

Recurring Architectural Decisions

A Context-Specific Guide through SOA And Cloud Design

SATURN 2010 Tutorial 7
Minneapolis, MN
May 21, 2010

Dr. Olaf Zimmermann
Executive IT Architect
olz@zurich.ibm.com



© 2010 IBM Corporation

Abstract

In this tutorial, we present excerpts from a Service-Oriented Architecture (SOA) and cloud design guidance model harvested from large-scale enterprise applications which have been subject to change since they went live several years ago. This guidance model comprises of 500+ issues (design problems) commonly encountered in SOA and cloud design, 2000+ known solutions (design alternatives) to these design problems, and best practices recommendations when to select which solution. We also outline how to support architectural decision capturing and reuse in existing and emerging modeling tools.

The decisions in the guidance model deal with selection and adoption of architectural patterns, with selection and profiling of technology standards, as well as with selection and configuration of vendor and open source assets. Examples of recurring SOA design issues featured in the tutorial are the selection of integration style and message exchange patterns, the definition of business and system transaction boundaries, as well as service and operation granularity issues. Recurring cloud design issues pertain to cloud virtualization layers (infrastructure vs. platform vs. software), workload type and application genre to be virtualized, and private vs. public cloud exposure.

Attendees of the tutorial will gain:

- An understanding of the importance of architectural decision capturing and sharing.
- The ability to identify required and recurring decisions in UML and other models, to make these decisions more consciously, and to enforce them adequately.
- Familiarity with a decision-centric approach to architecture design which starts from software quality attributes and architectural patterns.

Target Audience

- Researchers with a focus on architecture design methods and architectural knowledge
- Practicing software architects, particularly application, integration, and infrastructure architects
- Students with an interest in real-world architecture design problems and commonly chosen solutions to them

Learning Objectives

- Having attended this tutorial, you will:
 - Understand the importance of architectural decision making and the value of proactive architectural decision modeling.
 - Be able to identify required and recurring decisions in architectural patterns and other artifacts, be able to make these decisions more consciously, and be able to enforce them adequately based on their type.
 - Be in a position to apply a decision-centric approach to architecture design which starts from quality attributes and architectural patterns.
 - Be aware of the prevalent architectural decisions recurring in SOA and cloud design.

Agenda Overview – Modules, Objectives, Exercises

Module	Learning Objective (Architectural Decision Reuse)	Learning Objective (SOA Design)	Exercise
<i>Module 1: SOA Patterns and Decisions</i>	Recapitulate fundamental concepts: 1) quality attributes 2) architectural patterns 3) architectural decisions	Understand the patterns defining SOA as an architectural style	Reflection on today's decision capturing practices
<i>Module 2: Architectural Decision Modeling with Reuse</i>	SOA Decision Modeling (SOAD) framework overview SOAD meta model (issues, alternatives, outcomes), SOAD tool overview	n/a	Multiple choice exercise on issues, alternatives, and outcomes
<i>Module 3: Reusable Architectural Decision Model (RADM) a.k.a. Guidance Model for SOA</i>	Become familiar with guidance model organization (refinement levels, architectural layers)	Become familiar with key SOA and cloud design issues that recur and commonly used solutions (alternatives)	Multiple choice exercise on levels and layers in RADM for SOA guidance model
<i>Module 4: Case Study</i>	Recapitulate and apply concepts from Modules 1 and 2	Recapitulate and apply RADM for SOA content from Module 3	Make and capture selected SOA decisions in reusable manner
<i>Background reading</i>	Advanced SOAD concepts: RADM tailoring (search & filter) Managed issue list Decision harvesting advice (process, quality heuristics)	Broader and deeper coverage of SOA design issues and alternatives, pattern-based decision identification	Self study (e.g., [Zim09], [SAKM09])

SATURN 2010 Tutorial

Module 1: SOA Patterns and Decisions



Agenda for Module 1

- Introduction
- Case 1: Core banking SOA
- Software architecture fundamentals
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- Case 2: Order management in the telecommunications industry
- SOA patterns and decisions
 - Service Consumer-Provider Contract
 - Enterprise Service Bus (ESB)
 - Service Composition
 - Service Registry

Agenda for Module 1

- **Introduction**
- Case 1: Core banking SOA
- Software architecture fundamentals
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- Case 2: Order management in the telecommunications industry
- SOA patterns and decisions
 - Service Consumer-Provider Contract
 - Enterprise Service Bus (ESB)
 - Service Composition
 - Service Registry

Software Architecture in a Nutshell

- **Responsibilities** of a *software architect* in custom application development:
 - Synthesizes technical solution from requirements (supported by methods)
 - Estimates development efforts and technically leads project
 - Coaches developers and other technical staff

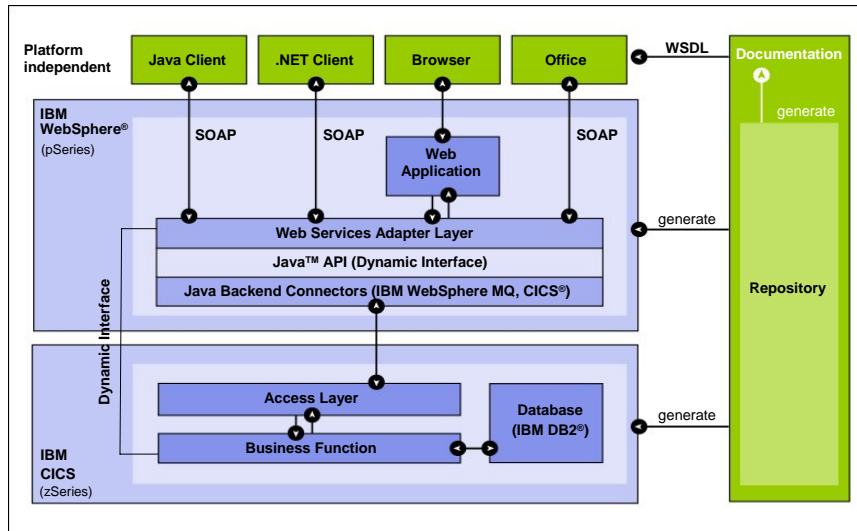
- **Key concepts:** *Quality attributes, architectural patterns, architectural decisions*
 - Quality attributes are architecturally significant requirements
 - ... that drive selection and adoption of architectural patterns
 - ... which is captured in the form of architectural decisions

- **Foundations** date back to 1990s (industry and academia participating)
 - Bass et al. “Software Architecture in Practice”, Shaw/Garlan “Software Architecture...”
 - Buschmann et al. “Patterns of Software Architecture” (POSA)

Agenda for Module 1

- Introduction
- **Case 1: Core banking SOA**
- Software architecture fundamentals
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- Case 2: Order management in the telecommunications industry
- SOA patterns and decisions
 - Service Consumer-Provider Contract
 - Enterprise Service Bus (ESB)
 - Service Composition
 - Service Registry

Case 1: Core Banking SOA with Web Services



Some of the Required Architecture Design Activities

Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., **Pattern-Oriented Software Architecture – a System of Patterns**. Wiley, 1996

- Selection of top-level integration pattern
 - BROKER from “Patterns of Software Architecture” (POSA) is an obvious choice
- POSA suggests six steps to implement the pattern:

“(1) Define an object model. (2) Decide which type of component interoperability the system should offer, binary or Interface Description Language (IDL). (3) Specify the APIs the broker component provides for collaborating with clients and servers. (4) Use proxy objects to hide implementation details from clients and servers. (5) Design the broker component (6) Develop IDL compilers.”

Step (5) has nine sub steps: “(5.1) On-the-wire protocol, (5.2) Local broker, (5.3) Direct communication variant, (5.4) (Un)marshalling, (5.5) Message buffers, (5.6) Directory service, (5.7) Name service, (5.8) Dynamic method invocation, (5.9) The case in which something fails”.
- This is helpful, but *which other patterns are suited to implement the steps? What about the required technology choices? Product selections?*

Agenda for Module 1

- Introduction
- Case 1: Core banking SOA
- **Software architecture fundamentals**
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- Case 2: Order management in the telecommunications industry
- SOA patterns and decisions
 - Service Consumer-Provider Contract
 - Enterprise Service Bus (ESB)
 - Service Composition
 - Service Registry

Quality Attributes (QAs)

- *How does a system provide its functionality (not what it does)*
 - Reliability, usability, efficiency (e.g., performance, scalability), maintainability, and portability areas defined in ISO/IEC specification 9126-2001
- Requirements in core banking case stating QAs tacitly or explicitly:
 - QA “Accuracy”: e.g. money transactions must not be lost, account balances maintained
 - QA “Efficiency” (performance): response times, throughput
 - QA “Interoperability”: multiple frontend platforms to be supported (Web, branch offices)
- Practical challenges:
 - Many attributes, many conflicts between them; many attributes hard to quantify
 - Often under-specified (unknown) or over-specified (overly ambitious)
 - Often stated on inadequate level of abstraction, e.g., per system (not per function/activity)

Common/Key Challenges in Enterprise Application Design

Zimmermann O., **An Architectural Decision Modeling Framework for SOA Design**.
Dissertation.de, 2009

- **User and channel diversity**
 - ISO/IEC 9126-1:2001 QA: *Usability* (among others)

- **Process and resource integrity**
 - ISO/IEC 9126-1:2001QA: *Accuracy* (among others)

- **Integration needs**
 - ISO/IEC 9126-1:2001QA: *Interoperability* (among others)

- **Semantics (here: conceptual dissonances in domain models)**
 - ISO/IEC 9126-1:2001QA: *Functionality* (among others)

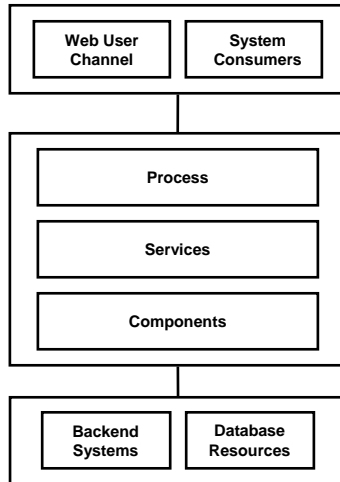
Architectural Patterns

- Proven, common solutions to known, recurring architecture design problems:
 - Context captured by *intent* section
 - QAs discussed in *forces* section
 - A *problem statement* is given, often in question form
 - There is a sketch of the *solution* (not a complete design!)

- Examples in core banking SOA (case 1):
 - “Layers” pattern from [POSA] structures the architecture overview diagram
 - See <http://www.vico.org/pages/PatronsDisseny/Pattern%20Layers/index.html>

- Practical challenges:
 - Many eligible pattern languages
 - Link to requirements covered insufficiently
 - Transition to platform-specific design not addressed (or only in the form of examples)

Layers Pattern Applied to SOA Design



- Humans and other applications as primary actors, presentation logic
- Long running business processes
 - E.g. Loan processing
 - E.g. Supply chain management
- Services and components as atomic units of reuse (business activities)
 - E.g. Check creditworthiness
 - E.g. Order bill of material
- Other systems as secondary actors, persistent storage

Architectural Decisions

- Capture rationale justifying a design, in addition to design itself
- Example in core banking case:
 - “We selected the Layers pattern to make the core banking SOA future proof, e.g., to be able to add retail banking channels in a flexible manner”
 - See <http://www.ibm.com/developerworks/architecture/library/ar-knowwiki1>
- Practical challenges:
 - Retrospective decision capturing takes time and may not yield sufficient benefits to become as broadly accepted in practice as it deserves
 - Relation to other architectural concepts and viewpoints (quality attributes, patterns) not fully understood and supported in methods and tools yet

Agenda for Module 1

- Introduction
- Case 1: Core banking SOA
- Software architecture fundamentals
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- **Case 2: Order management in the telecommunications industry**
- SOA patterns and decisions
 - Service Consumer-Provider Contract
 - Enterprise Service Bus (ESB)
 - Service Composition
 - Service Registry

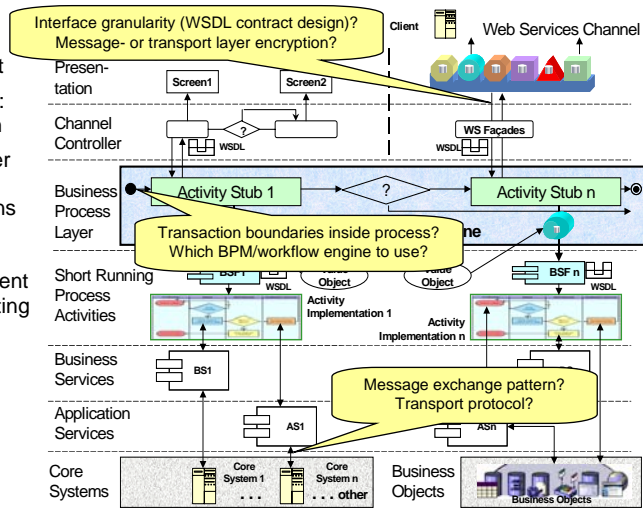
Case 2: Multi-Channel Order Management SOA in the Telecommunications Industry

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



Recurring Service Design Issues and Decisions

1. Selection and adoption of *application and integration patterns*
 2. Technology choices – *protocols, containers, operating systems, etc.*
 3. Commercial and open source *asset selection and configuration*
- Hundreds, if not thousands, of such decisions per *role, phase, scope*:
 - Many decisions already made before project starts (numerous stakeholders)
 - By client – previous projects, legacy systems Booch G., **Handbook of Software Architecture**, <http://www.booch.com/architecture>
 - By other parties – software vendor, IT consultants Booch G., **Handbook of Software Architecture**, <http://www.booch.com/architecture>
 - Many *forces* that influence the decision making, often contradictory [Booch]
 - Numerous complex *dependencies* between the decisions [Kruchten]
 - Never enough time to analyze the *alternatives (options)* in detail
 - Pattern literature is overwhelming Kruchten P. et al, **Building Up and Reasoning About Architectural Knowledge**, QOSA 2006

Agenda for Module 1

- Introduction
- Case 1: Core banking SOA
- Software architecture fundamentals
 - Quality attributes
 - Architectural patterns
 - Architectural decisions
- Case 2: Order management in the telecommunications industry
- **SOA patterns and decisions**
 - **Service Consumer-Provider Contract**
 - **Enterprise Service Bus (ESB)**
 - **Service Composition**
 - **Service Registry**

What is a Service-Oriented Architecture (SOA)?

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.k.a. client) and a *service requestor* (a.k.a. consumer or server).
- A set of architectural patterns such as *service consumer-provider contract*, *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.
- A *programming and deployment model* realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).

Business Domain Analyst

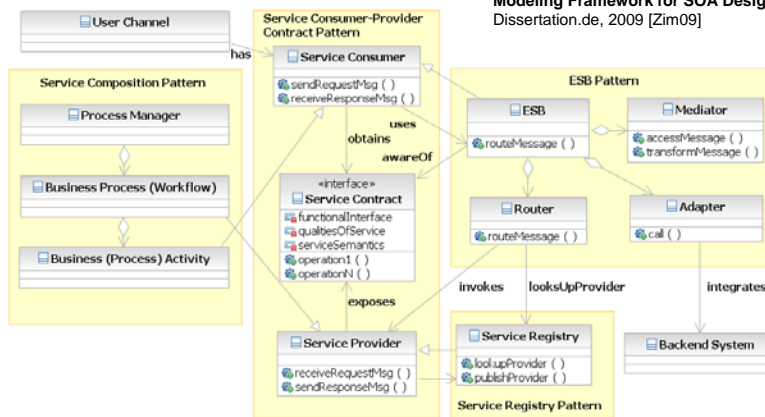
IT Architect

Developer, Administrator

Adapted from: [IBM SSS]

SOA Patterns Overview

Zimmermann O., *An Architectural Decision Modeling Framework for SOA Design*.
 Dissertation.de, 2009 [Zim09]



- No industry consensus on SOA principles and patterns yet:
 Each author defines his/her own – many terminology mismatches [PESOS09]

SOA Pattern: Enterprise Service Bus (ESB)



- Refinement of well-established **broker** pattern described in [POSA]
 - *Hub-and-spoke* architecture known from many Enterprise Application Integration (EAI) products, providing many-to-many connectivity between loosely coupled parties – the ,B' in ESB
 - Plus explicit, *machine-readable service interface contracts* – the ,S' in ESB
 - Plus *business alignment* and high-end *Quality of Service (QoS)* – the ,E' in ESB
- Key **capabilities** defined in [Keen] and “Enterprise Integration Patterns” [Hohpe]:



- Multiple *transport layers and message exchange patterns*:
 - Synchronous service invocations, e.g., via simple Web protocols (HTTP)
 - Asynchronous messaging (JMS, MQ) – key for loose coupling!
- Mediations providing *content-based routing, message format conversions*, instrumentation for *QoS management* (logging, billing, security controls)
- Declarative, *policy-based* configuration and management

SOA Pattern: Service Composition

- The **business logic layer** of n-tiered enterprise applications can be divided into two sublayers (architectural “refactoring”):
 - Processing steps assigned to roles placed in *process layer* (pattern a.k.a. workflow, business process choreography)
 - Basic computations, validation logic, manipulation of persistent business entities placed in *atomic services layer*
- **Foundations** for process layer execution semantics (workflows):
 - *Business Process Management (BPM)*, Petri nets, Pi-calculus, graph theory
 - One technology option is the *Web Services Business Process Execution Language (WS-BPEL or BPEL4WS)*, standardized by [OASIS]
- **Key issues**:
 - Where to draw the line between the two sublayers?
 - How to interface with the presentation layer?
 - Integration of legacy workflows, e.g., those residing in software packages?



Source: [Hohpe]

SOA Pattern: Service Registry

- SOA incarnation of **naming and directory services** known from CORBA, J2EE, DCE, and other distributed computing technologies

- **Key capabilities:**
 - Build time service publishing and lookup
 - Human user (developer)
 - Tools
 - Runtime registration and lookup of service providers
 - Semantic annotations
 - Matchmaking

- **Known uses:**
 - Web services specification: UDDI
 - IBM WebSphere Service Repository and Registry (WSRR)
 - Most of today's SOA references use custom built repositories tightly integrated into inhouse development processes

Exercise 1: Reflection on Today's Decision Making Practices

1. Which decision hurt your head most on your last project?
2. What did you do to resolve it?
3. Which forces influenced your decision making?
4. Where did you look for help?
5. How did you document your decision?
6. Were your lessons learned specific to your project, or would you say they are applicable to other project situations?
7. If so, how did you share them with the community?

Time for this exercise: 5 minutes

Module 1 Take Aways

- Key concepts applied by software architects in practice:
 - Quality attributes
 - Architectural patterns
 - Architectural decisions(and many others, which are covered elsewhere – viewpoints, methods, etc.)
- Key SOA patterns are:
 - Service Provider-Consumer Contract (known from economics and OOP)
 - Enterprise Service Bus (known from messaging and EAI)
 - Service Composition (known from CBD and workflow technology)
 - Service Registry (known from distributed computing and networking)
- Making the right decisions to satisfy the quality attributes is complex:
 - Based on complex thought processes
 - Using contextual information and decision criteria from many directions

SATURN 2010 Tutorial

Module 2: Decision Modeling with Reuse



Agenda for Module 2

- Motivation
- Existing work
 - Tyree/Akerman template
 - Kruchten/Lago/van Vliet ontology
 - Decision capturing template in IBM Unified Method Framework (UMF)
- SOA Decision Modeling (SOAD) vision and framework overview
 - Key SOAD concepts
 - Content examples
 - Tool support

What are Architectural Decisions? Why Bother?

- “The design decisions that are costly to change” (Grady Booch, 2009)
- Our definition [WICSA08, EelesCripps09]:
 - *“Architectural decisions capture key design issues and the rationale behind chosen solutions. They are conscious design decisions concerning a software system as a whole, or one or more of its core components, with impact on non-functional characteristics such as software quality attributes.”*
- From IBM UMF work product description ART 0513 (previous name: ARC 100):
 - “The purpose of the Architectural Decisions work product is to:
 - Provide a single place to find important architectural decisions
 - Make explicit the rationale and justification of architectural decisions
 - Preserve design integrity in the provision of functionality and its allocation to system components
 - Ensure that the architecture is extensible and can support an evolving system
 - Provide a reference of documented decisions for new people who join the project
 - Avoid unnecessary reconsideration of the same issues”

Decision Capturing Templates and Ontologies

- IBM Global Services Method a.k.a. Unified Method Framework (UMF)
 - Work product (artifact) “ART 0513/ARC 100 Architectural Decisions”
- Tyree and Akerman created a decision capturing template having worked with an ARC 100 work product for e-business architectures
 - Issue, decision, status, group, assumptions, constraints, positions, argument
 - Implications, related decisions, related artifacts, related principles, notes

J. Tyree, A. Akerman, *Architecture Decisions: Demystifying Architecture*, IEEE Software, vol. 22, no. 2, 2005

- Kruchten et al. define a decision ontology
 - State model, decision types, decision dependencies

Kruchten, P., Lago, P., van Vliet, H.: *Building up and Reasoning about Architectural Knowledge*. In: Hofmeister, C., Cmkovic, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 39-47. Springer, Heidelberg (2006)

Retrospective decision capturing (documentation purposes)!

The current state of the art and the practice makes it difficult to reuse issues decisions during architecture design.

Problem 1:
 Template designed to capture decisions made
Unwelcome documentation task, out of design context

Subject Area	Topic
Architectural Decision	AD ID
Decision	"We decided for the Broker pattern [POSA] ..."
Issue or Problem	... to integrate front end and backend...
Assumptions	
Motivation	
Alternatives	
Justification	...because we gained positive experience with it on many similar projects."
Implications	
Derived Requirements	
Related Decisions	

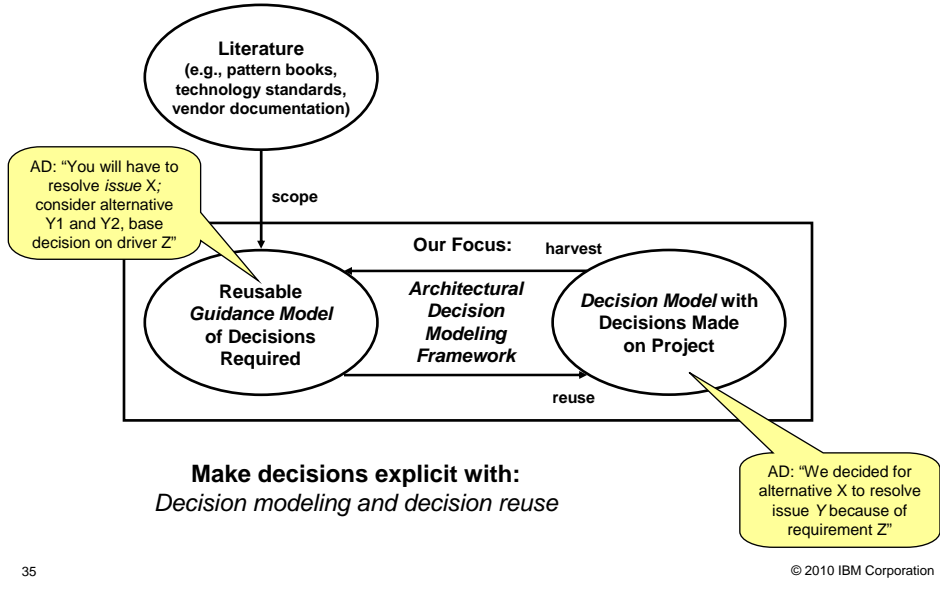
Problem 2:
 Text documents used
Manual work, limited scalability

Source: **UMF Work Product Description for Architectural Decisions**, © IBM, 1998

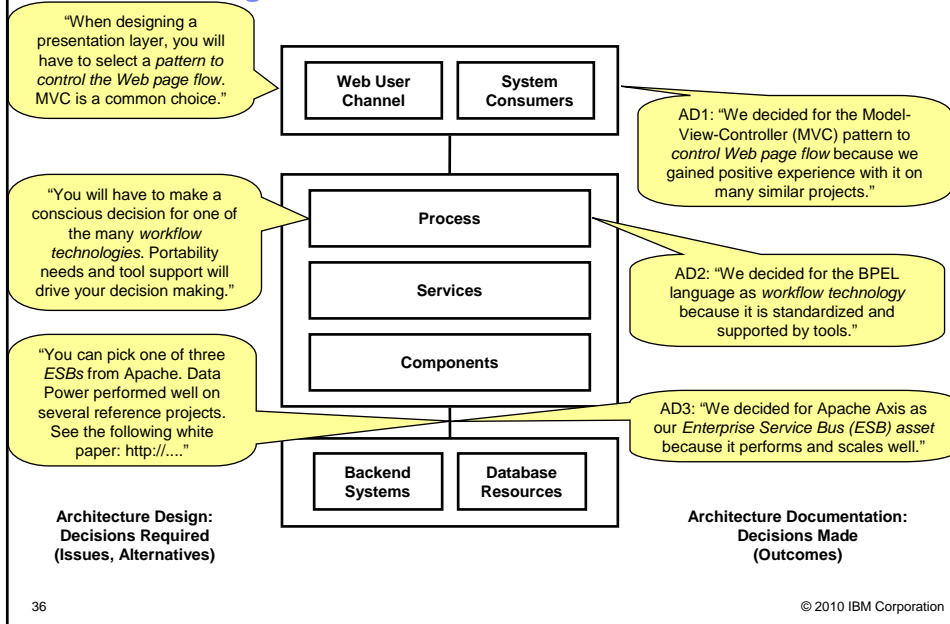
Consequence: Little reuse – no catalog of recurring design issues exists today (SOA or other style)

Design work always conducted from scratch and in isolation – time consuming, error prone

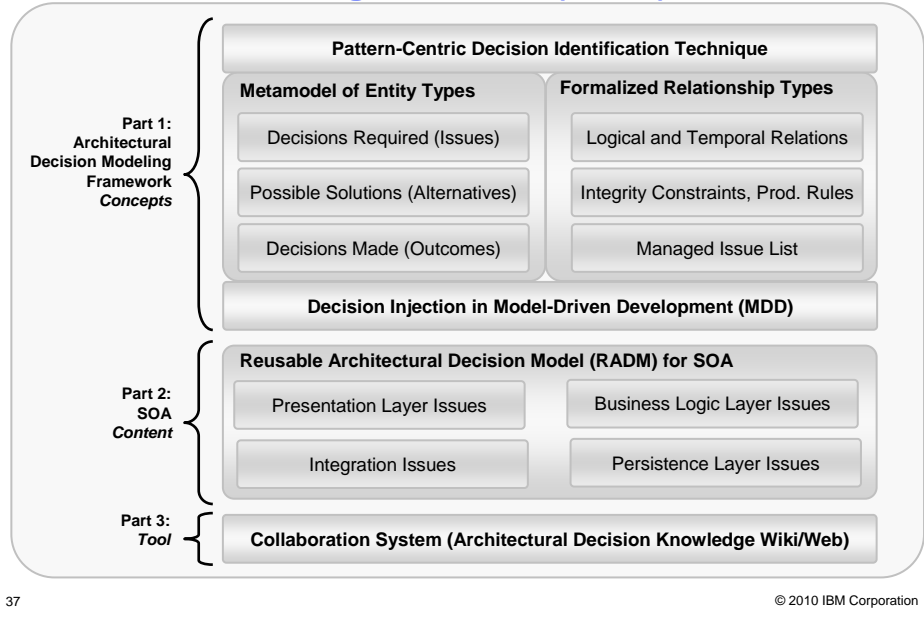
Vision: Architectural Decision Modeling Framework



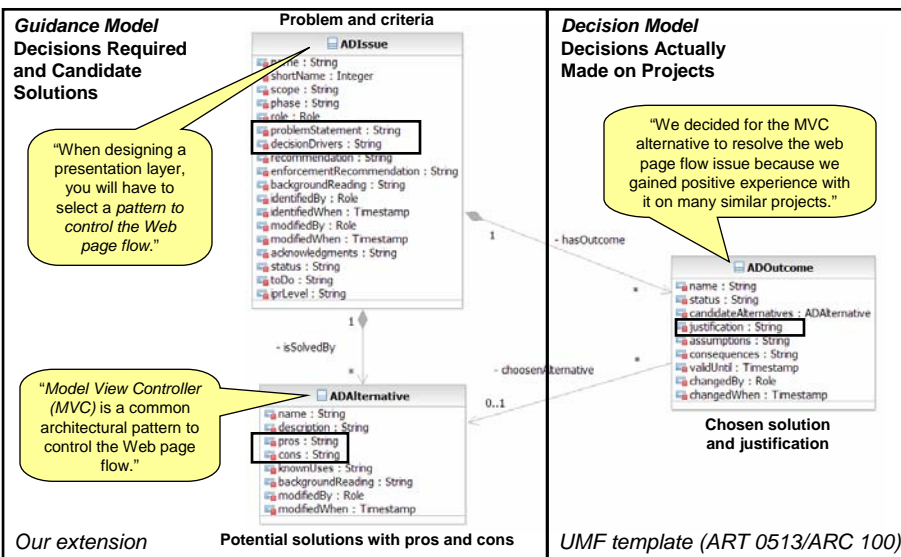
Extended Usage Scenario for Architectural Decisions



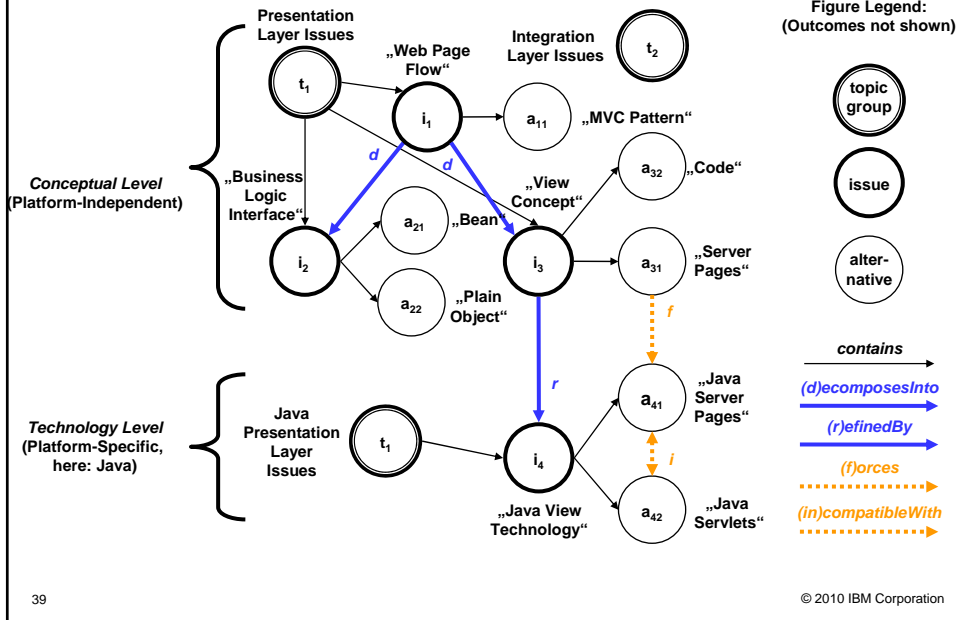
SOA Decision Modeling Framework (SOAD) Solution



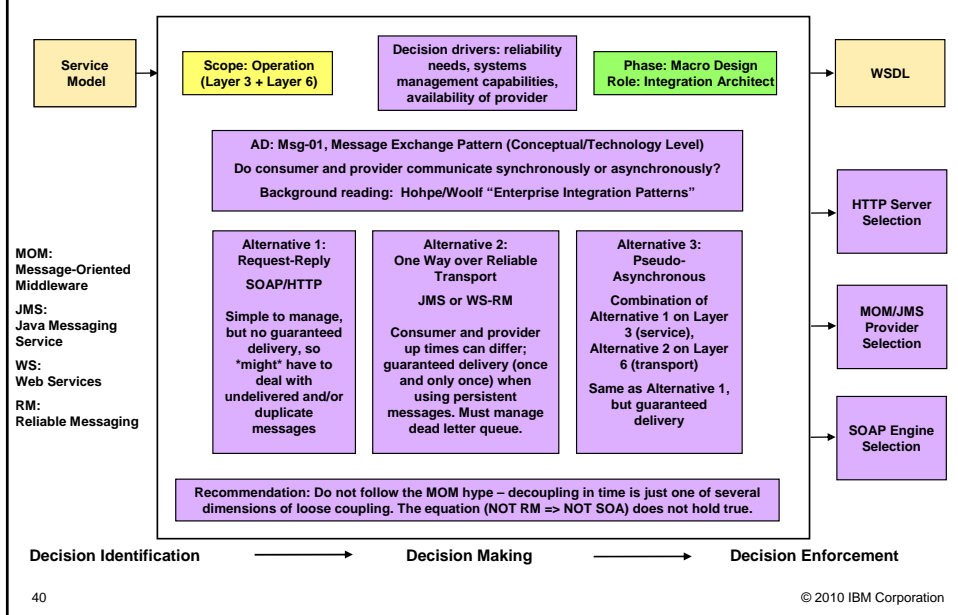
Entity Types and Associations in UML Metamodel



Relationship Types and Levels



Part 2 (Content): Sample SOA Issue and Alternatives

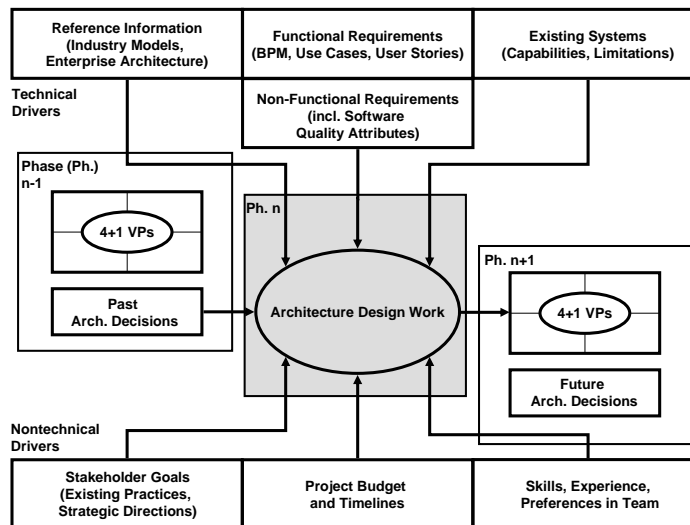


500+ SOA Design Issues in RADM Guidance Model Patterns

- In and Out Parameter Granularity
- Transaction Management
- Session Management
- SOAP Engine Selection
- Transaction Attributes

Refinement Stage	Architectural Decision (AD)	AD Alternatives (subset)
Stage 1 (RADM-E): Executive Decisions (EDs)	ED-1: Platform Language Tool Preferences ED-2: EAS Architectural Principles, Paradigms, Patterns, Processes	Technologies such as J2EE, .NET, LAMP/ Ruby on Rails Selection of reference architecture, layering approach, design method, relevant literature (many alternatives)
Stage 2 (RADM-C): Conceptual Decisions (CDs), dealing with selection of architectural patterns)	CD-1: Integration Style CD-2: Broker Pattern CD-3: Message Exchange Pattern CD-4: In and Out Message Parameter Granularity CD-5: Service Composition Paradigm CD-6: Presentation Layer Paradigm CD-7: Conversational State CD-8: Transaction Management CD-9: Message Exchange Technologies (e.g. SOAP, The-Wire Protocol [2])	Front end centric vs. integration centric vs. process centric vs. database centric [13] BROKER [1] (RPC or messaging) vs. direct client server connectivity Synchronous request-response, POLL, OBJECT, RESULT CALLBACK, SYNC WITH SERVER, or FIRE AND FORGET [22] MESSAGE ORIENTED MESSAGES (MOM) elements vs. flat strings Business process engine (following MICRO-MICROFLOW [12] or no separation of flows) vs. custom code, PROCESS-BASED INTEGRATION ARCHITECTURE [12] or not KCB centric vs. thin client vs. best of both worlds CLIENT SESSION STATE [8] vs. SERVER SESSION STATE [8] vs. none SYSTEM TRANSACTIONS vs. BUSINESS TRANSACTIONS [8] Web services vs. plain Message Oriented Middleware (MOM) vs. plain Remote Procedure Call (RPC) vs. CORBA vs. proprietary (e.g. CICS Transaction Gateway) vs. custom protocols Commercial ESB product vs. custom integration layer INVOCATION, CLIENT PROXY, MESSAGE LEE, INVOCATION INTERCEPTOR, INVOCATION CONTEXT, use of PROTOCOL, PLUG-IN [22] or not WS* and SOAP vs. REST and POX/JSON vs. plain TCP/IP and custom strings vs. other HTTP vs. messaging
Stage 3 (RADM-I): Technology Decisions (TDs), dealing with selection of design patterns, technologies	TD-1: Messaging Patterns TD-2: Message Exchange Style and Format TD-3: Transport Protocol Binding TD-4: Service Provider Type and Application Programming Interface (API) TD-5: Service Provider Component Container Technology TD-6: Business Process Language TD-7: Process-based Integration Architectural Design TD-8: Presentation Layer Technology TD-9: Presentation Process Layer Coordination TD-10: Presentation Layer Organization TD-11: Presentation Process Layer Protocol TD-12: Process Layer Interface Granularity TD-13: Presentation Process Layer Coordination TD-14: Presentation Process Layer Protocol TD-15: Activation Strategy Patterns TD-16: Resource Management Patterns TD-17: Session Management TD-18: Compensation Scheme TD-19: ESB Product TD-20: SOAP Message Exchange Engine TD-21: Service Provider Container TD-22: Service Provider Sourcing TD-23: Business Process Engine Vendor TD-24: Presentation Layer Application Server TD-25: Platform Specific Transaction Attribute	Enterprise Java Bean (EJB) vs. plain Java object vs. other provider in other programming language: JAX-RPC vs. JAX-WS/JAX-BS vs. proprietary Service Component Architecture (SCA) vs. Java 2 Enterprise Edition (J2EE) vs. Spring vs. Common Object Request Broker Architecture (CORBA) vs. .NET vs. other Business Process Execution Language (BPEL), other EAS: EAS-IMP/INSPEC/IE or other; deployment and structure of MICROFLOW ENGINES and MICROFLOW ENGINES, CONFIGURABLE ADAPTERS and REPOSITORIES, etc. (see [12]) Plain HTML (thin client) vs. portal (thin client) vs. Web 2.0 RIA vs. Eclipse RCP APPLICATION CONTROLLER vs. PAGE FRONT CONTROLLER [11] vs. BUSINESS TASK LIST [18] Batch vs. conversational Pull (presentation leading) vs. push (process leading) STATIC INSTANCES, PER-REQUEST INSTANCES, or CLIENT-DEPENDENT INSTANCES (all from [22]) LEXICONS, POOLING, LATE ACQUISITION, and PASSIVATION (all from [22]) Client (e.g. full state in cookie) vs. presentation layer (HTP session) vs. process layer (BPEL correlation) vs. backend (database) BPEL vs. vendor specific engines vs. custom logic E.g. IBM WebSphere ESB, Progress Sonic ESB, Mule Apache Axis, Codehaus XFire, vendor engines such as IBM WebSphere engine, WSIF and Apache SOAP J2EE application server with(out) EJB support, SCA container such as WebSphereProcess Server Make or buy; adapt or refactor existing asset E.g. IBM WebSphere Process Server, Oracle BPEL Process Manager, open source (Activiti/BPEL) E.g. various server engines and portal servers, application web engines E.g. various SCA quadders and EJB attributes

From Issues and Alternatives to Outcomes – Driver Types



Valid Outcome Justifications... and Counter Examples

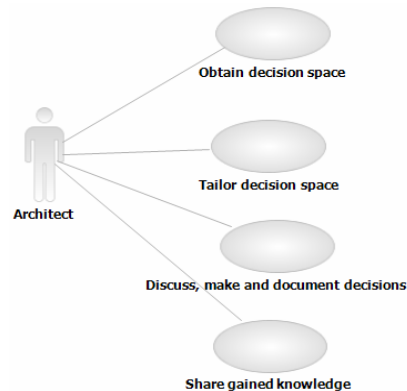
- Good:
 - Direct link to (non-)functional requirements, quality attributes in particular
 - Positive experience on previous project
 - Existing skills, license agreements
- Bad:
 - Market momentum (technology or vendor push)
 - Only one alternative known/considered
 - Keep CVs of team members current

Adobe Acrobat
7.0 Document

- More examples given in this IBM developer works article:
<http://www.ibm.com/developerworks/architecture/library/ar-knowwiki1/>

Part 3: Architectural Decisions Knowledge Tools Project (Technology Incubator, IBM Rational and IBM Research)

- Regulatory compliance
 - E.g., maturity models
- Collaboration
 - In geographically distributed teams
- Reuse
 - Of already gained knowledge
- Other required features:
 - Import and export
 - Searching and filtering
 - Dependency management
 - Report generation



Guidance Modeling Tool (Emerging)

The screenshot shows the 'Guidance model editor' window. The title bar includes 'Issue editor'. The main window is titled 'Guidance Model: SoadMaster16'. Below the title bar, there is a description: 'This is version 162 of the SOAD content master (RADM for SOA) which is part of GTS SOAI RA (February 23, 2009)'. The interface is split into two main panes. The left pane, 'Model Structure', shows a tree view of decision categories: BusinessExecutiveLevel, ConceptualLevel, and OperationalLevel. The right pane, 'Details', contains fields for 'Short name' (Ser-03), 'Name' (OperationToPortTypeGrouping), and 'Problem statement'. The 'Problem statement' field contains text about port type granularity and WSDL 1.1 port type interfaces. Below this is a 'Motivation' section with a reference to 'IMPACT 2010 session 1839' and a 'Decision drivers' section with a paragraph of text.

Decision Modeling Tool (Emerging)

The screenshot shows the 'Decision model editor' window. The title bar includes 'Issue editor'. The main window is titled 'Decision Model: SoadMaster16'. Below the title bar, there is a description: 'This is the decision model for the case study from the telecommunications industry (order management)'. The interface is split into two main panes. The left pane, 'Model Structure', shows a list of decision points (Pows and Prd) with their default values. The right pane, 'Details', shows the 'Outcome: External Relocation Process'. It includes an 'Issue' field with 'ProcessLifetimePatt', a 'Status' of 'DECIDED', and a 'Valid until' date of '3/18/2010'. Below this is a 'Name' field with 'External Relocation Process', a 'Chosen Alternative' dropdown with 'Macroflow, interruptible', and a 'Justification' field with text about a 24-hour process duration. There are also sections for 'Assumptions' (Process model is stable) and 'Consequences' (Need to define business transaction boundaries).

Module 2 Exercise

- Classify the following model elements according to the UML metamodel classes (issue/alternative/outcome) that were introduced on page 38:
 - “Which pattern should we pick to structure the integration layer?”
 - “We investigated the Broker pattern from POSA and ESB in Krafzig’s SOA book”
 - “Should we use a binary or a textual representation of service requests?”
 - “Integration technology”
 - “SOAP over “HTTP” and any “WS-*” specification (WSDL, WS-Security, WS-Policy)”
 - “XML” as message exchange format
 - “RESTful integration”
 - “Eclipse Rich Client Platform (RCP) or Java-based Web application”?
 - “Apache Mule” as ESB implementation and provider
 - “HSQLDB from Source Forge.net”
 - “We could use the IBM WebSphere application server, seems to have market traction”
 - “We decided to use Oracle Business Process Manager, licenses already purchased (enterprise license agreement in place)”

- Time for this exercise: 5-10 minutes

Module 2 Take Aways

- Several decision capturing templates exist in industry and academia: their main focus is on recording *decisions made* (after-the-fact)
- SOA Decision Modeling (SOAD) advances the state of the art of decision modeling with native support for knowledge reuse and embedding *decisions required* into the architecting process
- Issues, (potential) alternatives, and (actual) outcomes are three aspects of a decision with different reuse characteristics; therefore, these aspects are modeled as separate entities in SOAD
- A 500-node SOA decision model has been compiled, which proves that decisions recur and can be modeled in a reusable fashion
- Prototypical tool support for the modeling concepts is available

SATURN 2010 Tutorial

Module 3: A Guidance Model created with SOAD – Reusable Architectural Decision Model (RADM) for SOA and Cloud Design



© 2010 IBM Corporation

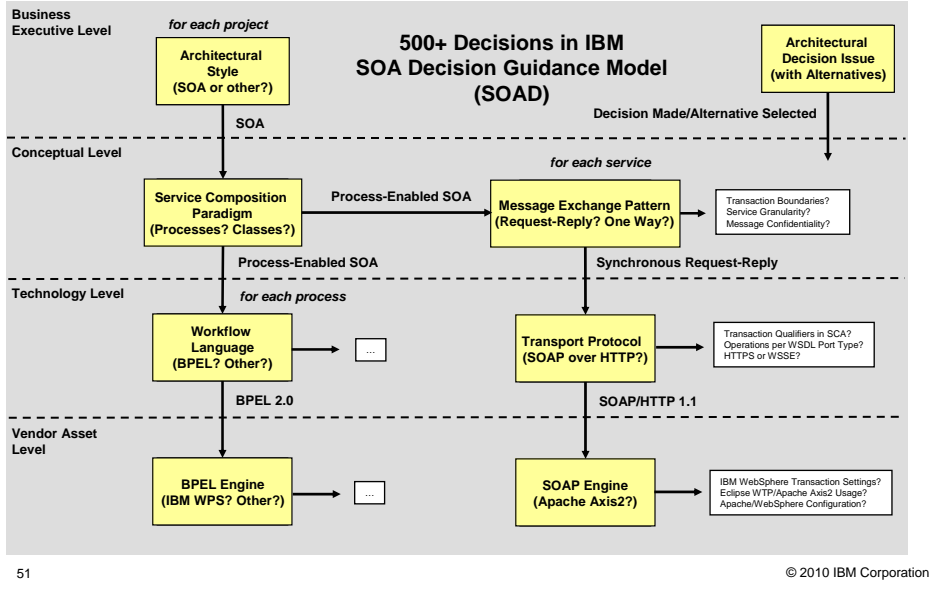
Agenda for Module 3

- RADM for SOA guidance model – organizing principles and overview:
 - Refinement levels
 - Architectural layers

- Top 10 SOA decisions (issues/alternatives):
 - Architectural style and reference architecture
 - Integration
 - Service composition
 - Granularity

- Cloud design decisions (emerging)

From AD Documentation to Active Method Guidance

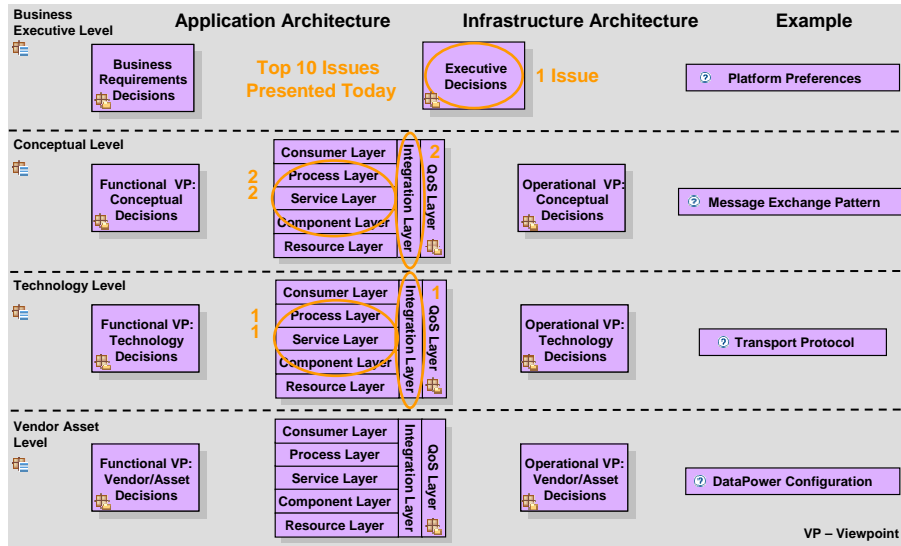


Excerpt from RADM for SOA Guidance Model [Zim09]

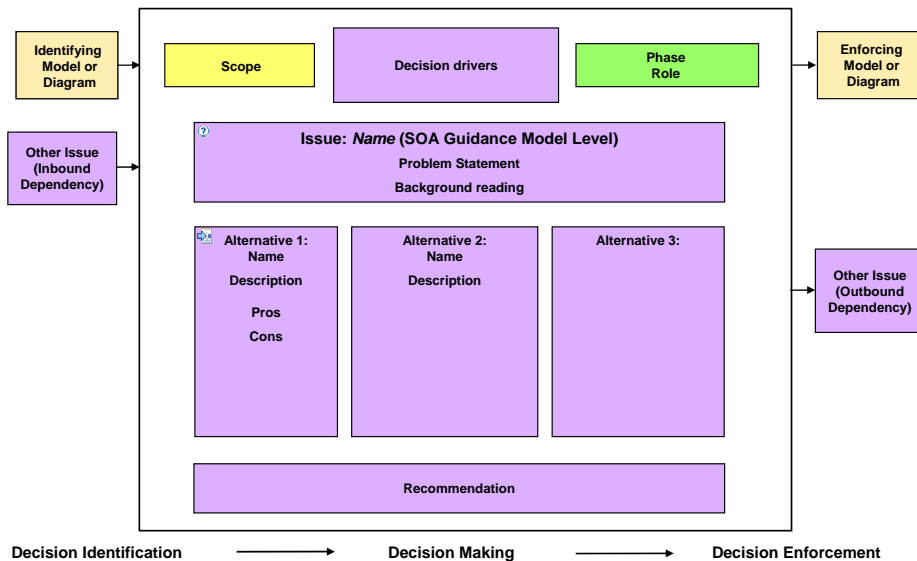
Identification Rule (IR) and Level	Layer	Issue (Decision Required)
IR1: (Technical) executive decisions, requirements analysis decisions	n/a	Architectural Style Layering Language and Platform Preferences Tooling Directions Functional Requirements Notation BPM Notation
IR2 and IR3: Pattern Selection Decisions (PSDs), Pattern Adoption Decisions (PADs)	Atomic service layer Integration layer Service composition layer	In Message Granularity Out Message Granularity Operation-to-Service Grouping Message Exchange Pattern Invocation Transactionality Pattern Service Provider Transactionality (ST) Integration Paradigm Communications Transactionality (CT) Service Composition Paradigm Process Lifetime Session Management Resource Protection Strategy Process Lifetime Process Activity Transactionality (PAT)
IR4 and IR5: Technology Selection Decisions (TSDs), Technology Profiling Decisions (TPDs)	Atomic service layer Integration layer Service composition layer	Transport Protocol Binding Message Exchange Format SOAP Communication Style Web Services Transactionality Web Services API Java Service Provider Type XML Schema (XSD) Constructs Integration Technology Transport QoS (SCA Qualifiers) Workflow Language BPEL Version Compensation Technology
IR6 and IR7: Vendor Asset Selection Decisions (ASDs), Vendor Asset Configuration Decisions (ACDs)	Atomic service layer Integration layer Service composition layer	SOAP Engine ESB Product ESB Topology (IBM DataPower Configuration) BPEL Engine Invoke Activity Transactionality

52 © 2010 IBM Corporation

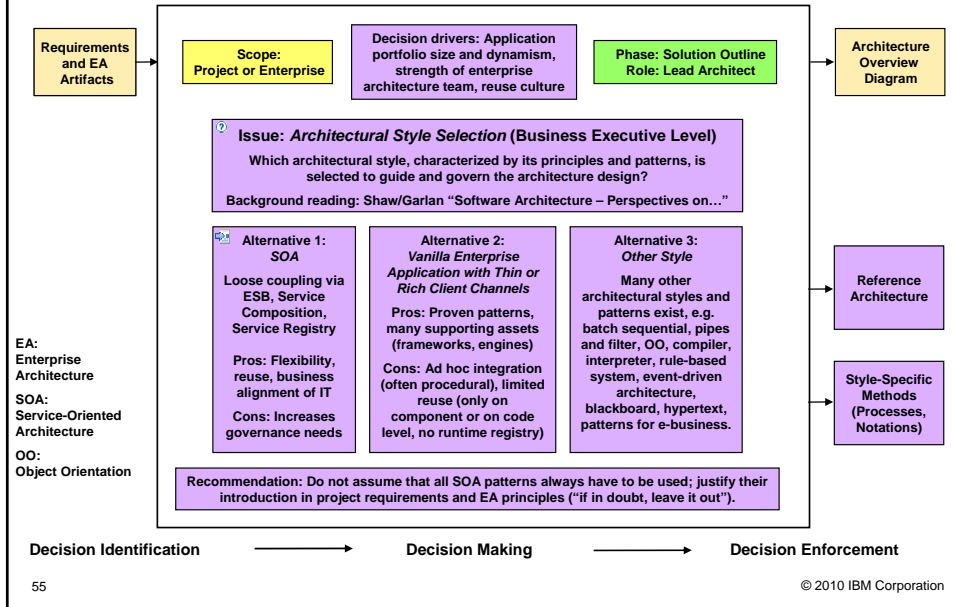
SOA Design Issues Organized by Levels and Layers



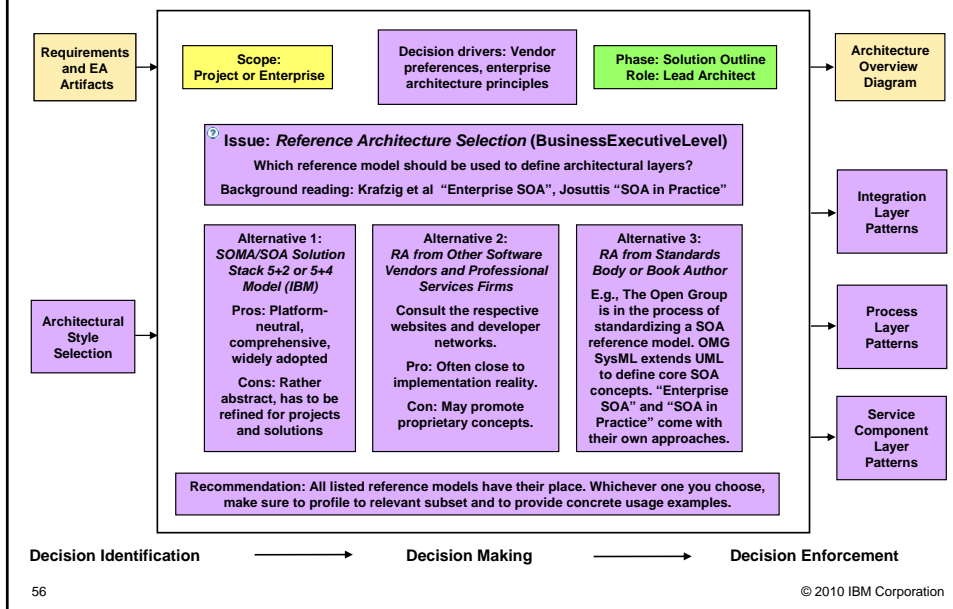
Template Used to Present Issues and Alternatives



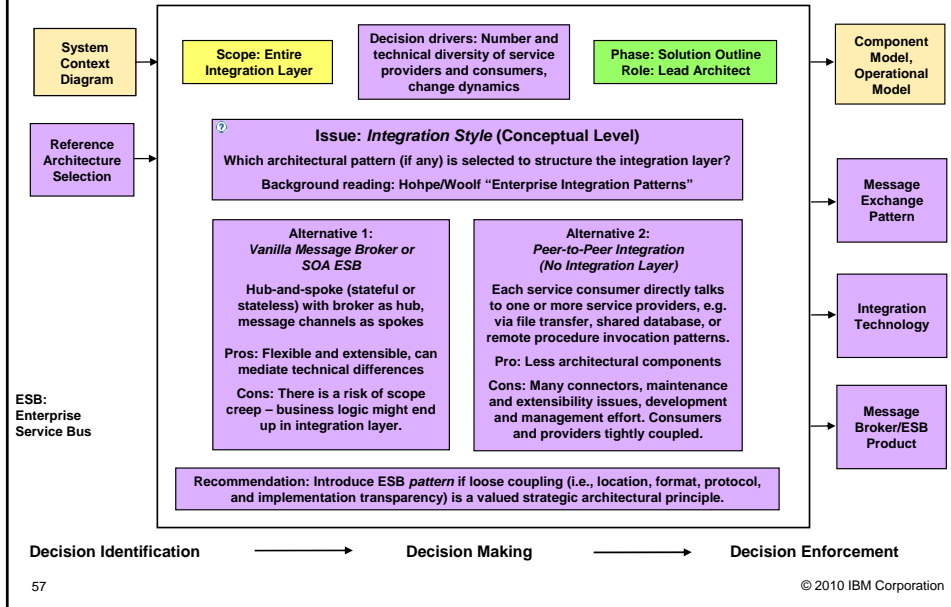
A Decision Already Made – Architectural Style Selection



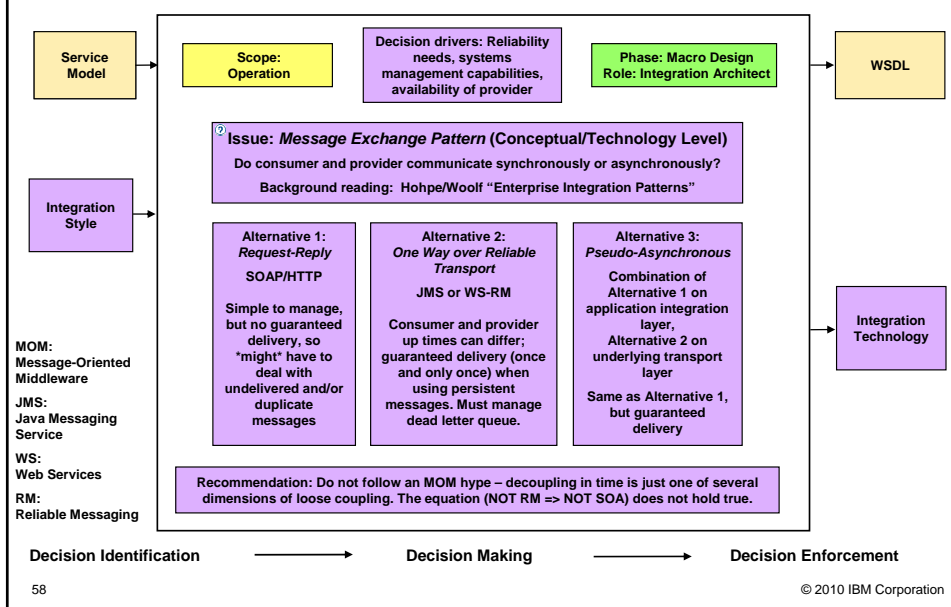
AD Issue #10 – Reference Architecture (RA)



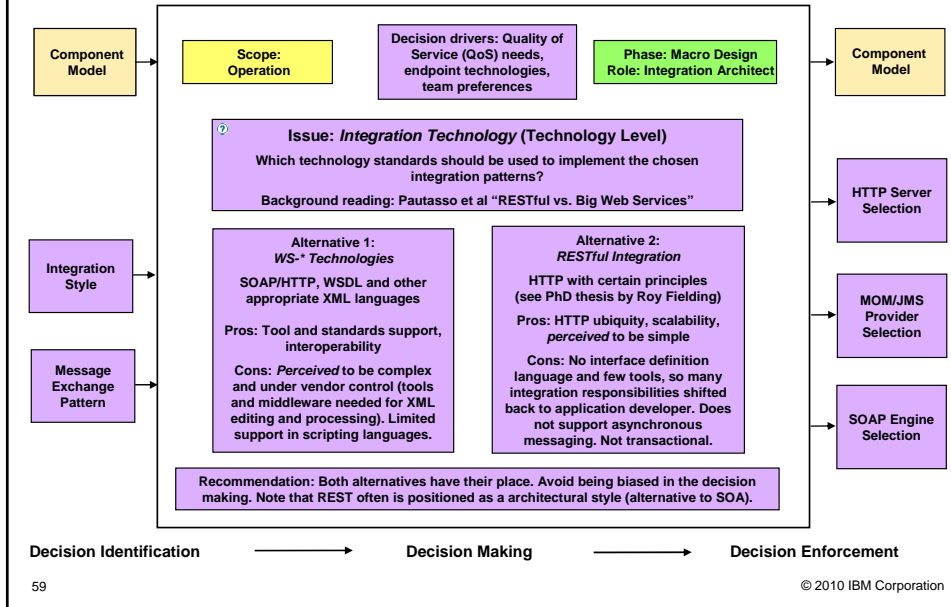
AD Issue #9 – Overall Integration Layer Design Pattern



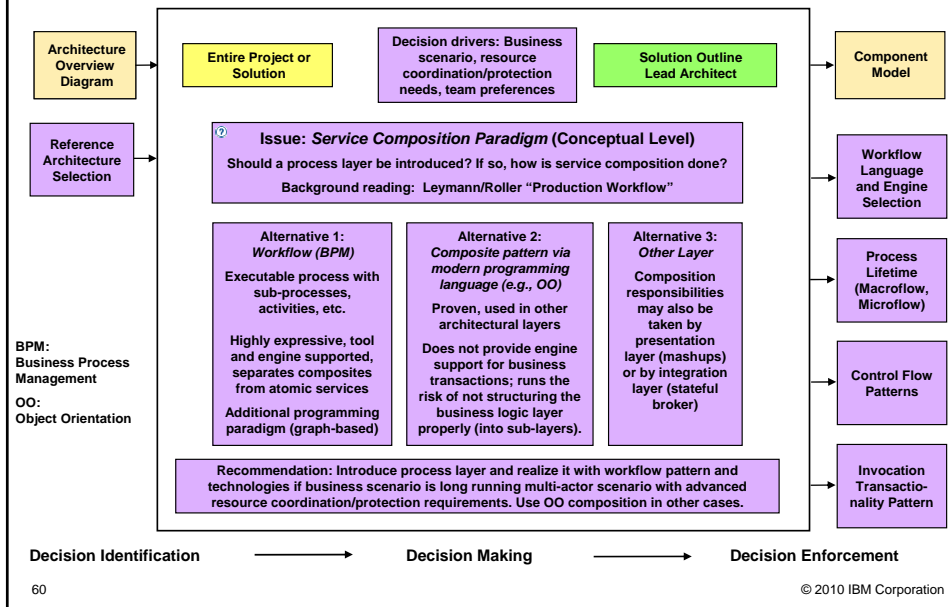
AD Issue #8 – A Detailed Integration Layer Design Issue



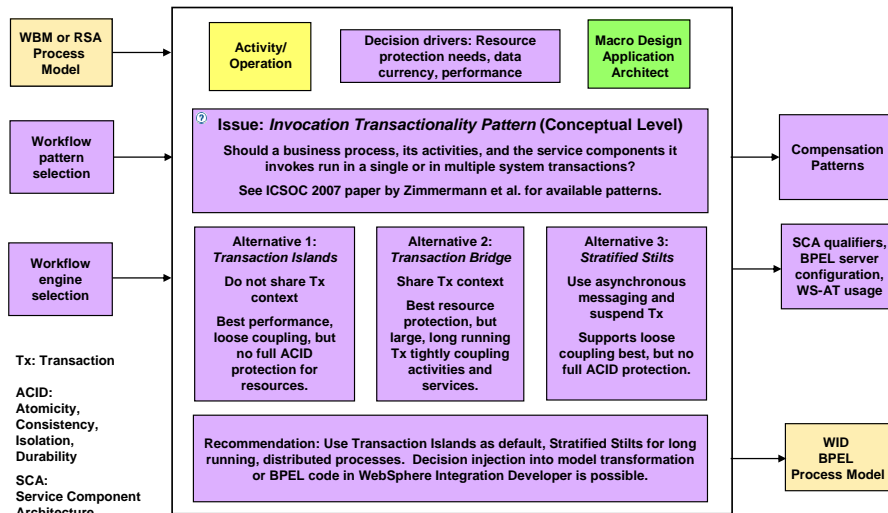
AD Issue #7 – Integration Layer Technologies



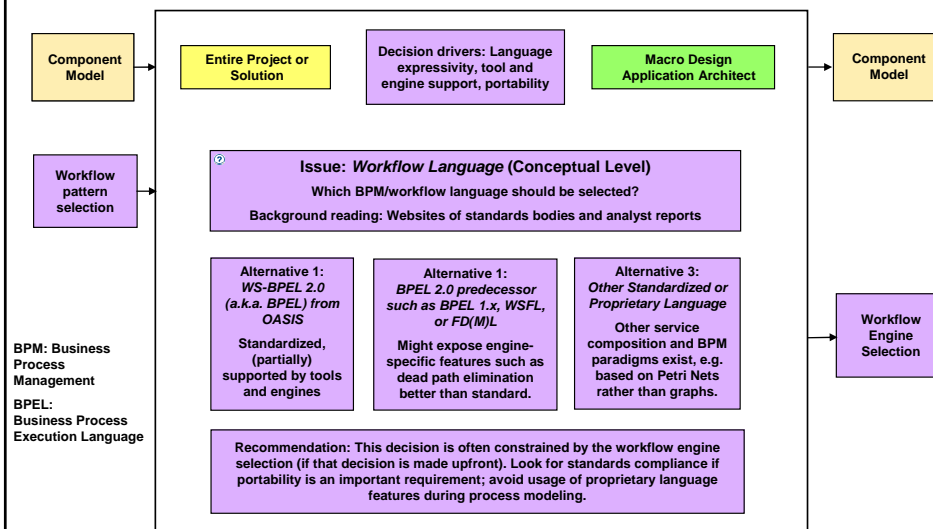
AD Issue #6 – Overall Process Layer Design Pattern



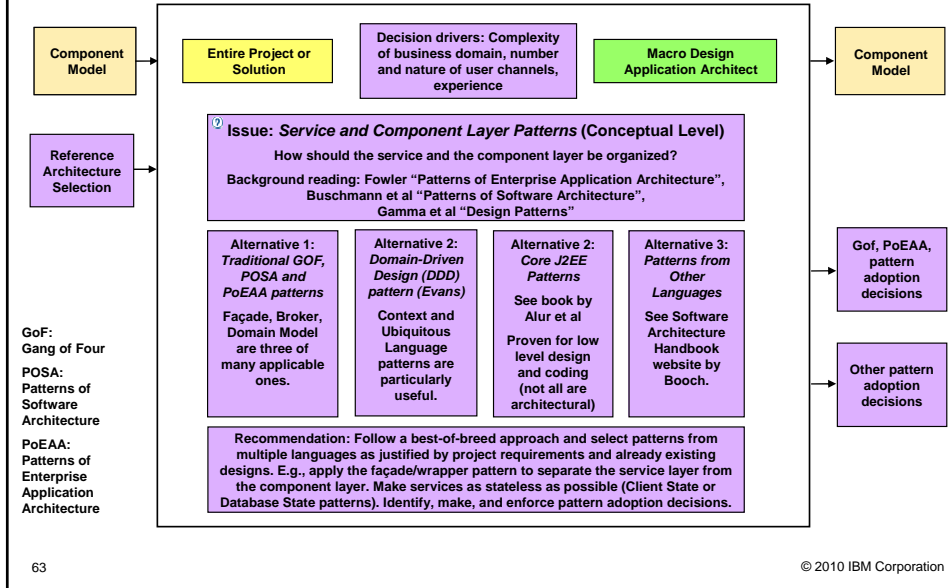
AD Issue #5 – Transaction Management Topic



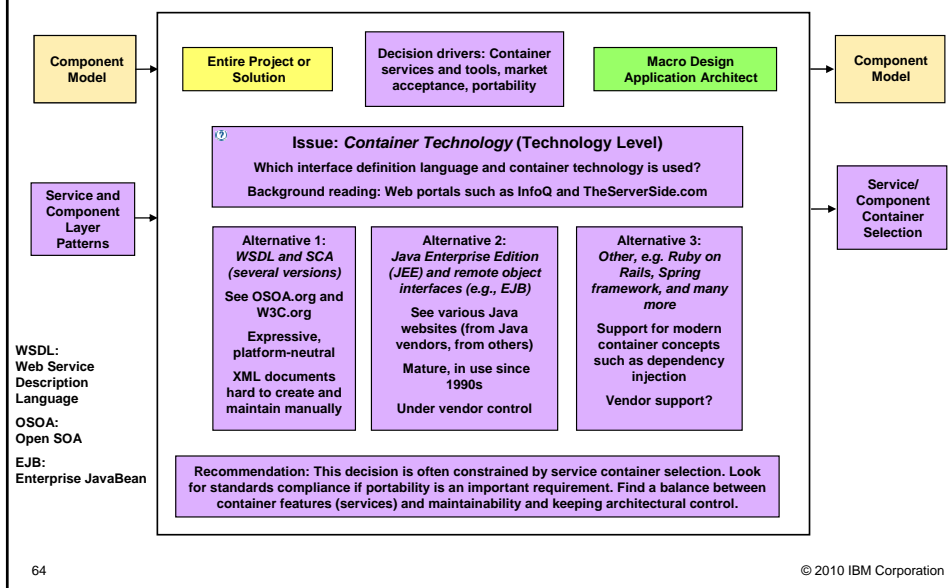
AD Issue #4 – Process Layer Technologies



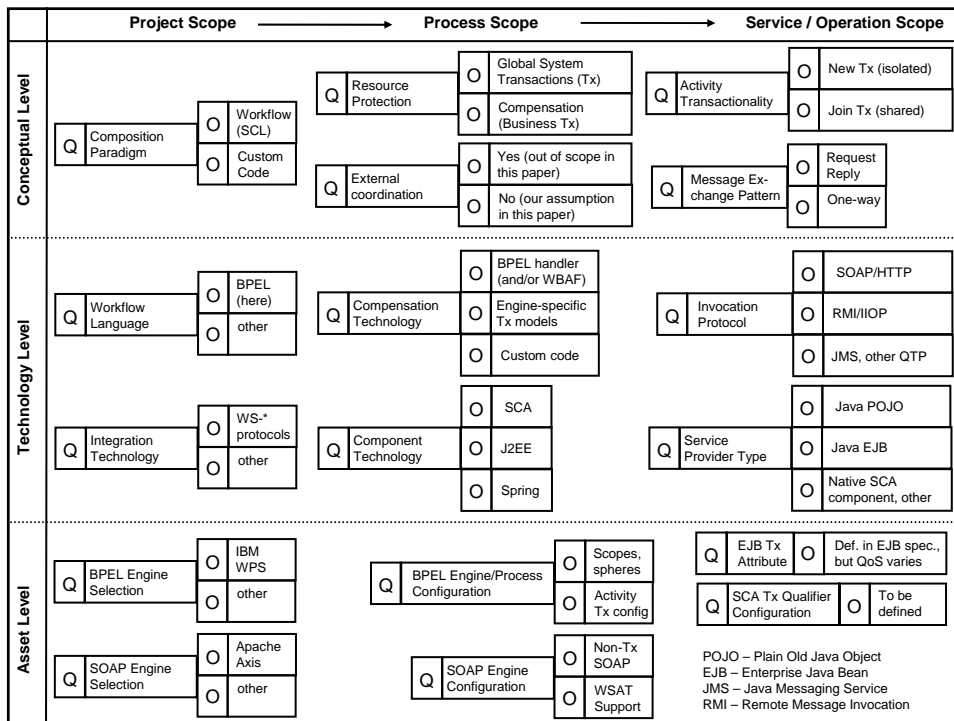
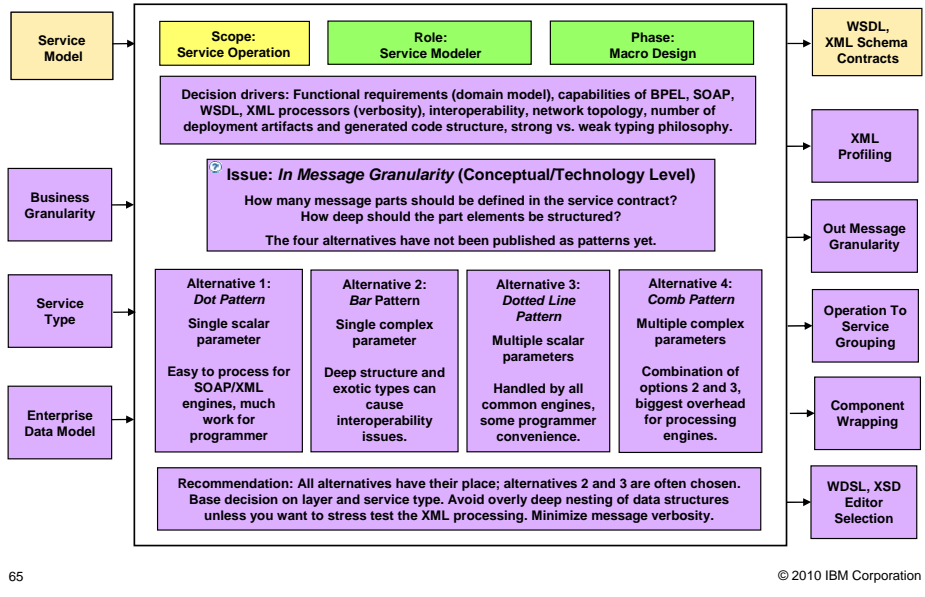
AD Issue #3 – Service/Component Layer Design Patterns



AD Issue #2 – Service/Component Layer Technologies



AD Issue #1 – Addressing Service Granularity Topic



	Project Scope	Process Scope	Service / Operation Scope
Conceptual Level	Q Integration Style <ul style="list-style-type: none"> <input type="radio"/> Shared Database <input type="radio"/> File transfer <input type="radio"/> RPC <input type="radio"/> Messaging 	Q Message Exchange Pattern <ul style="list-style-type: none"> <input type="radio"/> Request Reply <input type="radio"/> One-way 	Q Operation Design <ul style="list-style-type: none"> <input type="radio"/> DIY Q Parameter Design <ul style="list-style-type: none"> <input type="radio"/> DIY
Technology Level	Q RPC Technology <ul style="list-style-type: none"> <input type="radio"/> CORBA <input type="radio"/> J2EE <input type="radio"/> RESTful Web <input type="radio"/> WS-* Q Messaging Technology <ul style="list-style-type: none"> <input type="radio"/> Native MOM <input type="radio"/> WS-RM Q Transactions <ul style="list-style-type: none"> <input type="radio"/> DIY <input type="radio"/> WS-AT, WS-BAF 	Q Service Identification <ul style="list-style-type: none"> <input type="radio"/> URI <input type="radio"/> WS-Address. Q Transport Protocol <ul style="list-style-type: none"> <input type="radio"/> Many options Q Security <ul style="list-style-type: none"> <input type="radio"/> HTTPS <input type="radio"/> WSSE <input type="radio"/> other 	Q Payload Format <ul style="list-style-type: none"> <input type="radio"/> SOAP Q Interface Contract <ul style="list-style-type: none"> <input type="radio"/> WSDL Q Provider Impl. (Payload Proc.) <ul style="list-style-type: none"> <input type="radio"/> JAX-RPC stub <input type="radio"/> Native HTTP/XML
Asset Level	Q WSDL Editor <ul style="list-style-type: none"> <input type="radio"/> Many options 	Q HTTP Server, SOAP Engine <ul style="list-style-type: none"> <input type="radio"/> Apache <input type="radio"/> other 	Q SOAP Client (Proxy) <ul style="list-style-type: none"> <input type="radio"/> JAX-RPC <input type="radio"/> other

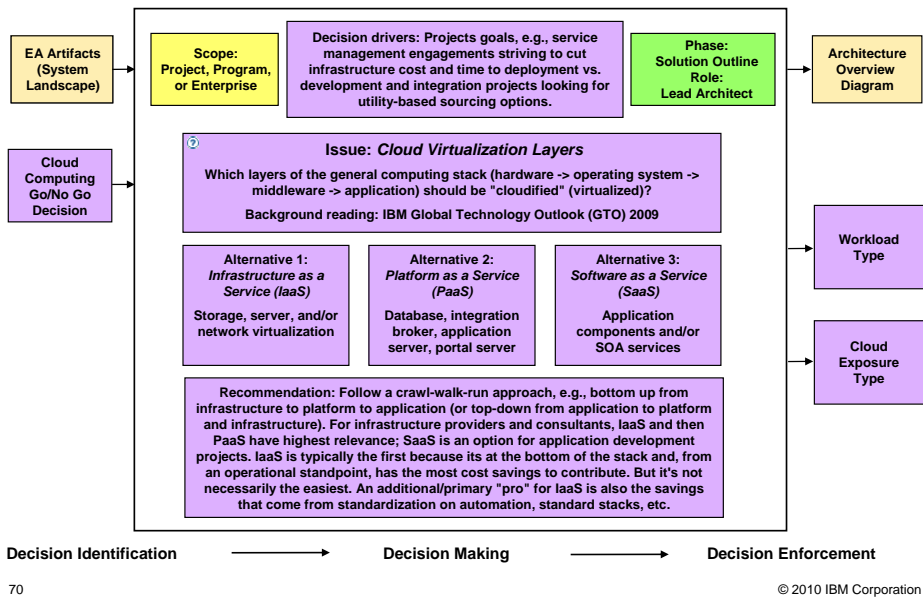
	Project Scope	Domain / Web Application Scope	Resource / Verb Scope
Conceptual Level	Q Integration Style <ul style="list-style-type: none"> <input type="radio"/> Shared Database <input type="radio"/> File transfer <input type="radio"/> RPC <input type="radio"/> Messaging 	Q Message Exchange Pattern <ul style="list-style-type: none"> <input type="radio"/> Request Reply <input type="radio"/> One-way 	Q Resource IF Design <ul style="list-style-type: none"> <input type="radio"/> DIY
Technology Level	Q RPC Technology <ul style="list-style-type: none"> <input type="radio"/> CORBA <input type="radio"/> J2EE <input type="radio"/> RESTful Web <input type="radio"/> WS-* Q Transactions <ul style="list-style-type: none"> <input type="radio"/> DIY Q Reliability <ul style="list-style-type: none"> <input type="radio"/> DIY 	Q Resource Identification <ul style="list-style-type: none"> <input type="radio"/> URI Q REST style (principles) <ul style="list-style-type: none"> <input type="radio"/> High REST <input type="radio"/> Low REST Q Transport Protocol <ul style="list-style-type: none"> <input type="radio"/> HTTP Q Interface Contract <ul style="list-style-type: none"> <input type="radio"/> HTTP verbs Q Security <ul style="list-style-type: none"> <input type="radio"/> HTTPS <input type="radio"/> none 	Q URI design principles <ul style="list-style-type: none"> <input type="radio"/> „nice“ URIs <input type="radio"/> arbitrary Q Payload Format <ul style="list-style-type: none"> <input type="radio"/> POX <input type="radio"/> JSON <input type="radio"/> RDF (plus POX?) <input type="radio"/> Other (YAML, ATOM, MIME) Q Provider Impl. (Payload Proc.) <ul style="list-style-type: none"> <input type="radio"/> DIY (Servlets)
Asset Level	Q WADL Editor <ul style="list-style-type: none"> <input type="radio"/> Future option 	Q HTTP Server, (Servlet Engine) <ul style="list-style-type: none"> <input type="radio"/> Apache, (Tomcat) <input type="radio"/> other 	Q HTTP Client <ul style="list-style-type: none"> <input type="radio"/> Java.net (J2SE) <input type="radio"/> Browser (JS)

SOA Decisions Summary and Outlook

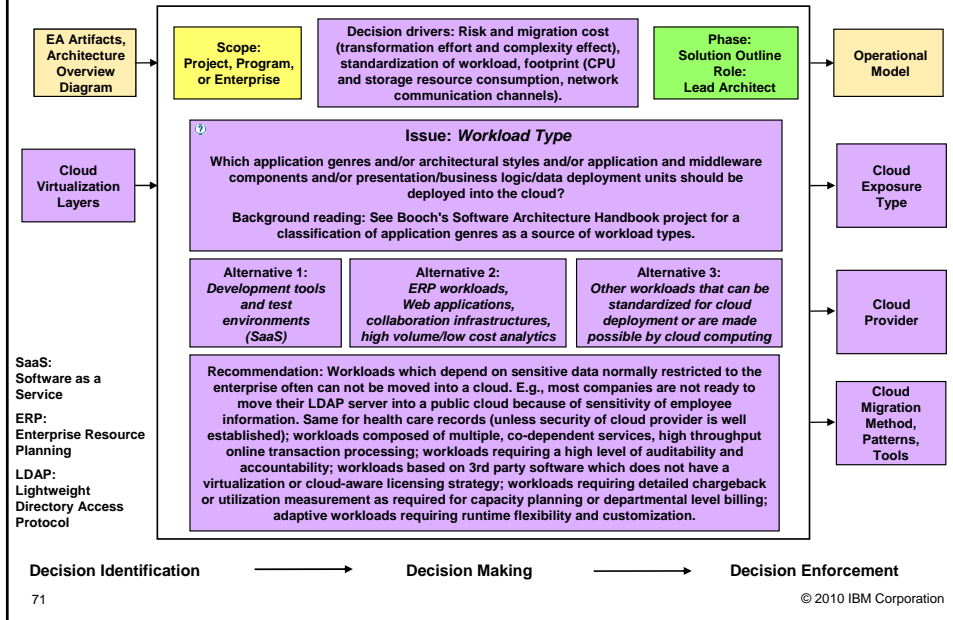
- We covered the following recurring SOA design issues:
 - Two executive decisions, *architectural style* and *reference architecture selection*
 - Three integration layer decisions, including one on *message exchange patterns*
 - Three process layer decisions, including one on *invocation transactionality patterns*
 - Three service/component layer decisions, including one on *service granularity*

- We did not cover (but the existing full RADM for SOA guidance model does)
 - More pattern selection and adoption decisions on process, service component, and integration layers, e.g., *session state management*, *error handling*
 - *Security* patterns and technologies
 - *Systems management* patterns and technologies
 - *Presentation layer* patterns and technologies
 - *Persistence layer* patterns and technologies
 - *Organization and governance* patterns and technologies

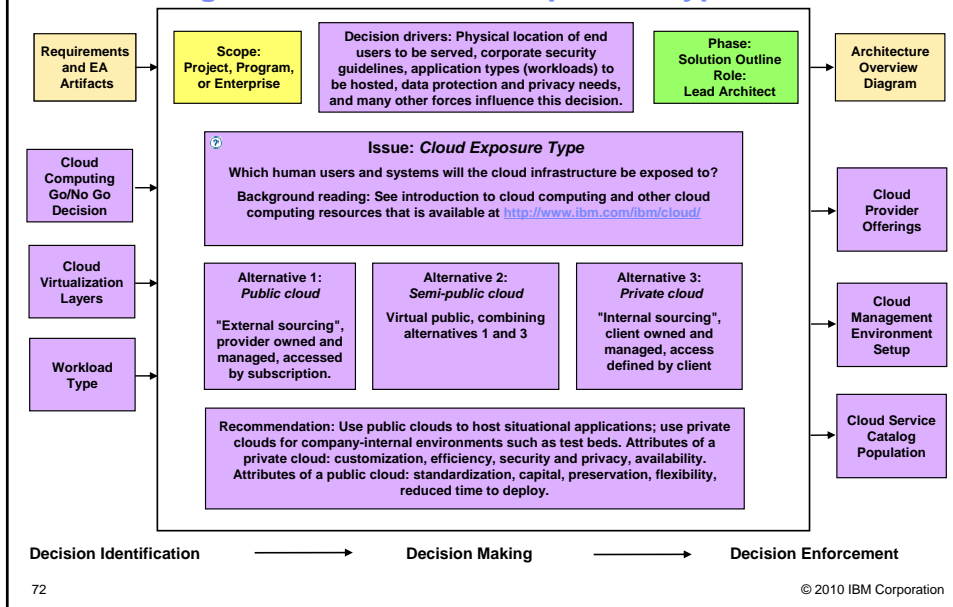
Cloud Design Issue #1 – Cloud Virtualization Layers



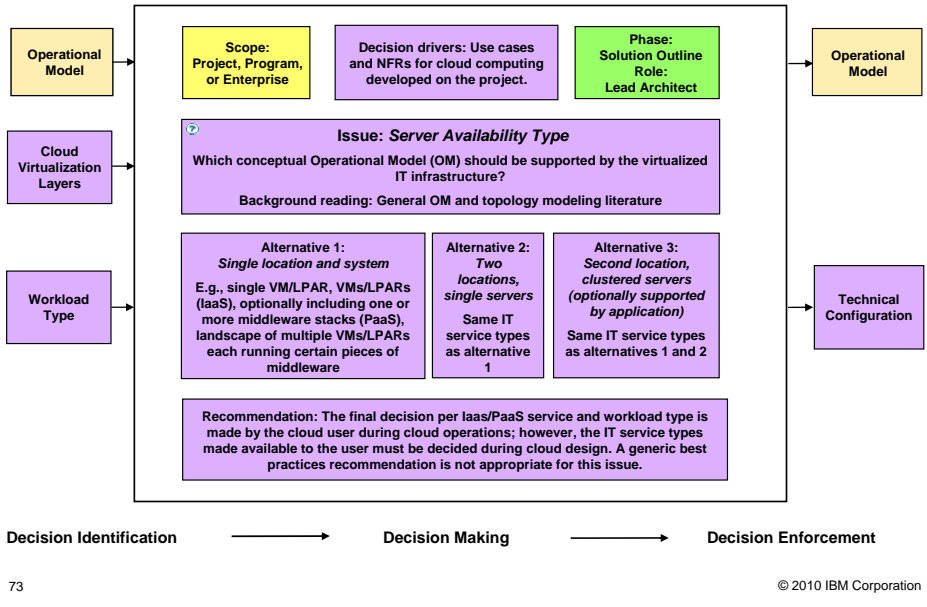
Cloud Design Issue #2 – Workload Type



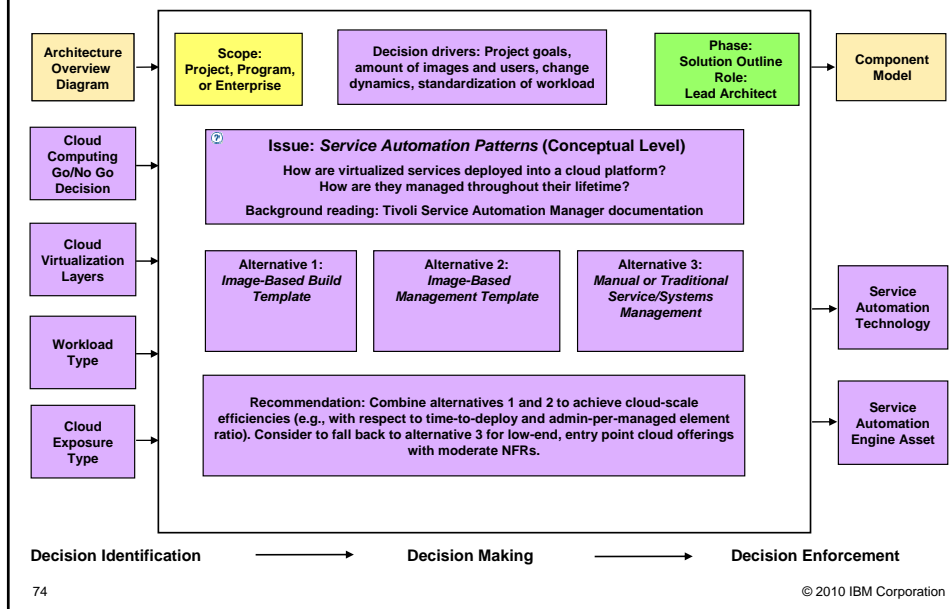
Cloud Design Issue #3 – Cloud Exposure Type



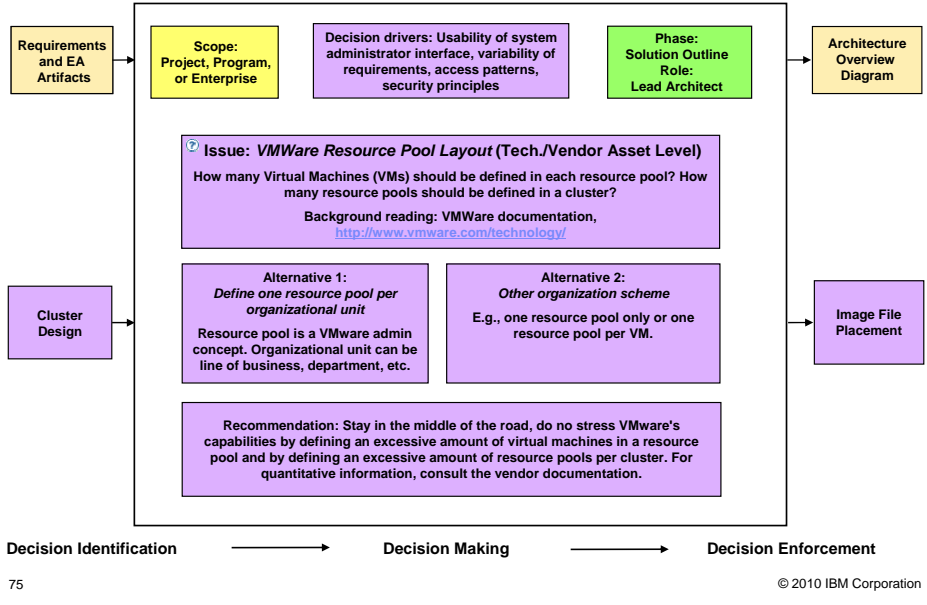
Cloud Design Issue #4 – Server Availability Class



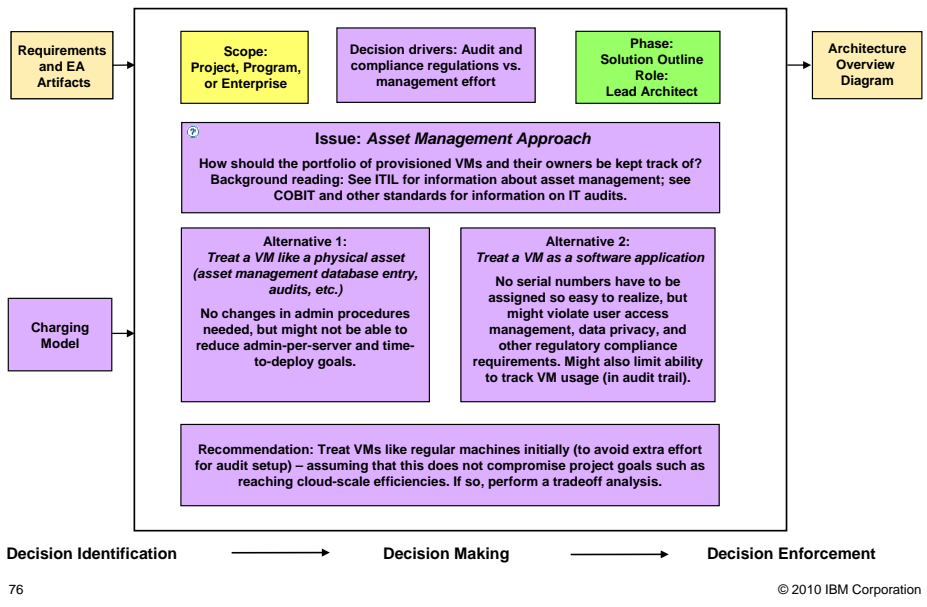
Cloud Design Issue #5 – Service Automation Patterns



Cloud Design Issue #6 – VMWare Resource Pool Layout



Cloud Design Issue #7 – Asset Management Approach



Other Topics/Issues Recurring in Cloud Computing

- Charging model
 - Via registration (flat rate)
 - Per provisioned resource
 - By resource consumption
 - None
- Datamodel
 - Relational
 - Own, e.g. based on key-value pairs (partially relaxing ACID properties of relational)
- Service catalog
- Integration with already existing IT infrastructures
- Cloud security
- Business support system design and API
- Operational support system design and API

Exercise 3a): Multiple Choice Test on RADM Organization

- Which refinement level do the following issues and alternatives reside on?
 - “Which pattern should we pick to structure the integration layer?”
 - “We investigated the Broker pattern from POSA and ESB in Krafzig’s SOA book”
 - “Should we use a binary or a textual representation of service requests?”
 - “Integration technology”
 - “SOAP over “HTTP” and any “WS-*” specification (WSDL, WS-Security, WS-Policy)”
 - “XML” as message exchange format
 - “RESTful integration”
 - “Eclipse Rich Client Platform (RCP) or Java-based Web application?”
 - “Apache Mule” as ESB implementation and provider
 - “HSQLDB from Source Forge.net”
 - “We could use the IBM WebSphere application server, seems to have market traction”
 - “We decided to use Oracle Business Process Manager, licenses already purchased (enterprise license agreement in place)”
- Time for this exercise: 5-10 mins

Exercise 3b) More Decision Types, Identification Rules, Meta Issues

- Study the Figures and Tables in Chapter 5 of “An Architectural Decision Modeling Framework for Service-Oriented Architecture Design” [Zim09]
 - Ph. D. thesis [Zim09] is available for download at:
http://elib.uni-stuttgart.de/opus/frontdoor.php?source_opus=5228&la=de

Exercise 3 c): Levels, Layers, and Content Creation (Optional/Advanced)

1. Download the WWW 2008 paper “RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision”
 - Available at <http://soadecisions.org/soad.htm#www>
 2. Review Tables 1, 2, and 3 in that paper:
 - Which levels are covered?
 - How many architectural decisions are discussed?
 - How do the decisions map to ADTopics defined in the Reusable ADM for SOA (RADM for SOA) that we introduced in this module?
 3. (optional) Create your own project in Architectural Knowledge Decision Web and capture the ADs and ADAalternatives discussed in the paper.
- Time for this exercise: 30 minutes (steps 1 and 2)

Module 3 Take Aways

- RADM for SOA guidance model structure follows proven modeling concepts
 - Refinement levels, resembling platform-independent/platform-specific model types in MDA/MDD
 - Architectural layers, taken from a reference architecture for a domain or an architectural style
- Key SOA decisions include:
 - Executive decisions:
 - Architectural style, reference architecture, layering scheme, programming language
 - Conceptual decisions
 - SOA pattern selection, subpattern selection and adoption (ESB, composition, registry)
 - Integration style
 - Granularity of in and out messages
 - Transactionality
 - Technology decisions
 - WS-* or RESTful integration
 - Security technologies
 - Vendor asset decisions
 - Application server topology, standalone or clustered
 - Database vendor and RDBMS configuration
- Emerging cloud design issues pertain to virtualization layers, workload and exposure type

SATURN 2010 Tutorial

Module 4: Case Study



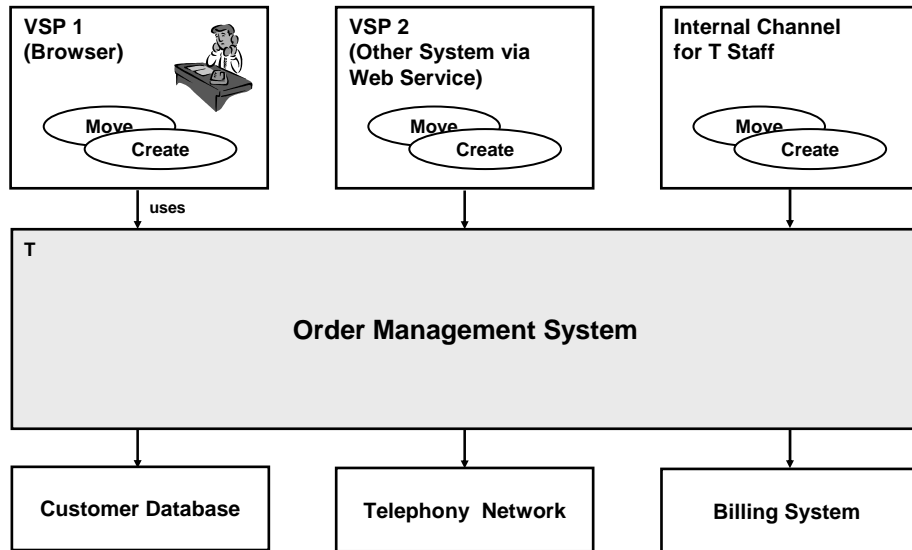
Agenda for Module 4

- “T” Case Study
 - Context and business problems
 - System context
 - Functional requirements
 - Non-functional requirements
- Architecture overview diagram and patterns already selected
- The assignment
 - Task 1 to task 5
- Additional project information

“T” Case Study: Context & Business Problems

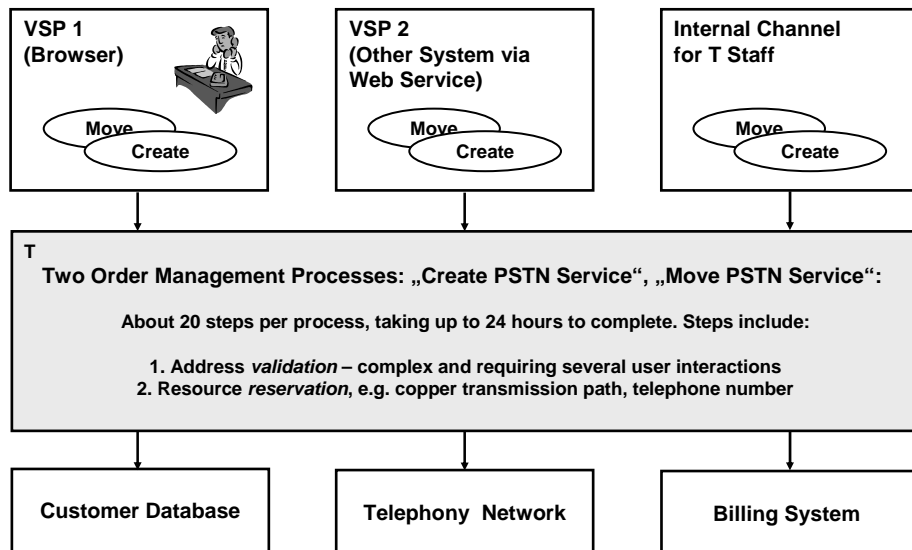
- **Wholesale subsidiary** of large telecommunications company T (former monopolist, deregulated):
 - Provides wire line and wireless telephony services to retailer subsidiary of T and to 150 other companies, called Virtual Service Providers (VSPs)
 - One physical telephony network, owned and operated by T
- **Strategic imperative** of T:
 - Drive down cost of operations by interacting with VSPs efficiently
- Response: Single, partially automated **order management system**
 - VSPs are expected to use the order management processes of T to connect, configure, or disconnect telephony services for their end users
 - Multiple channels required, including the World-Wide Web (Internet)

System Context Diagram



Functional Requirements

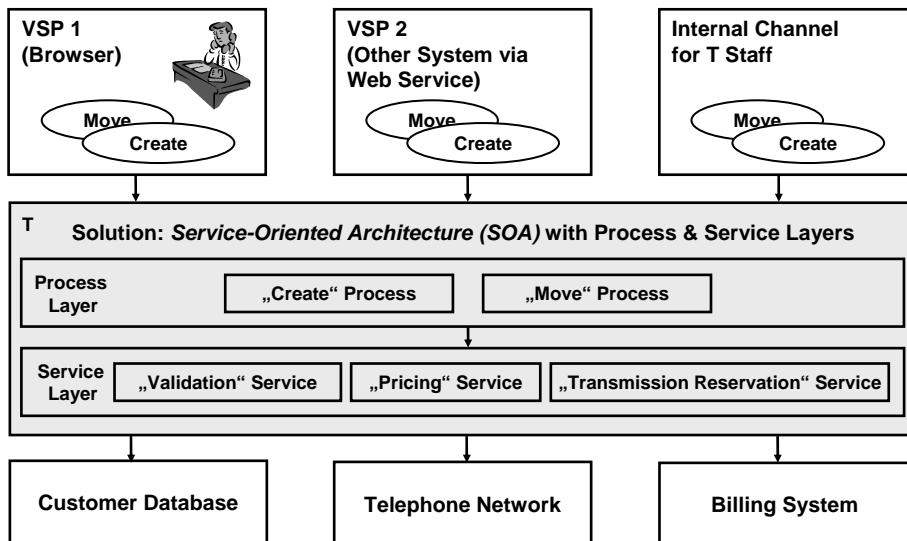
PSTN – Public Switched Telephone Network
VSP – Virtual Service Provider



Important Non-Functional Requirements (NFRs)

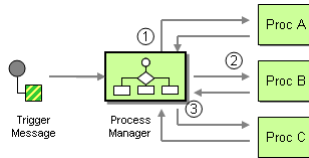
1. The software system supporting the two order management processes must be accessible both over a private industry-sponsored network and the Internet. The VSPs and the backend systems to be integrated (e.g., billing system) change over time.
2. Business volumes are approximately 20,000 “Create PSTN service” requests and 15,000 “Move PSTN service” requests per month.
 - Given up to 20 steps per process, and a peak hour load of 30% above average, this equates to a peak load of about 4,550 steps executed per hour (based on core business hours of ten hours per day, 20 days per month)
3. Initially, process instances must be able to persist from first step to last for three hours; however, this time will be extended to up to 24 hours in the future.
4. VSPs are spread across a number of time zones, operating 23 hours per day and seven days per week.
5. Average response time targets vary by process step, typically 3-5 seconds; 95% of the user interactions need to complete execution in 5-8 seconds.
6. Domain-specific architecture design challenges include: 1. Address validation completeness and timeliness, 2. Releasing reserved resources (copper transmission path, telephone number) when a process instance fails or customer walks away.
7. Communication patterns and protocols must support multiple platforms.

Architecture Overview Diagram (First Design Iteration)



Process Manager Pattern

“How do we route a message through multiple processing steps when the required steps may not be known at design-time and may not be sequential?”



“Use a central processing unit, a *Process Manager*, to maintain the state of the sequence and determine the next processing step based on intermediate results.”
 [EIP]

Process manager takes care of process instance creation and deletion, control flow routing, error handling in timeout situations, etc.

Pattern source: Hohpe/Woolf, “Enterprise Integration Patterns” (EIP),
<http://www.eaipatterns.com/ProcessManager.html>

Architectural Decision (AD) to Use Process Mgr. Pattern

Subject Area	Process layer design	Topic	BPM
AD	Service Composition Paradigm	AD ID	2
Decision	We decided for the Process Manager pattern from the [EIP] book.		
Issue or Problem	How to control the “Create/Move PSTN service” processing?		
Assumptions	Process model and requirements NFR 1 to NFR 7 are valid and stable		
Motivation	Process instance management is a mandatory requirement in our order management scenarios; it has high architectural significance.		
Alternatives	Object-oriented programming, proprietary EAI tools		
Justification	The problem statement and the sketched solution of the pattern fit the requirements of the project (multiple processing steps, not sequential, correctness QA and timeout management NFR).		
Implications	Need to educate the team on process management and SOA		
Derived Requirements	Security requirements: to be able to correlate process instances and VSPs, users have to be identified in the order management system		
Related Decisions	Next, we have to decide on an integration style supporting the link between processes and services. We also have to select implementation technologies and providers for all selected patterns.		

Summary of Status Quo in Case Study

- T has compiled a short list of vendors (professional services) and hosted an information day today. During the information day, T has presented:
 - Business strategy and system context
 - Order management requirements (functional and non-functional)
 - Design decisions already made including a partially specified business process and an architecture overview diagram (first design iteration)

- You learned about:
 - Architectural decision making based on quality attributes and patterns

The Assignment

- T issues a “Request for Proposal (RFP)” and expects a response within a week:
 - You are taking software architect role in one of the professional services companies on the vendor short list

- Your RFP response should contain:
 1. Documentation of at least one architectural decision made (mandatory)
 2. Second iteration of architecture overview diagram (mandatory)
 3. Answers to technical questions about methods, languages, and tools (optional)
 4. Value and benefits argumentation (mandatory)
 5. Cost estimate (optional – handle with care)

Task 1: Review Existing Architecture Documentation

- As presented so far (in this module and in tutorial handouts)

Task 2: Refine Architecture with Integration Patterns

- Decide for an *integration style* to connect the activities in the two order management processes (“Create PSTN Service”, “Move PSTN Service”) to the atomic services (“validation” and “transmission reservation”) and justify your decision.

– [File Transfer](#) pattern?

– [Shared Database](#) pattern?

– [Remote Procedure Invocation](#) pattern?

– [Messaging](#) pattern?

- Input
 - System context diagram, architecture overview diagram (iteration 1/2), NFRs
- Output:
 - Documentation of architectural decision and updated architecture overview diagram

Task 3: Method Aspects and Technical Questions (Optional)

- Which requirements engineering method do you propose, and why?
 - Which other ones did you consider, and why did you decide against them?

- Are you content with the introduction of a process manager into the architecture (and why)?
 - Which service composition paradigm to use – workflow, other?
 - Which programming language – Java, BPEL, other?
 - Which workflow or other engine (open source, commercial)?

- Which remote communication protocol would you use to support the chosen integration style, making the services available to the processes? Name the relevant technology standard.

Task 4 and 5

- Task 4 – Value and benefits argumentation (mandatory):
 - What are the benefits of your solution?
 - Why should you get the deal?

- Task 5 – Cost estimate (optional – handle with care):
 - Can you give a first indication how much it will cost to build your solution?
 - How long will it take?

Additional Project Information

- Cultural environment
 - Developed country, multicultural society. High amount of immigration/emigration.
 - Telecommunications company positions itself as a thought leader and early adopter of emerging technology in its geography, ready to operate in risk-reward sharing mode
 - Java skills available, Business Process Execution Language (BPEL) education planned
- Team
 - About 100 people already on board, who worked on previous order management system (in production, but not process-oriented): ~5 project managers, ~5 architects, ~20 business requirement analysts, ~50 developers, ~20 testers
- Existing assets
 - 20 backend systems to be integrated in total, Java adapters available
 - 100.000 lines of legacy code (Java and other), e.g., validation and reservation EJBs
 - Technology platforms to be supported: Microsoft .NET clients, JEE clients and servers, SQL and relational database, proprietary telephony hardware, commercial billing package

Module 4 Take Aways

- Multiple options exist for architecture design issues on all four SOAD levels
 - And typically there is more than one valid solution
- The relations between quality attributes and alternatives/decisions are multi-faceted
 - n:m cardinalities, conflicts between requirements, dependencies between issues
- The SOAD modeling concepts are applicable beyond SOA
 - Any application genre that has been relatively stable for some time qualifies
- A RADM does not intend to put the architect out of the job, but to structure the architecture design work and to facilitate a knowledge exchange
 - Acceleration of orientation in complex design space
 - Best practices knowledge exchange
- Discussion

SATURN 2010 Tutorial

Concluding Thoughts

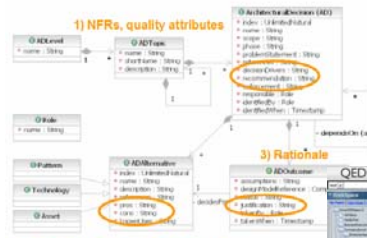


© 2010 IBM Corporation

Summary of What We Have Learned (Take Aways)

- Decision-centric software architecture design process
 - Start from functional requirements and – even more importantly – NFRs, particularly the explicit and tacit quality attributes
 - Find architectural patterns which resolve the forces underneath the NFRs, e.g., in the [POSA] book series and the [PoEAA] and [EIP] books
 - Make conscious pattern selection decisions and follow-on decisions about technologies and products
- Leverage your personal experience, but also reusable assets:
 - Repeatable general software architecting process (roles, phases, artifacts) [EelesCripps09]
 - SOA Decision Modeling (SOAD) concepts [Zim09]
 - RADM for SOA guidance model content

SOAD Concepts, Content, Tool



Part 1: Framework (method/model) to represent architectural decision knowledge and decision outcomes

- Structural representation
- Integrity constraints to ensure consistency of knowledge base and decisions



Part 3: Tool that implements the model to view and work with the content

- Evaluated UML (and others) – different modeling paradigm
- IBM QED Wiki and Mashup Center for alphaWorks releases (2008, 2009)
- Eclipse-based tooling emerging (incubator project)



Part 2: Guidance model content for SOA

- Announced by IBM GTS
- Proven successful in several industry projects

Key Concepts in SOAD

- **Decision models** rather than text templates
 - Three levels of refinement from concepts to technology to assets
 - Population of decision space based on experience from earlier projects
- Decision model as domain-specific guide through the stages, **architecture and design patterns** describe alternatives in detail
 - Architectural patterns are conceptual alternatives
 - Design and technology patterns come in on subsequent stages
- Prescriptive **design method** for domain-specific architecture design
 - Comprehensive, but still comprehensible
 - Works for domains and styles with recurring decisions for which patterns have been mined (new decision alternatives also are a pattern source)
 - For example: enterprise applications and SOA

How to Apply SOAD on Projects

- Use the metamodel to make your architectural knowledge more reusable
 - Available at <http://www.ibm.com/developerworks/architecture/library/ar-knowwiki1/>
- Follow the advice in the RADM for SOA parts that have been published already
 - <http://soadecisions.org/soad.htm> has the latest updates
- Create a RADM for your organization, application genre, and/or architectural style of choice
 - And/or share issues, alternatives, and outcomes with us

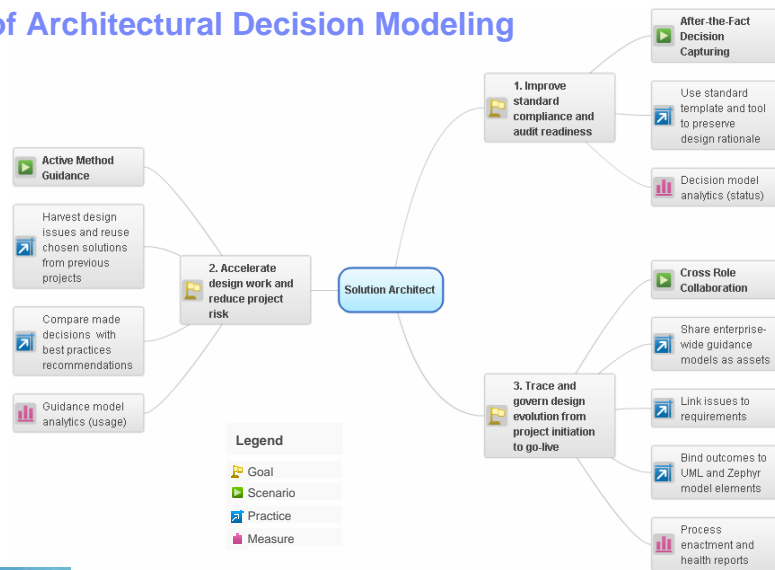
Despite its name, SOAD is applicable to all application genres and architectural styles in which architectural decisions recur!

RADM Creation Process and Decision Modeling Guidance

Zimmermann O. Kopp P, Pappe S., **An SOA Infrastructure Reference Architecture**, in: Software Architecture Knowledge Management, Springer 2009

- Four harvesting steps:
 - *Review, Integrate, Harden, Align (RIHA)*
- Write in a suggestive, positive tone
 - No “police” impression (design authority), no “bossing around” (extra work)
- Do not aim for perfection
 - If in doubt leave it out – premature optimization is the root of all evil
- Find good names
 - Patterns community has a lot of advice to give
- Refactor
 - Split conceptual from technology from vendor asset part

Value of Architectural Decision Modeling



Advanced Usage Scenarios and Future Research

- **Project planning and health checking**
 - Work breakdown structures can be created from the decision model, as open decisions correspond to required activities
 - If there are many, frequent changes, or many questions are still unresolved in late project phases, the project is likely to be troubled
- **Decision models as input to software configuration planning**
 - Product selection and operational modeling decisions define which software licenses are required, and on which hardware nodes the required software has to be installed
- **Enterprise architecture instrument**
 - Company-wide RADM, possibly a tailored RADM for SOA instance, consisting of a subset of decisions and alternatives to give freedom of choice to individual project teams without sacrificing overall architectural integrity

Additional Open Research Problems in This Field

- **Visualization** of decision space
- **Process alignment**
 - Decisions on agile projects
 - Ordering of decision space (notion of a *managed* issue list in [Zim09])
 - Workflow concepts
- **Access control and collaboration**
- **Metamodel standardization**
- **Tool interoperability**
- **Comparing solutions**
- **Metrics**
- **Fuzzy decision modeling**

M. Nowak, C. Pautasso, O. Zimmermann, **Architectural Decision Modeling with Reuse: Challenges and Opportunities**, accepted to the 5th ICSE Workshop on SHaring and REusing Architectural Knowledge ([SHARK 2010](#)), May 2010

References: Software Architecture, Service Engineering

- [EelesCripps09], P. Eeles, P. Cripps, [The Software Architecting Process](#), Addison Wesley, 2009
 - Provides a phase and role terminology that can be leveraged in a RADM
- [SAKM09] M. Ali Babar, T. Dingsøy, P. Lago, und H. van Vliet, [Software Architecture Knowledge Management: Theory and Practice](#). Springer, 2009
 - Features a case study chapter on SOA Infrastructure Reference Architecture
- [PESOS09] W.-J. van den Heuvel, O. Zimmermann, F. Leymann, P. Lago, I. Schieferdecker, U. Zdun, and P. Avgeriou, [Software Service Engineering: Tenets and Challenges](#). Proc. of ICSE 2009 PESOS workshop. Basis: [Dagstuhl 2009 seminar report](#)
- [WICSA08] O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann, Combining Pattern Languages and Architectural Decision Models into a Comprehensive and Comprehensible Design Method. Proc. of IEEE WICSA 2008.
 - <http://soadecisions.org/soad.htm#wicsa>
- [Zim09] An Architectural Decision Modeling Framework for SOA Design, Ph. D. thesis, Universität Stuttgart, 2009
 - All concepts introduced in this tutorial are covered in detail (among others)
 - http://elib.uni-stuttgart.de/opus/frontdoor.php?source_opus=5228&la=de (PDF)
 - <http://www.dissertation.de/buch.php3?buch=5918> (book)

References: SOA (1/2)

- [CBDI] Sprott, D.: *On SOA Methodology*, Editorial March 2005 CBDI Journal, <http://www.cbdiforum.com/>
- [Dijkstra] Dijkstra, E. W., *A Discipline of Programming*, Prentice Hall, 1976
- [Evans] Evans E., *Domain-Driven Design*, Addison Wesley, 2003
- [Ferguson] Ferguson D., Storey T., Lovering B., Shewchuk J., *Secure, Reliable, Transacted Web Services*, <http://www.ibm.com/developerworks/webservices/library/ws-secutrans/index.html>
- [Fowler] Fowler M., *Patterns of Enterprise Application Architecture*, Addison Wesley, 2003
- [GoF] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995
- [Hohpe] Hohpe G., *Developing Software in A Service-Oriented World*, White Paper, January 2005, *Enterprise Integration Patterns* website, <http://www.eaipatterns.com>
- [IBM SOA] Service-Oriented Architecture from IBM – Success Stories, Products, Services <http://www.ibm.com/software/solutions/soa>
- [IBM SOMA] Arsanjani A., *Service-Oriented Modeling and Architecture*, <http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/>
- [IBM SSS] Ibrahim M., Long G. *Service-Oriented Architecture and Enterprise Architecture*, <http://www.ibm.com/developerworks/webservices/library/ws-soa-enterprise1/>
- [IBM ITSO] Wahli U., *Application Developer Version 6 Web Services*, IBM ITSO Workshop 2005, <http://www.redbooks.ibm.com>

References: SOA (2/2)

- [Keen] Keen M. et al, *Patterns: Implementing an SOA using an ESB*, IBM Redbook 2004
- [Meyer] Meyer B., *Object-Oriented Software Construction*, Second Edition, Prentice Hall 1997
- [OASIS] Web Services Business Process Execution Language Version 2.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
- [OSI] International Standardization Organisation, Open System Interconnection Basic Reference Model, http://en.wikipedia.org/wiki/OSI_model
- [POSA] Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., *Pattern-Oriented Software Architecture – a System of Patterns*. Wiley, 1996
- [PoWS] Zimmermann O., Tomlinson M., Peuser S., *Perspectives on Web Services – Applying SOAP, WSDL and UDDI to Real-World Projects*, Springer-Verlag, 2003
- [RAMP], Reliable, Asynchronous Messaging Profile 1.0, IBM, Ford Motor Company, DaimlerChrysler, <http://www.ibm.com/developerworks/webservices/library/specification/ws-ramp>
- [SAP] ESA zone of SAP Developer Network (SDN), via <http://www.sdn.sap.com/sdn/esa.sdn>
- [Sauter] Sauter G. et al, *Information Service Patterns, Part 1: Data Federation Pattern* <http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv1/>
- [Schumacher] Schumacher M., Fernandez E.B., Hybertson D., Buschmann F., and Sommerlad P., *Security Patterns: Integrating security and systems engineering*, Wiley 2006.
- [Zdun] Zdun U., Dustdar S., *Model-Driven and Pattern-Based Integration of Process-Driven SOA Models*, <http://drops.dagstuhl.de/opus/volltexte/2006/820>

References: Web Services

- [XML] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, 6 October 2000, <http://www.w3.org/TR/2004/REC-xml-20040204/>
- [XMLNS] Namespaces in XML, W3C, 14 January 1999, <http://www.w3.org/TR/REC-xml-names/>
- [XMLSch] XML Schema W3C Recommendation Parts 0-2, 2 May 2001, <http://www.w3.org/XML/Schema>
- [SOAP] SOAP Version 1.1 W3C Recommendation, (1.2 Rec. since 2007), <http://www.w3.org/2000/soap/Group/>
- [WSDL] WSDL Version 1.1 W3C Note, March 2001 (2.0 Rec. since 2007), <http://www.w3.org/2002/ws/desc/>
- [UDDI] UDDI Version 3.0.2 OASIS Draft, October 2004, http://uddi.org/pubs/uddi_v3.htm
- [SSL] SSL Protocol Version 3.0, Netscape Communications, 1996, <http://wp.netscape.com/eng/ssl3/>
- [TLS] Transport Layer Security 1.0, Internet Engineering Task Force, January 1999, <http://www.ietf.org/html.charters/tls-charter.html>
- [WSS] Web Services Security: SOAP Message Security 1.0 Specification, OASIS, March 2004, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- [WSI] Web Services Interoperability Profiles, WS-I.org, 2004-2008, <http://www.ws-i.org/deliverables/matrix.aspx>
- [JAXRPC] Java API for XML Remote Procedure Calls 1.1 Specification, Java Community Process, October 2003, <http://www.jcp.org/en/jsr/detail?id=101>
- [EWS] Enterprise Web Services 1.1, Java Community Process, November 2003, <http://www.jcp.org/en/jsr/detail?id=921>
- [JAXWS] The Java API for XML Web Services, (JAX-WS) 2.1, May 2007

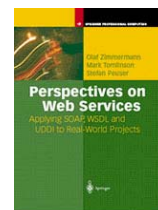
Thank You!

- Questions?
- Comments?

- “Perspectives on Web Services”, Springer-Verlag
 - Captures project experience 2001-2003, incl. 26 reusable architectural decisions
 - Foreword by Grady Booch
 - Website also has case study reports and other referenced papers

WWW: <http://www.soadecisions.org>

IBM Intranet: <http://bpia.zurich.ibm.com/downloads/adkwik>



SATURN 2010 Tutorial: Sample Solutions



© 2010 IBM Corporation

Sample Solutions to Exercise 1: Reflection on Today's Decision Making Practices

1. Which AD hurt your head most on your last project?
 - *Probably too many to mention them all... were they project-specific or recurring? How about service granularity, system transaction boundaries in process-enabled SOA, SOAP vs. REST?*
2. What did you do to resolve it?
 - *Gut feel, architecture workshops, run PoC, consult general-purpose method (with SOA plugin)*
3. Which forces influenced your decision making?
 - *Previous experience, NFRs, general software quality attributes such as performance, scalability*
4. Where did you look for help?
 - *Personal network, online articles, official knowledge repositories*
5. How did you document your decision?
 - *Meeting minutes, component/operational model, ARC 100 (standard/own template)*
6. Were your lessons learned specific to your project, or would you say they are applicable to other client project situations?
 - *All sample decisions mentioned above recur on most if not all SOA projects.*
7. If so, how did you share them with your community (practitioner network)?
 - *No time (had to move on to next project), AoT conference template (PowerPoint)*

Solutions to Exercise 2 and 3a)

- “Which pattern should we pick to structure the integration layer?” (CI)
- “We investigated the Broker pattern from POSA and ESB in Krafzig’s SOA book” (CA)
- “Should we use a binary or a textual representation of service requests?” (CI, 2 CAs)
- “Integration technology” (TI)
- “SOAP over “HTTP” and any WS-* specification (WSDL, WS-Security, WS-Policy) (TA)
- “XML” as “message exchange format” (TA, TI)
- “RESTful integration” (*debatable, both conceptual and technology issue/alternative*)
- “Eclipse Rich Client Platform (RCP) or Java-based Web application?” (TAA/VAA, TA)
- “Apache Mule” as “ESB implementation and provider” (VAA, VAI)
- “HSQLDB from Source Forge.net” (VAA)
- “We could use the IBM WebSphere application server, seems to have market traction” (VAA and outcome)
- “We decided to use Oracle Business Process Manager, licenses already purchased (enterprise license agreement in place)” (VAA and outcome)

CI – Conceptual Issue, CA – Conceptual Alternative, TI – Technology Issue, TA – Technology Alternative,
VAI – Vendor Asset Level Issue, VAA – Vendor Asset Level Alternative

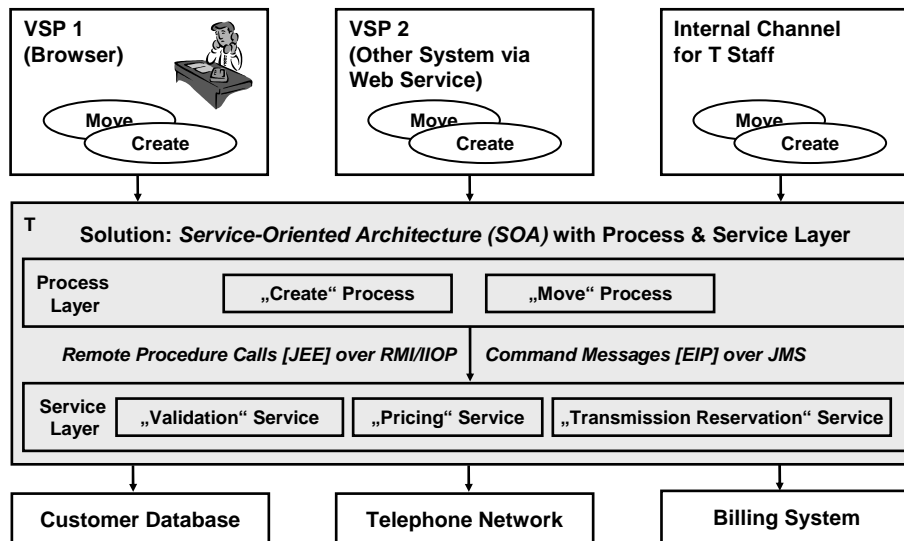
Sample Solutions to Exercise 3c)

1. Download the WWW 2008 paper “RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision”
 - Available at <http://soadecisions.org/soad.htm#www>
2. Review Tables 1, 2, and 3 in that paper:
 - Which levels are covered?
 - *The Conceptual and the Technology Level. The Vendor Asset Level is mentioned, not included due to space constraints. As the theme of the paper is technology comparison, there is no Business Executive Level, but coverage of architectural principles (“decision drivers” in SOAD).*
 - How many architectural decisions are discussed?
 - *9 conceptual decisions, 10 technology decisions (plus 3 principles).*
 - How do the decisions map to ADTopics defined in the Reusable ADM for SOA (RADM for SOA) that we introduced in this module?
 - *Conceptual Level: Layer3ServiceRealizationDecisions (and sub-topics, not shown)*
 - *Technology Level: Layer3WebServiceBindingDecisions, Layer3RestfullIntegrationDecisions*
3. (optional) Create your own project in Architectural Knowledge Decision Web Tool and capture the ADs and ADAalternatives discussed in the paper.
 - *No sample solution provided*

Architectural Decision (AD) about Integration Style

Subject Area	Process and service layer design	Topic	Integration
AD	Integration Style	AD ID	3
Decision	We decided for RPC and the Messaging pattern from the [EIP] book		
Issue or Problem	How should process activities and underlying services communicate?		
Assumptions	Process model and requirements NFR 1 to NFR 7 are valid and stable		
Motivation	If logical layers are physically distributed, they must be integrated.		
Alternatives	File transfer, shared database, no physical distribution (local calls)		
Justification	This is an inherently synchronous scenario: VSP users as well as internal T staff expect immediate responses to their requests (NFR 5). Messaging will give us guaranteed delivery (NFR 3, NFR 6).		
Implications	Need to select, install, and configure a message-oriented middleware.		
Derived Requirements	Many finer grained patterns are now eligible and have to be decided upon [EIP]: message construction, channel design, message routing, message transformation, system management.		
Related Decisions	Next, we have to decide on one or more integration technologies implementing the selected two integration styles. Many alternatives exist, e.g., Java Message Service (JMS) providers.		

Architecture Overview Diagram (Second Design Iteration)



Sample Solution for Optional Technical Questions (1/2)

- Which requirements engineering method do you propose, and why?
 - *BPM due to multi-actor nature of problem and process control requirements*
 - *Use cases in Object-Oriented Analysis and Design (OOAD) for user interface design and detailed actor-system interactions at the system boundary*
 - *User stories (agile community)*

Sample Solution for Optional Technical Questions (2/2)

- Are you content with the introduction of a process manager into the architecture (and why)?
 - *See OOPSLA 2005 practitioner report for more detailed rationale: <http://soadecisions.org/download/pr07-zimmermann1.pdf>*
 - *BPEL chosen – standardized, engine support, market momentum*
 - *IBM WebSphere Process Server (actually, predecessor product)*
- Which remote communication protocol would you use to support the chosen integration style, making the services available to the processes? Name the relevant technology standard.
 - *Web services: SOAP over HTTP or SOAP over JMS providers, due to standardization and tool support (interoperability proven on many projects)*
 - *Alternative: RESTful integration over plain HTTP*
 - *Many more options*