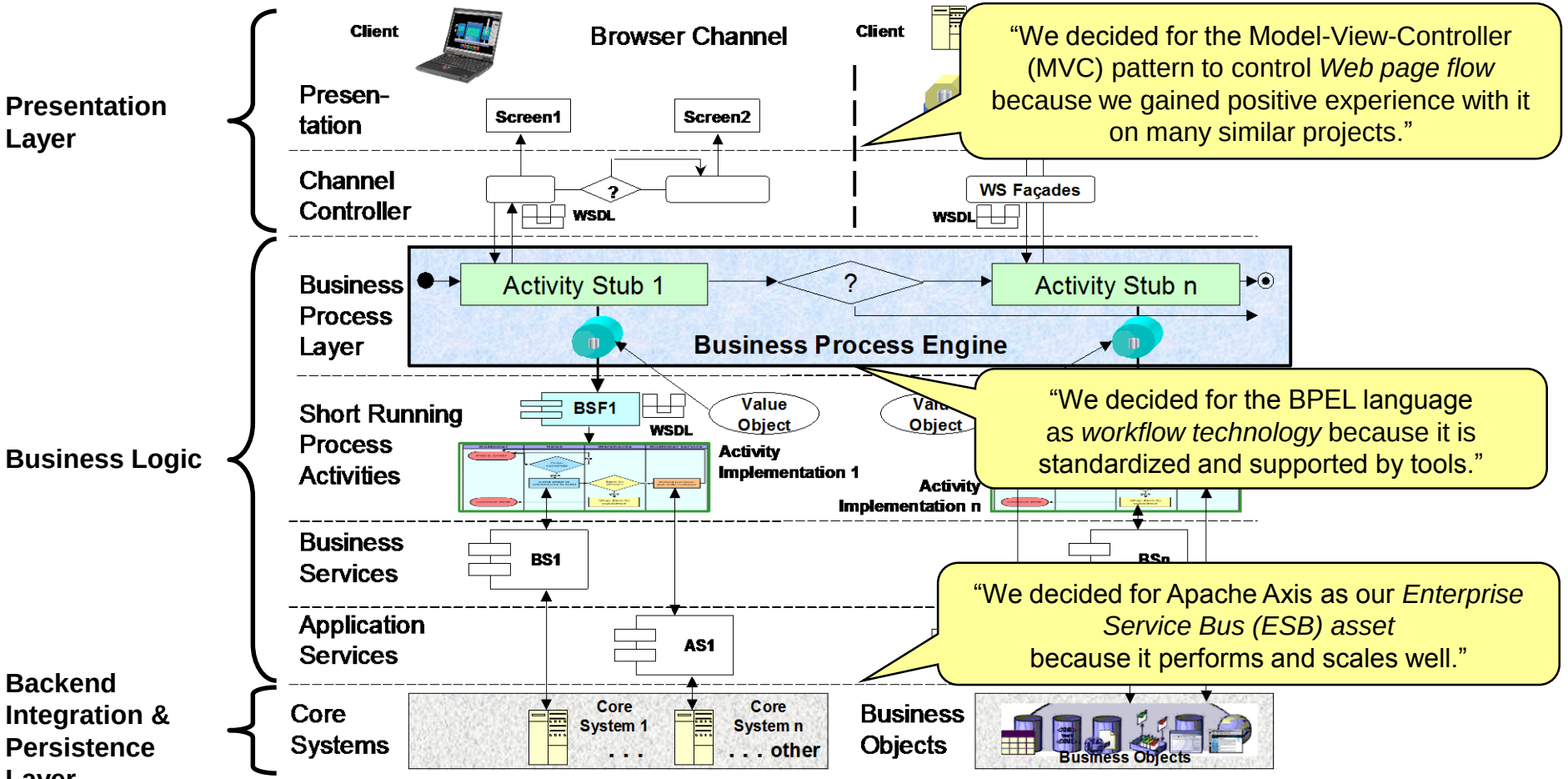
Agenda

1. ***Paradigms:* Service-Oriented Computing (Re-)Visited**
   - Service-Oriented Architecture vs. Microservices Architecture (?)
   - Microservices tenets: agile approach to service realization
2. ***Principles:* From OOAD to SOAD and Agile Architecting**
   - IDEAL cloud application architectures, coupling criteria
   - Architectural Decision Capturing and Sharing (OLAF, AKMAD)
3. ***Patterns:* From Enterprise Application/Integration to Service Design and Conversations**
   - *Interface Representation Patterns (IRP):*
     Pagination, Service Granularity (Business/Technical), Quality of Service
4. **Open Source Tools for Service Design and Arch. Decision Making**
   - ADMentor, Service Cutter
5. **Conclusions and Outlook**
   - Research challenges, vision and roadmap

Position Summary & Key Take Away Messages of this Talk

- **Microservices do not constitute a new style, but services are here to stay**
  - 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 Parameter Tree, Pagination, Wish List**
- **Pattern-centric service design involves architectural decisions that recur**

Process-Enabled Order Mgmt. SOA for Telecom Service Provider



Reference: Zimmermann et al, „SOA and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned", OOPSLA 2005 conference companion, ACM Press, 2005

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.

Business Domain Analyst

- An architectural style which requires a *service provider*, a *service requestor* (consumer) and a *service contract* (a.k.a. client/server).
- 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.

IT Architect

- A *programming and deployment model* realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).

Developer, Administrator

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

The Seven ZIO Tenets for Microservices Implementations of SOA

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.**
5. ***Lightweight containers*** **are used to deploy 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 (a.k.a. *DevOps*).**

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).

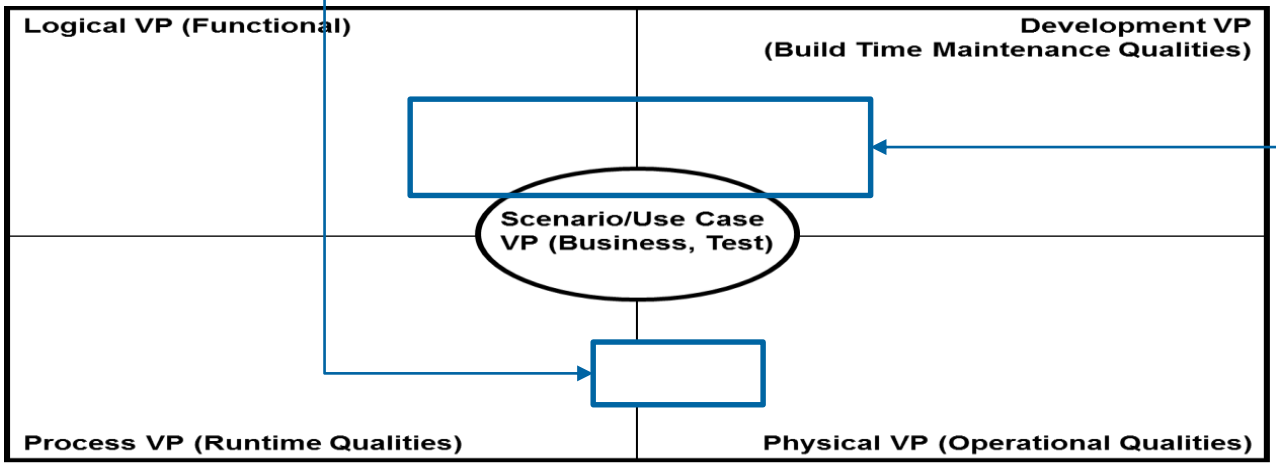
Microservices – An Early and Popular Definition (2014)

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:**

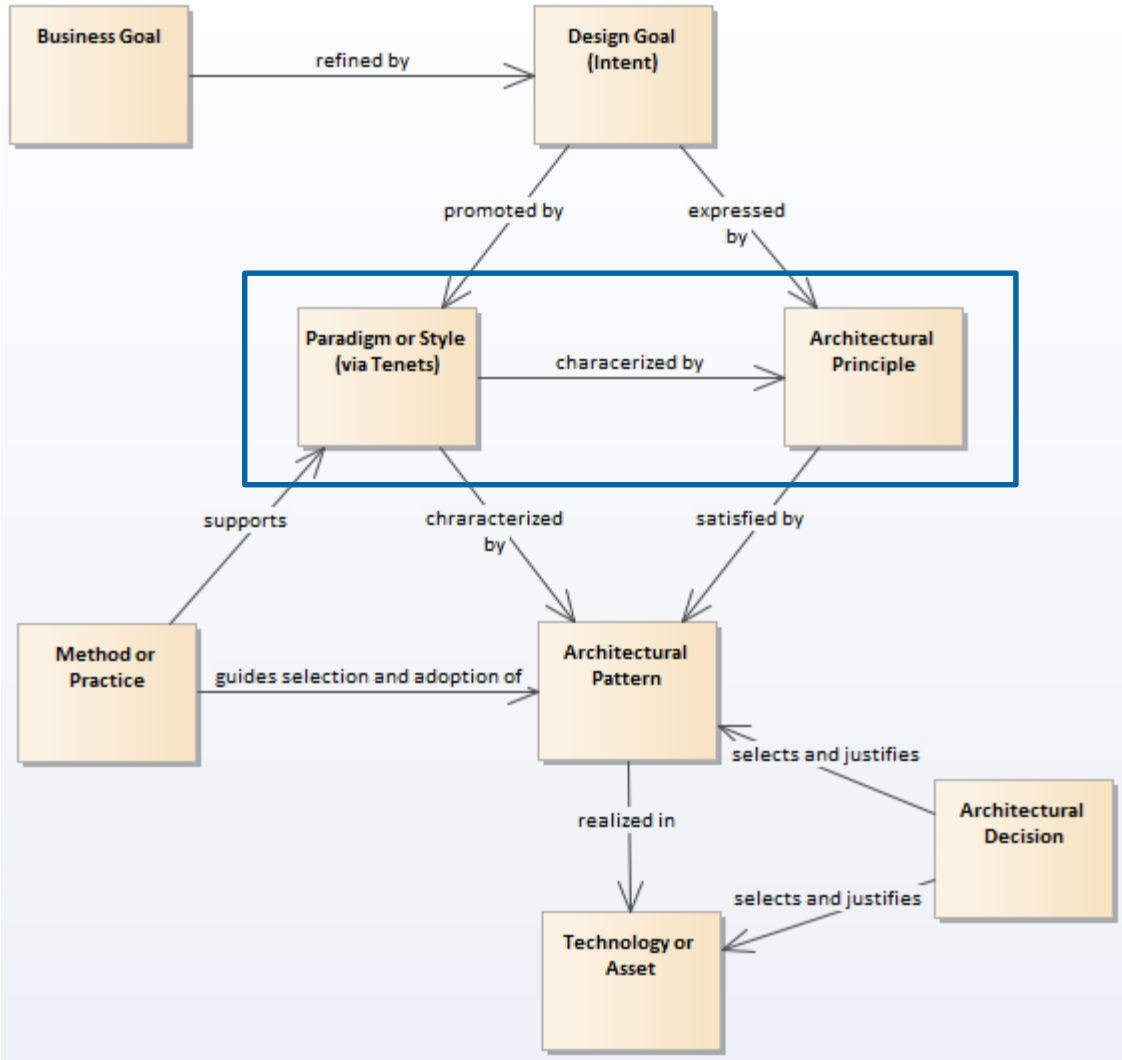
Microservices Definition: 4+1 Viewpoint Mapping (More: CSR&D Paper)

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



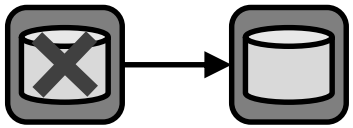
From Tenets and Principles to Patterns and Decisions

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

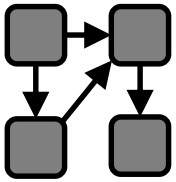


IDEAL Cloud Application Properties (Fehling, Leymann et al.)

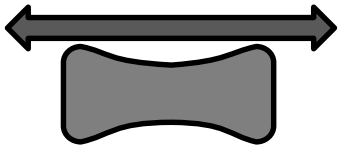
Reference: Cloud Computing Patterns, Springer 2014, http://cloudcomputingpatterns.org/



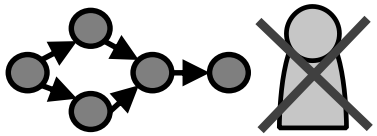
Isolated State: most of the application is *stateless* with respect to:
Session State: state of the communication with the application
Application State: data handled by the application



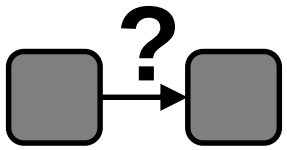
Distribution: applications are decomposed to…
… use multiple cloud resources
… support the fact that clouds are large globally distributed systems



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

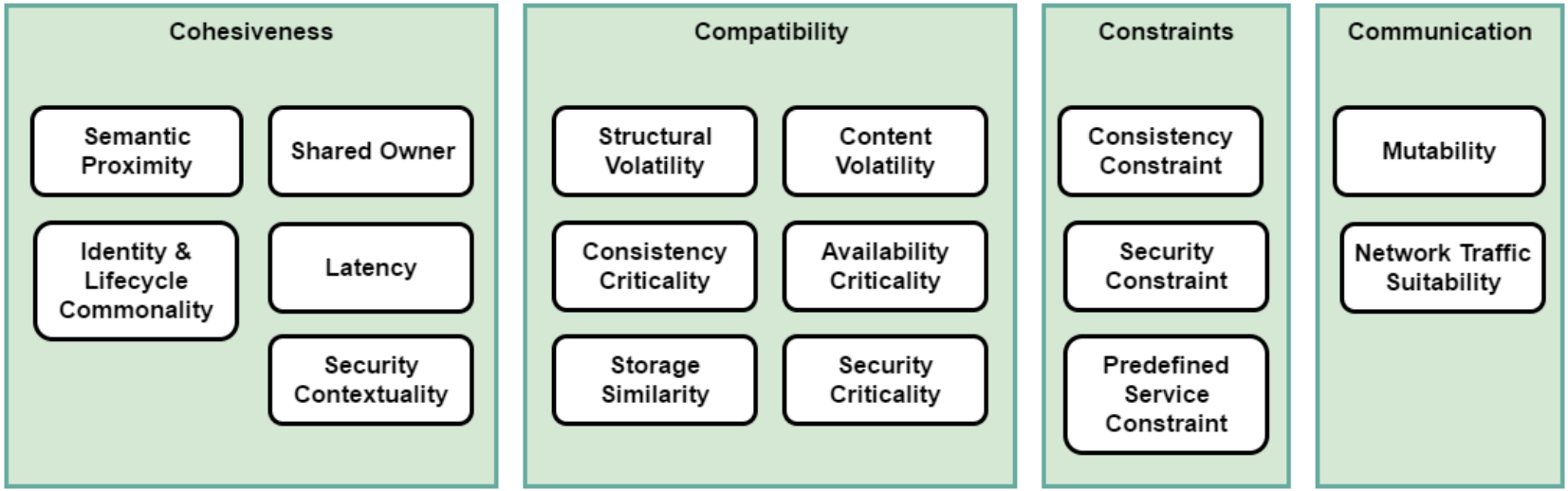


Loose Coupling: influence of application components is limited
Example: failures should not impact other components
Example: addition / removal of components is simplified

SOA Principle and IDEAL Application Property: Loose Coupling

- **Academic contributions (research results):**
  - General software engineering/architecture literature since 1960s/1970s
    - Starting from D. Parnas (modularization, high cohesion/low coupling)
  - ESOCC 2016 keynote by F. Leymann and PhD theses (e.g. C. Fehling):
    - Four types of *autonomy*: reference (i.e., location), platform, time, format
  - WWW 2009 presentation and paper by C. Pautasso and E. Wilde:
    - 12 facets used for a remoting technology comparison, e.g., discovery, state, granularity
- **Practitioner heuristics (a.k.a. coupling criteria) scattered 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, …

Coupling Criteria (CC) in “Service Cutter” (ESOCC 2016 Paper)

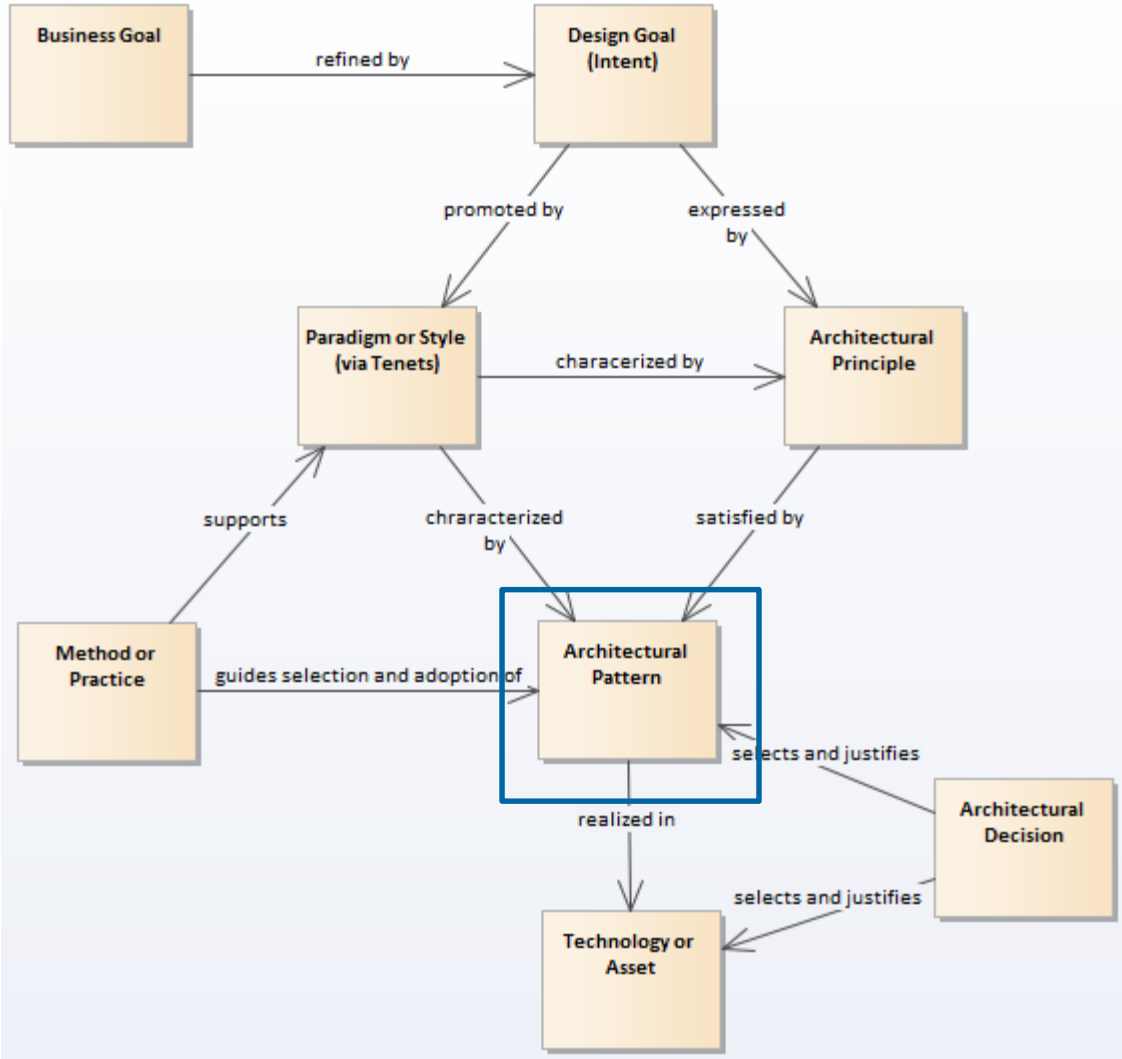


Full descriptions in CC card format: https://github.com/ServiceCutter/ServiceCutter/wiki/Coupling-Criteria

- **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 Tenets and Principles to Patterns and Decisions

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



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)?

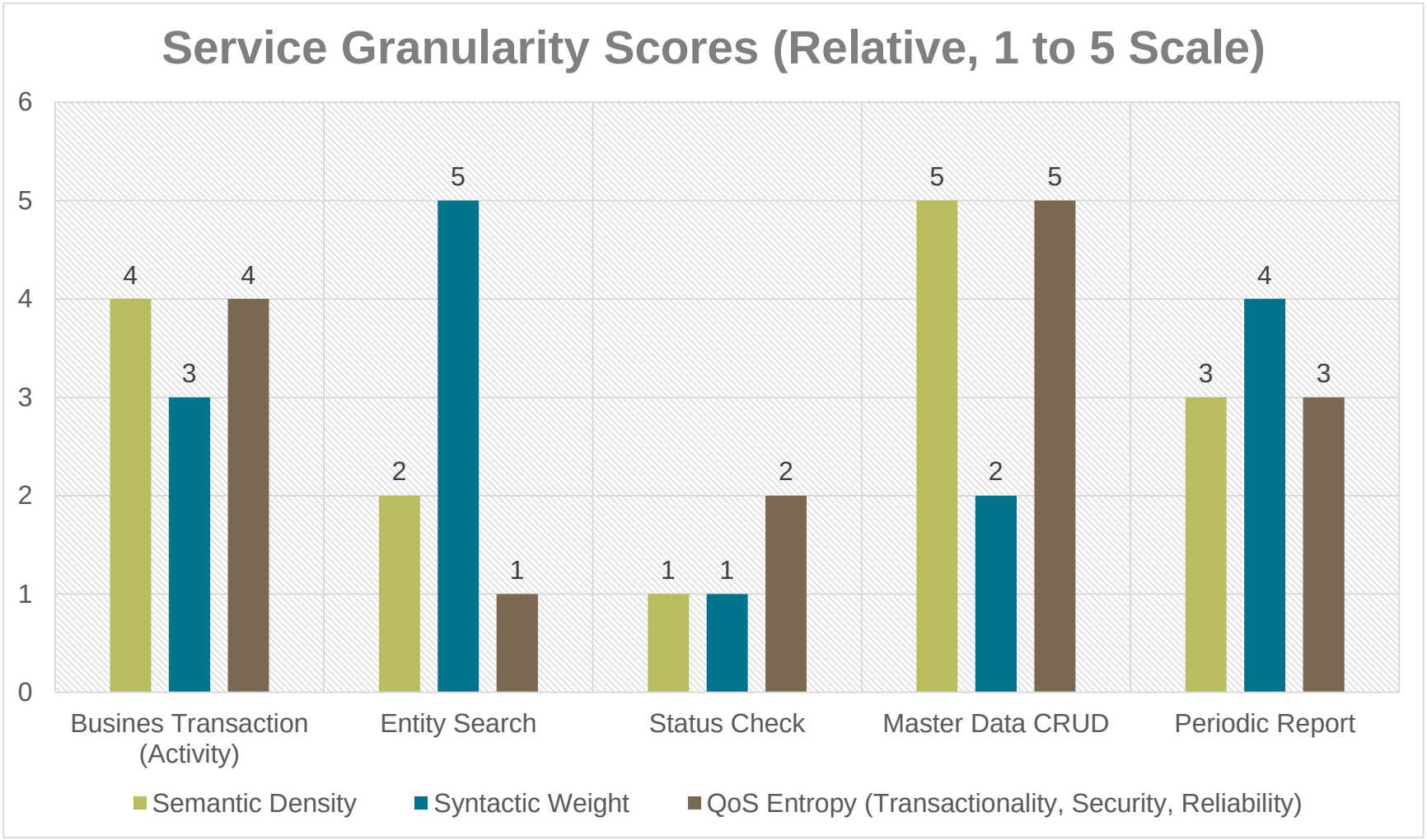
- **Example A: *Exchange Rates* in YaaS/Hybris (SAP):**
  - https://devportal.yaas.io/services/exchangerates/latest/

- **Example B: *Create 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 interface dependencies and their properties (e.g. consistency)
    - Security, reliability, consistency requirements (coupling criteria)

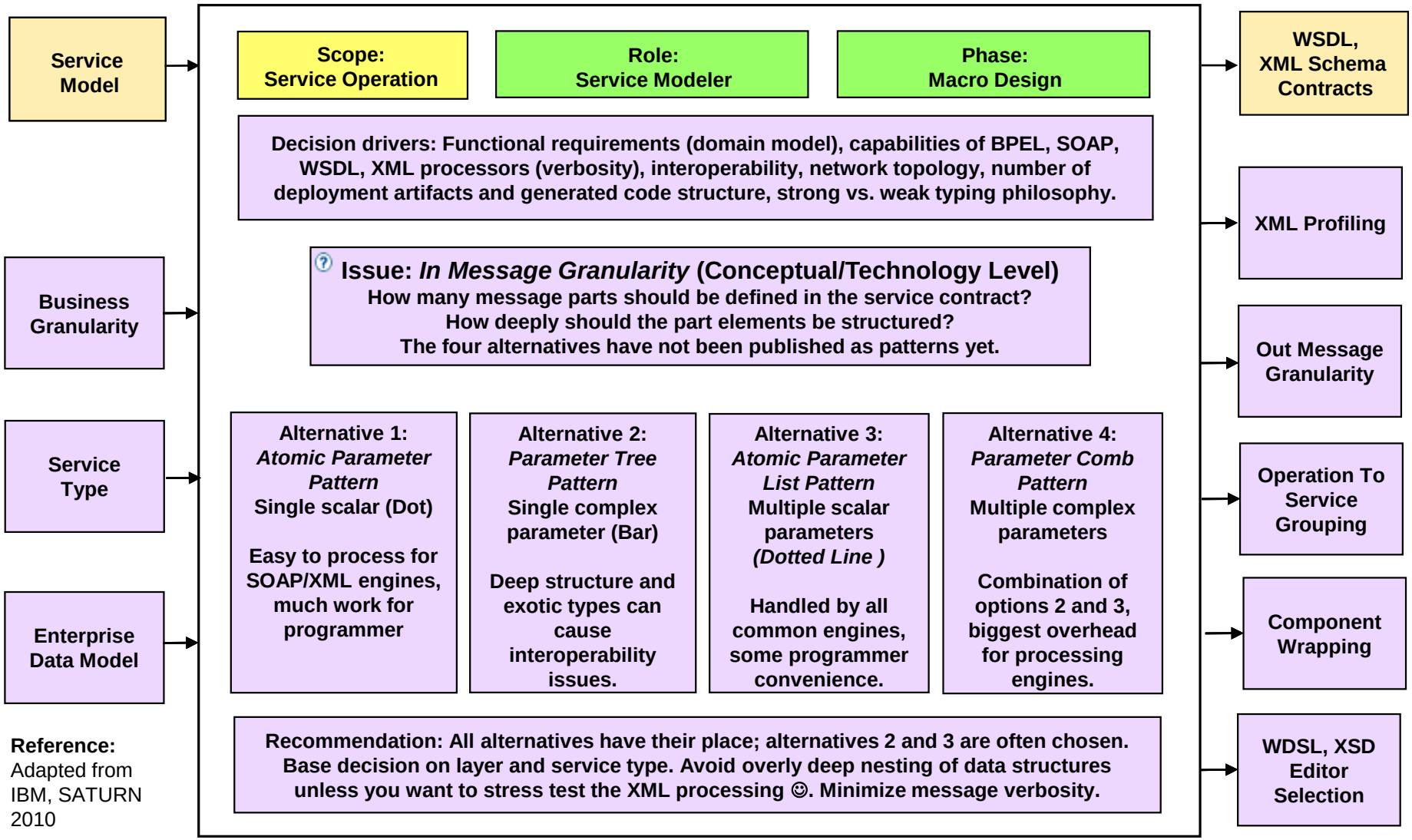
Granularity Scores by Service Pattern and Granularity Type



Granularity Types and Criteria – Findings

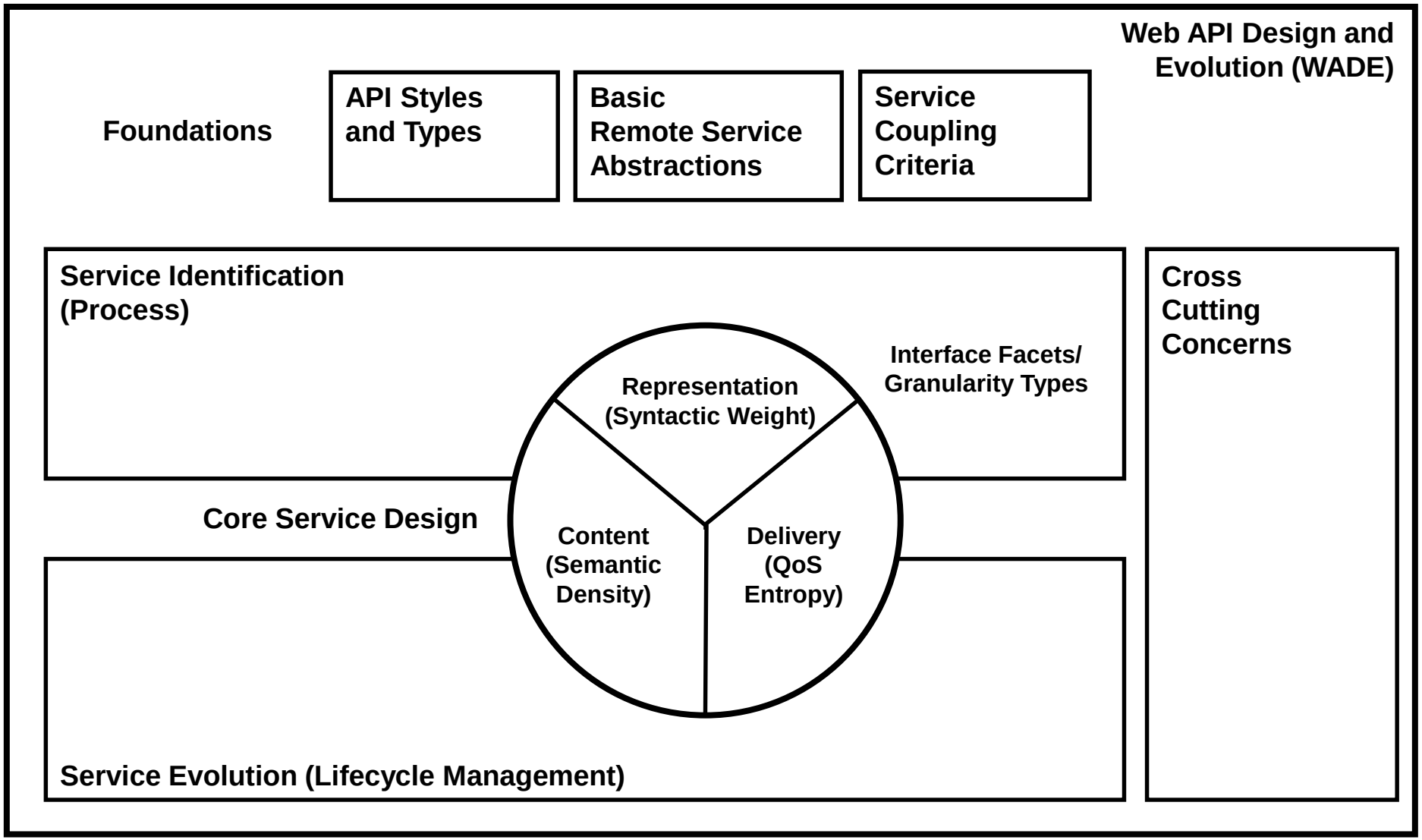
- **Granularity is property of service contract exposed by a service provider**
  - Not an exact measure/metric, but a heuristic/an indicator of modularity and cohesion (on different levels of abstraction)
  - Business granularity vs. technical granularity (syntax, QoS)
- **Can't really tell the "right" size w/o use cases and (de)coupling criteria – "it depends":**
  - 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
- **Sometimes granularity is also seen as an architectural principle:**
  - https://en.wikipedia.org/wiki/Service_granularity_principle

Service Granularity Patterns: In/Out Message Structure



Reference:
Adapted from IBM, SATURN 2010

Towards an Interface Representation Pattern Language (IRP)



Candidate Patterns in IRP (Work in Progress)

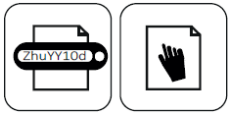
Category				
Foundations	Vertical Integration, Horizontal Integration	Public API	Community API	Solution-Internal API
Process	Contract First	Static Discovery	Dynamic Discovery	Service Model
Representation	AtomicParameter (Single Scalar, Dot)	Parameter Tree (Single Complex)	Atomic Parameter List (Mult. Scalars, Dotted Line)	ParameterComb (Multiple Complex)
	Pagination, Page	Query Parameter	Cursor	Offset
	Wish List	Request Deck	Metadata Parameter	Annotated Parameter List
Content Semantics	Command	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, submitted to EuroPLOP 2017

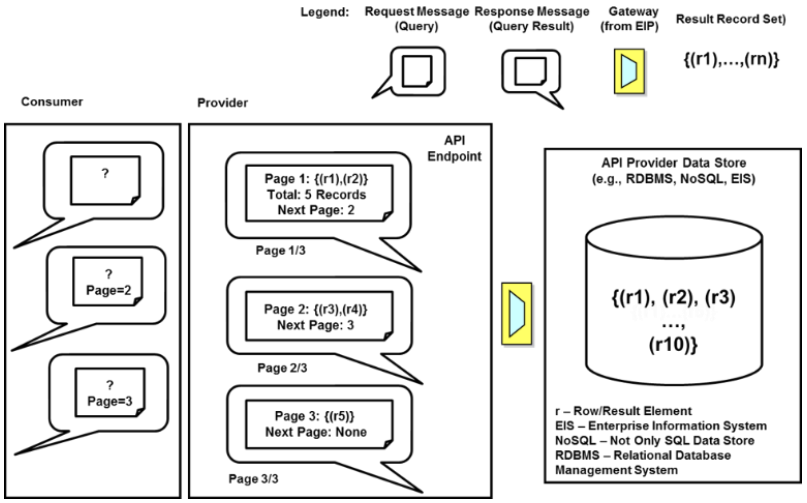
Example IRP: Pagination (1/2)

- **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)

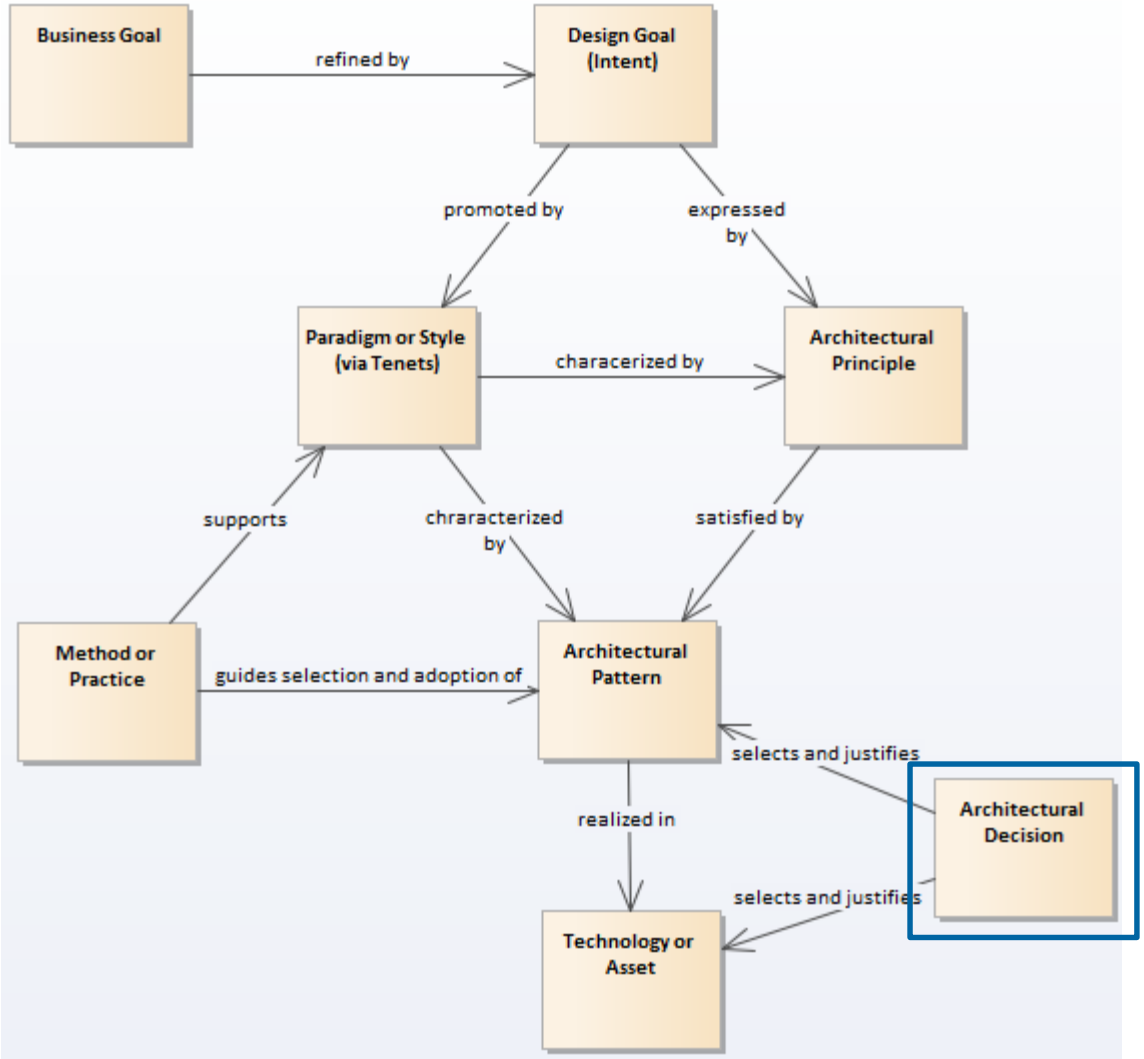


- **Solution**
  - Divide large response data sets into manageable and easy-to-transmit chunks.
  - *Send only partial results in the first response message and inform the consumer how additional results can be obtained/retrieved incrementally.*
  - Process some or all partial responses on the consumer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



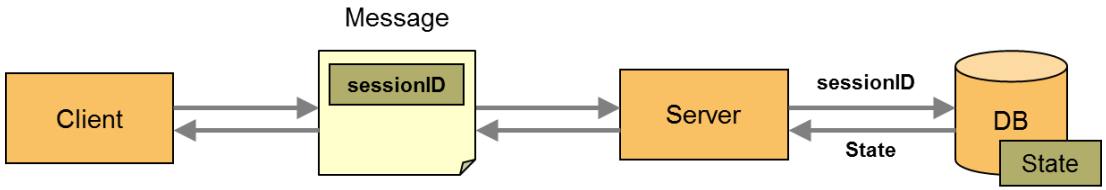
From Tenets and Principles to Patterns and Decisions

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



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**

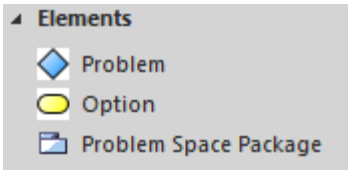
"In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State Pattern from the PoEAA book (and against Client Session State or Server Session State) to achieve cloud elasticity, accepting that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

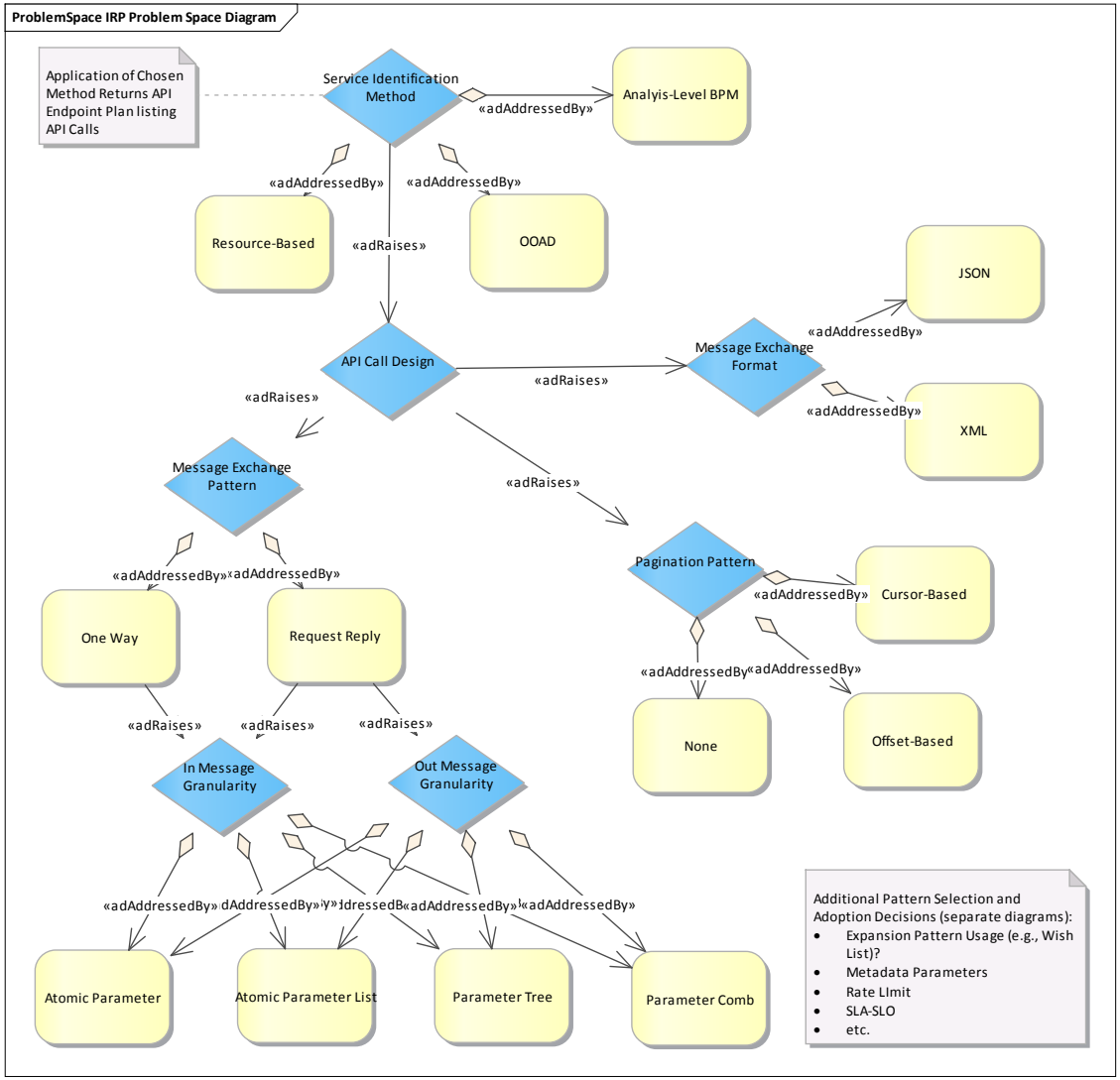
Curate {decision need, solutions, qualities} for reuse – but *not* the actual decision outcomes

- "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

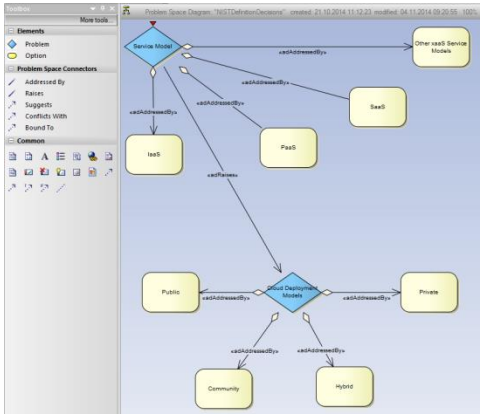
- **Patten selection and adoption qualifies as AD making**
  - Rationale to be captured: qualities, conformance with principles, etc.
- **Guidance through service design space via problem-option pair modeling**
  - In ADMentor





ADMentor Tool (AddIn to Sparx Enterprise Architect)

- **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**

Research Problems in ESOCC 2007 Keynote – Still Open!

Leveraging Reusable Architectural Decision Models as a Design Method for Service-Oriented Architecture Construction

ECOWS Keynote
Nov 28, 2007

Olaf Zimmermann
IBM Zurich Research Lab
olz@zurich.ibm.com

Software Service Engineering

DAGSTUHL REPORTS

(screen caption clickable)

(screen caption clickable)

IBM

The Need for Service Modeling and Engineering Research

- Focus @ ECOWS 2007 and other conferences:
  - Dynamic matchmaking, automated service composition, semantics notations
  - Only 2/24 ECOWS 2007 papers have a pure design time focus!
- Service modeling (SOAD) is where OOAD was in ~1995
  - Several methods emerging, none of them is complete (page count!)
  - Many overlapping techniques – no "silver bullet", no consensus, not actionable
  - None of them addresses detailed design concerns based on quality attributes
- Who advances state of the art in…
  - … IDEs for programming without call stack: resolving architectural forces, refactoring to services, build time matchmaking?
  - … quality metrics for interface granularity and other design issues?
  - … resolving tensions between different forms of pre- and postconditions in contract design (human service designer vs. logic-based)?

➔ Dagstuhl seminar proposed: "Software Service Engineering"

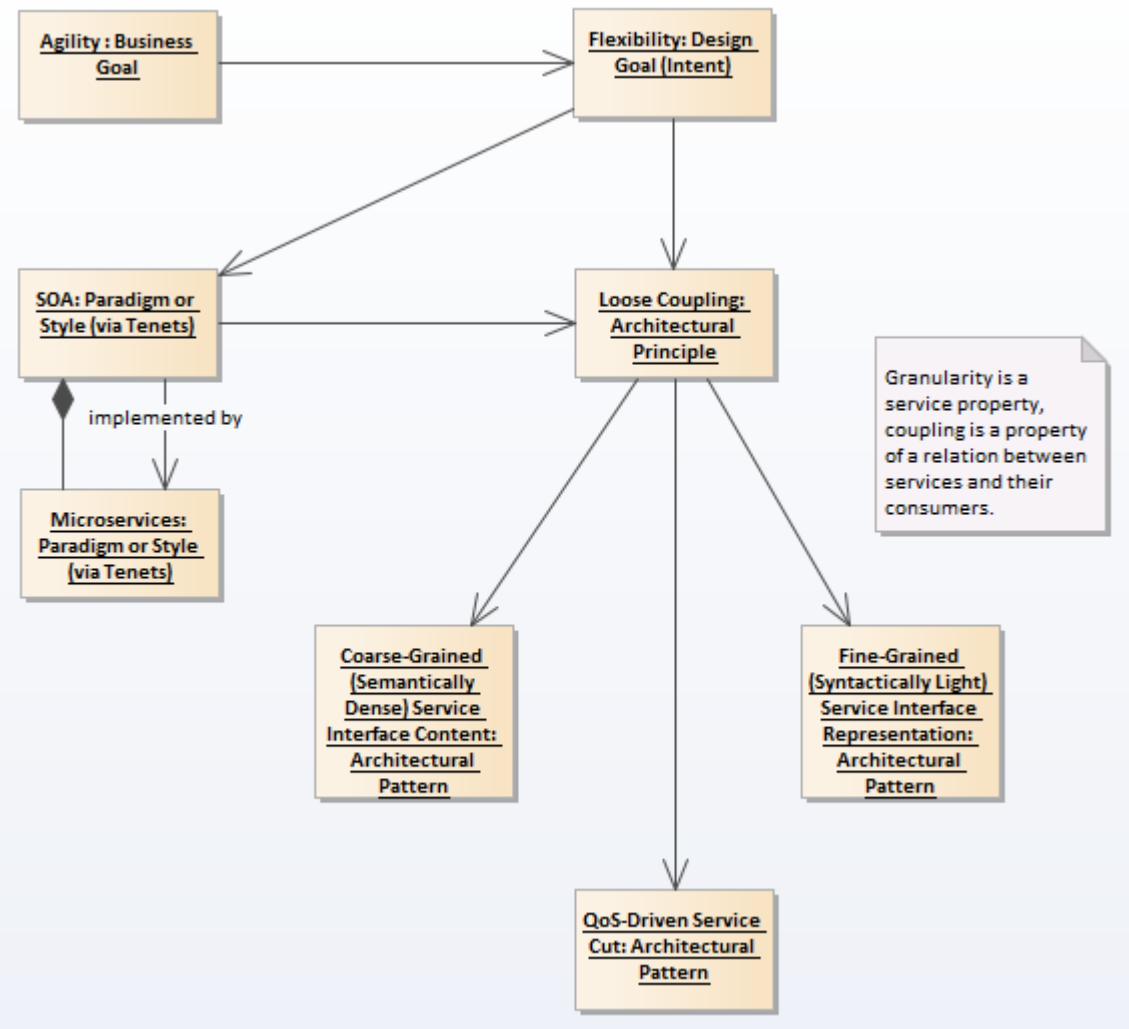
39 Zurich Research Laboratory

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 trends in service-oriented computing/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)

Summary



- **Thank you vey much!**
- **Feedback?**
  - Questions?
  - Comments?
  - Ideas?
- **Next Steps?**

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

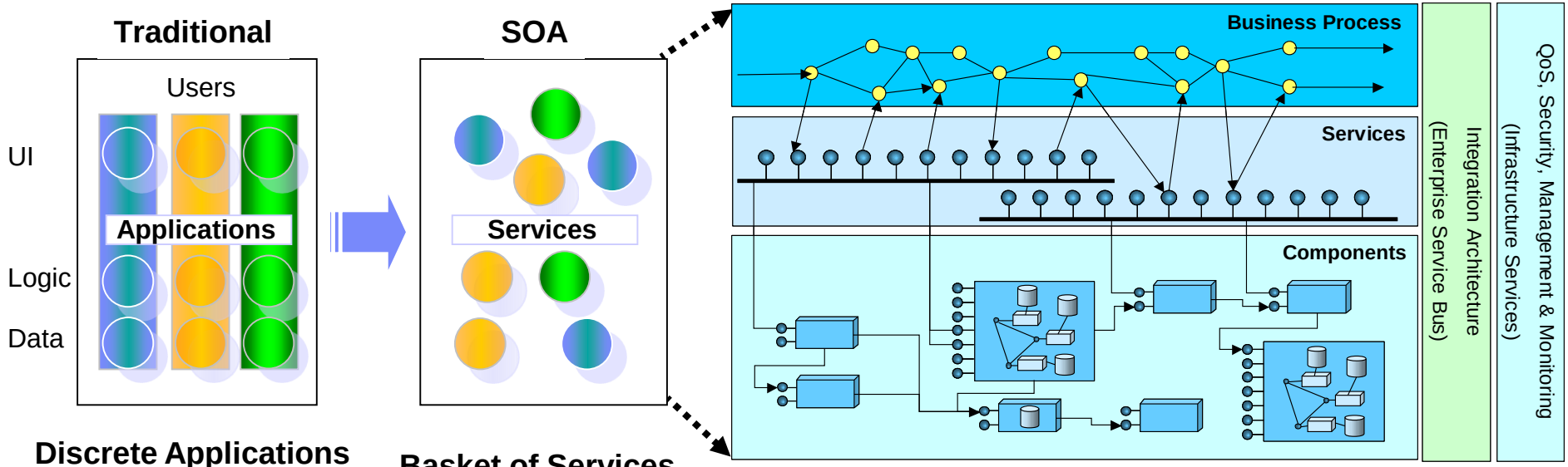
Service Design and Service Granularity – BACKGROUND INFORMATION

April 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

SOA Foundation: Partitioning into Components and Services



Discrete Applications (Two or Three Tiers)

Basket of Services

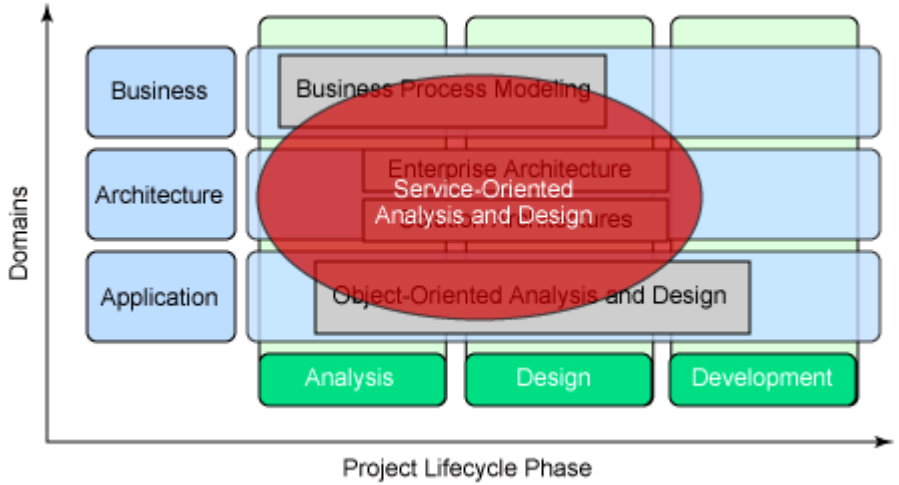
Layering based on IBM SOA reference architecture

Example:
An insurance company uses three SAP R/3, MS Visual Basic, and COBOL applications to manage customer information, check for fraud, and calculate payments. The user interfaces (UIs) are the only access points.

A multi-step, multi-user business process for claim handling, executing in IBM WebSphere, is supposed to reuse the functions in the existing applications. How to integrate the new business process with the three legacy applications in a flexible, secure, and reliable way?

Architectural Principles and Heuristics

- **Object-Oriented Analysis and Design (OOAD), Component-Based Software Engineering (CBSE)**
  - Single Responsibility Principle (SRP) by R. Martin (revisited by K. Henney)
  - Fowler's First Law of Distributed Objects: Don't distribute your objects
  - Cheesman/Daniels, Catalyst approach, Larman GRASP
- **SOAD vision (2004)**
  - EAM + BPM + OOAD
  - Coarse *and* fine grained granules
  - Several proposals from industry responded to call for methods (none of which seem to sustain)



- **Rules of Thumb (ZIO)**
  - POINT principles for API design
  - "If in doubt leave it out" (metamodeling, method engineering and adoption)
  - "Worst first" (architectural decision making)

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.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

Bachelor Thesis Fall Term 2015

Software

Service Cutter

Lukas Kölbener

Michael Gysel

Advisor: Prof. Dr. Olaf Zimmermann

Co-Examiner: Prof. Dr. Andreas Rinkel

Project Partner: Zühlke Engineering AG

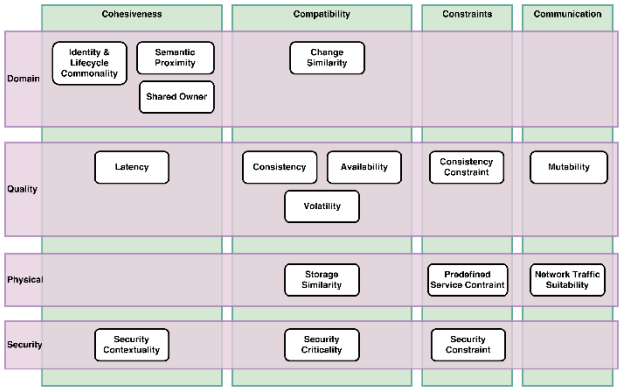
A Software Architect's Dilemma….

How do I split my system into services?

Step 1: Analyze System

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

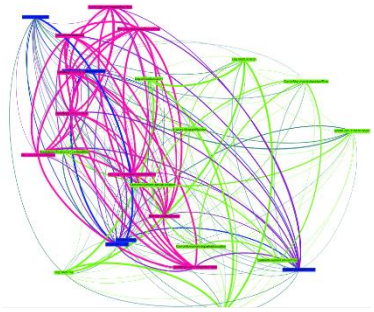
Coupling information is extracted from these artifacts.



The catalog of 16 coupling criteria

Step 2: Calculate Coupling

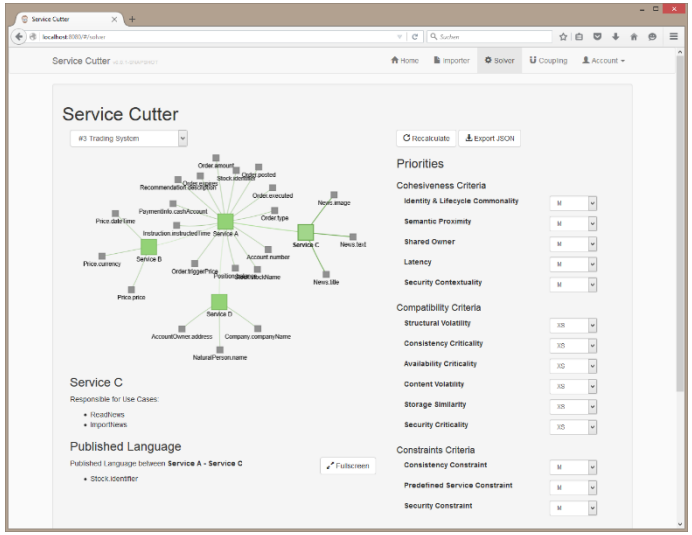
- Data fields, operations and artifacts are nodes.
- Edges are coupled data fields.
- Scoring system calculates edge weights.
- Two different graph clustering algorithms calculate candidate service cuts (=clusters).



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

Other Research Projects: Architectural Refactoring Content/Tool

Architectural Refactoring for the Cloud (ARC)

Pattern- and Decision-based Modernization of Existing Applications for the Cloud

Project results (overview):

- Decision- and task-centric definition of term architect
- Quality story templates
- Architectural refactoring template
- Catalog of general-purpose architectural refactorings
- Architectural best practices, e.g. IDEAL and FIT cloud
- POINT criteria for API design and management; servi
- ADMentor, prototypical tool support for architectural reuse, and Cloud Design Guidance Model crafted with

Selected project results are featured in an OOP 2014 pr

- **Pattern language for Web API Design and Evolution?**
- **Lean Decision Backlog?**

(screen captions clickable)

Reviewed papers in journals and proceedings

- (50) M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann, Service Cutter: A Systematic Approach to Service Decomposition, Proc. of ESOCC 2016, Springer LNCS.
- (49) O. Zimmermann, Microservices Tenets: Agile Approach to Service Development and Deployment, Overview and Vision Paper., Proc. of Symposium and Advanced School on Service-Oriented Computing (SummerSoC), 2016
- (48) H. Muccini, K. E. Harper, R. Heinrich, J. Bosch, N. Plouzeau, O. Zimmermann, L. Baresi, V. Cortellessa, Welcome Message from the Chairs of WICSA, QoSA and CBSE. Proc. of CBSE 20116.
- (47) O. Zimmermann, C. Pautasso, G. Hohpe, B. Woolf, A Decade of Enterprise Integration Patterns: A Conversation with the Authors., IEEE Software 33(1): 13-19 (2016)
- (46) O. Zimmermann, Architectural Refactoring for the Cloud: a Decision-Centric View on Cloud Migration, Proc. of Symposium and Advanced School on Service-Oriented Computing (SummerSoC), 2015
- (45) O. Zimmmermann, Metrics for Architectural Synthesis and Evaluation: Use Cases and Compilation by Viewpoint, Proc. of ICSE SAM 2015 (Second International Workshop on Software Architecture and Metrics)

Software Architecture and SoC Resources

IEEE Software

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

CURRENT ISSUE: THE ROLE OF THE SOFTWARE ARCHITECT

(screen captions clickable)

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.