Abstract

This presentation provides an overview of completed and ongoing architectural knowledge management activities in an industrial setting.

After a brief review of the concepts for architectural decision modeling with reuse presented at SATURN 2010, we conduct an in-depth investigation of ten particularly relevant and recurring SOA design decisions, and outlines several decisions that must be made when a cloud-computing solution is designed.

Furthermore, the presentation covers how we extended existing decision modeling concepts to target the designers of outsourcing solutions and introduces tooling options for working with the presented guidance models and for decision making.

Finally, we outline directions for future work.
Agenda

- Architectural decision modeling concepts
- Recurring SOA design decisions
- Guidance models for other domains
  - Cloud computing
  - Strategic outsourcing
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary
What are Architectural Decisions? Why Bother?

- “The design decisions that are costly to change” (Grady Booch, 2009)

  “Architectural decisions capture key design issues and the rationale behind chosen solutions. They are conscious design decisions concerning a software-intensive system as a whole or one or more of its core components and connectors in any given view. The outcome of architectural decisions influences the system’s nonfunctional characteristics including its 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”

Architectural Decision (AD) about Integration Style – Documented according to IBM Unified Method Framework (UMF)

<table>
<thead>
<tr>
<th>Subject Area</th>
<th>Process and service layer design</th>
<th>Topic</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Integration Style</td>
<td>AD ID</td>
<td>3</td>
</tr>
<tr>
<td>Decision Made</td>
<td>We decided for RPC and the Messaging pattern (Enterprise Integration Patterns)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue or Problem</td>
<td>How should process activities and underlying services communicate?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assumptions</td>
<td>Process model and requirements NFR 1 to NFR 7 are valid and stable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motivation</td>
<td>If logical layers are physically distributed, they must be integrated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternatives</td>
<td>File transfer, shared database, no physical distribution (local calls)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Justification</td>
<td>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).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implications</td>
<td>Need to select, install, and configure a message-oriented middleware.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived Requirements</td>
<td>Many finer grained patterns are now eligible and have to be decided upon: message construction, channel design, message routing, message transformation, system management (see Enterprise Integration Patterns book).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related Decisions</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Architectural Decisions in Enterprise Application Architectures

Presentation Layer
- Web Users
- Other Systems

Business Logic Layer
- Business Processes
- Business Services
- Components

Backend Integration & Persistence Layer
- Other Systems
- Databases

AD: “We decided for pattern/technology/product X to resolve issue Y because of requirement Z”

Observation:
Required architectural decisions not specific to the case – they recur

Challenges:
1. SOA literature does not make required decisions explicit
2. Hundreds of decisions to be made
3. Decision making order unclear

Entity Types and Associations in UML Metamodel

Guidance Model
Decisions Required and Candidate Solutions

“Model View Controller (MVC) is a common architectural pattern to control the Web page flow.”

Problem and criteria

Decision Model
Decisions Actually Made on Projects

“Our extension
Potential solutions with pros and cons

UMF template (ART 0513/ARC 100)
Value of Architectural Decision Modeling

Agenda

- Architectural decision modeling concepts
- Recurring SOA design decisions
- Guidance models for other domains
  - Cloud computing
  - Strategic outsourcing
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary
From AD Documentation to Active Method Guidance

**500+ Recurring Decisions in IBM SOA Decision Guidance Model (SOAD)**

**Architectural Style**

SOA or other?

**Conceptual Level**

Service Composition Paradigm
(Processes? Classes?)

**Technology Level**

Workflow Language
(BPEL? Other?)

**Vendor Asset Level**

BPEL Engine
(IBM WPS? Other?)

**Decision Issue#10 – Reference Architecture (RA)**

**Issue: Reference Architecture Selection (Business Executive Level)**

Which reference model should be used to define architectural layers?

Background reading: Kraftig et al “Enterprise SOA”, Josuttis “SOA in Practice”

**Alternative 1:** SOMA/SoA Solution Stack 5+2 or 5+4 Model (IBM)

Pros: Platform-neutral, comprehensive, widely adopted

Cons: Rather abstract, has to be refined for projects and solutions

**Alternative 2:** RA from Other Software Vendors and Professional Services Firms

Consult the respective websites and developer networks.

Pros: Often close to implementation reality.

Cons: May promote proprietary concepts.

**Alternative 3:** RA from Standards Body or Book Author

E.g., The Open Group is in the process of standardizing a SOA reference model. OMG SysML extends UML to define core SOA concepts. “Enterprise SOA” and “SOA in Practice” come with their own approaches.

Recommendation: All listed reference models have their place. Whichever one you choose, make sure to profile to relevant subset and to provide concrete usage examples.

**Decision Identification**

**Decision Making**

**Decision Enforcement**
AD Issue #9 – Overall Integration Layer Design Pattern

Scope: Entire Integration Layer

Decision drivers: Number and technical diversity of service providers and consumers, change dynamics

Phase: Solution Outline
Role: Lead Architect

Component Model, Operational Model

Message Exchange Pattern
Integration Technology
Message Broker/ESB Product

ESB: Enterprise Service Bus

Reference Architecture Selection
System Context Diagram

Issue: Integration Style (Conceptual Level)
Which architectural pattern (if any) is selected to structure the integration layer?

Background reading: Hohpe/Woolf “Enterprise Integration Patterns”

Alternative 1: Vanilla Message Broker or SOA ESB
Hub-and-spoke (stateful or stateless) with broker as hub, message channels as spokes
Pros: Flexible and extensible, can mediate technical differences
Cons: There is a risk of scope creep – business logic might end up in integration layer.

Recommendation: Introduce ESB pattern if loose coupling (i.e., location, format, protocol, and implementation transparency) is a valued strategic architectural principle.

Alternative 2: Peer-to-Peer Integration (No Integration Layer)
Each service consumer directly talks to one or more service providers, e.g. via file transfer, shared database, or remote procedure invocation patterns.
Pros: Less architectural components
Cons: Many connectors, maintenance and extensibility issues, development and management effort. Consumers and providers tightly coupled.

AD Issue #8 – A Detailed Integration Layer Design Issue

Scope: Operation

Decision drivers: Reliability needs, systems management capabilities, availability of provider

Phase: Macro Design
Role: Integration Architect

WSDL
Integration Style

Service Model

Issue: Message Exchange Pattern (Conceptual/Technology Level)
Do consumer and provider communicate synchronously or asynchronously?

Background reading: Hohpe/Woolf “Enterprise Integration Patterns”

Alternative 1: Request-Reply SOAP/HTTP
Simple to manage, but no guarantied delivery, so “might” have to deal with undelivered and/or duplicate messages

Alternative 2: One Way over Reliable Transport JMS or WS-RM
Consumer and provider up times can differ; guaranteed delivery (once and only once) when using persistent messages. Must manage dead letter queue.

Alternative 3: Pseudo-Asynchronous Combination of Alternative 1 on application integration layer, Alternative 2 on underlying transport layer
Same as Alternative 1, but guaranteed delivery

Recommendation: Do not follow an MOM hype – decoupling in time is just one of several dimensions of loose coupling. The equation (NOT RM => NOT SOA) does not hold true.
AD Issue #7 – Integration Layer Technologies

**Integration Style**
- **Message Exchange Pattern**
  - **Component Model**
  - **Scope:** Operation
  - **Decision drivers:** Quality of Service (QoS) needs, endpoint technologies, team preferences
  - **Phase:** Macro Design
    - Rule: Integration Architect

**Alternative 1:**
- **WS-* Technologies**
  - SOA/HTTP, WSDL and other appropriate XML languages
  - **Pros:** Tool and standards support, interoperability
  - **Cons:** Perceived to be complex and under vendor control (tools and middlewares needed for XML editing and processing). Limited support in scripting languages.

**Alternative 2:**
- **RESTful Integration**
  - HTTP with certain principles (see PhD thesis by Roy Fielding)
  - **Pros:** HTTP ubiquity, scalability, perceived to be simple
  - **Cons:** No interface definition language and few tools, so many integration responsibilities shifted back to application developer. Does not support asynchronous messaging. Not transactional.

**Recommendation:** Both alternatives have their place. Avoid being biased in the decision making. Note that REST often is positioned as an architectural style (alternative to SOA).

AD Issue #6 – Overall Process Layer Design Pattern

**Issue: Service Composition Paradigm (Conceptual Level)**
- **Should a process layer be introduced?** If so, how is service composition done?
  - **Background reading:** Leymann/Roller “Production Workflow”

**Alternative 1:**
- **Workflow (BPM)**
  - Executable process with sub-processes, activities, etc.
  - Highly expressive, tool and engine supported, separates composites from atomic services
  - Additional programming paradigm (graph-based)

**Alternative 2:**
- **Composite pattern via modern programming language (e.g., OO)**
  - Proven, used in other architectural layers
  - Does not provide engine support for business transactions; runs the risk of not structuring the business logic layer properly (into sub-layers).

**Alternative 3:**
- **Other Layer Composition responsibilities may also be taken by presentation layer (mashups) or by integration layer (stateful broker)**

**Recommendation:** Introduce process layer and realize it with workflow pattern and technologies if business scenario is long running multi-actor scenario with advanced resource coordination/protection requirements. Use OO composition in other cases.
AD Issue #5 – Transaction Management Topic

Issue: Invocation Transactionality Pattern (Conceptual Level)
Should a business process, its activities, and the service components it invokes run in a single or in multiple system transactions?
See ICSOC 2007 paper by Zimmermann et al. for available patterns.

Alternative 1: Transaction Islands
Do not share Tx context
Best performance, loose coupling, but no full ACID protection for resources.

Alternative 2: Transaction Bridge
Share Tx context
Best resource protection, but large, long running Tx tightly coupling activities and services.

Alternative 3: Stratified Stilts
Use asynchronous messaging and suspend Tx
Supports loose coupling best, but no full ACID protection.

Recommendation: Use Transaction Islands as default, Stratified Stilts for long running distributed processes. Decision injection into model transformation or BPEL code in WebSphere Integration Developer is possible.

AD Issue #4 – Process Layer Technologies

Issue: Workflow Language (Conceptual Level)
Which BPM/workflow language should be selected?
Background reading: Websites of standards bodies and analyst reports

Alternative 1: WS-BPEL 2.0 (a.k.a. BPEL) from OASIS
Standardized, partially supported by tools and engines

Alternative 2: BPEL 2.0 predecessor such as BPEL 1.x, WSFL, or FDL/MIL
Might expose engine-specific features such as dead path elimination better than standard.

Alternative 3: Other Standardized or Proprietary Language
Other service composition and BPM paradigms exist, e.g. based on Petri Nets rather than graphs.

Recommendation: This decision is often constrained by the workflow engine selection (if that decision is made upfront). Look for standards compliance if portability is an important requirement; avoid usage of proprietary language features during process modeling.
AD Issue #3 – Service/Component Layer Design Patterns

Issue: Service and Component Layer Patterns (Conceptual Level)

How should the service and the component layer be organized?
Background reading: Fowler “Patterns of Enterprise Application Architecture”, Buschmann et al “Patterns of Software Architecture”, Gamma et al “Design Patterns”

Recommendation: Follow a best-of-breed approach and select patterns from multiple languages as justified by project requirements and already existing designs. E.g., apply the façade/wrapper pattern to separate the service layer from the component layer. Make services as stateless as possible (Client State or Database State patterns). Identify, make, and enforce pattern adoption decisions.

GOF: Gang of Four
POSA: Patterns of Software Architecture
PoEAA: Patterns of Enterprise Application Architecture

Alternative 1: Traditional GOF, POSA and PoEAA patterns
Façade, Broker, Domain Model are three of many applicable ones.

Alternative 2: Domain-Driven Design (DDD) patterns (Evans)
Context and Ubiquitous Language patterns are particularly useful.

Alternative 2: Core J2EE Patterns
See book by Alur et al
Proven for low level design and coding (not all are architectural)

Alternative 3: Patterns from Other Languages
See Software Architecture Handbook website by Bosch.

Recommendation: Follow a best-of-breed approach and select patterns from multiples languages as justified by project requirements and already existing designs. E.g., apply the façade/wrapper pattern to separate the service layer from the component layer. Make services as stateless as possible (Client State or Database State patterns). Identify, make, and enforce pattern adoption decisions.

AD Issue #2 – Service/Component Layer Technologies

Issue: Container Technology (Technology Level)

Which interface definition language and container technology is used?
Background reading: Web portals such as InfoQ and TheServerSide.com

Recommendation: This decision is often constrained by service container selection. Look for standards compliance if portability is an important requirement. Find a balance between container features (services) and maintainability and keeping architectural control.

WSDL: Web Service Description Language
OSOA: Open SOA
EJB: Enterprise JavaBean
AD Issue #1 – Addressing Service Granularity Topic

Scope: Service Operation  
Role: Service Modeler  
Phase: Macro Design

**Decision drivers:** Functional requirements (domain model), capabilities of BPEL, SOAP, WSDL, XML processors (verbosity), interoperability, network topology, number of deployment artifacts and generated code structure, strong vs. weak typing philosophy.

**Issue:** In Message Granularity (Conceptual/Technology Level)

- **How many message parts should be defined in the service contract?**
- **How deep should the part elements be structured?**

The four alternatives have not been published as patterns yet.

**Alternative 1:** Dot Pattern  
Single scalar parameter  
Easy to process for SOAP/XML engines, much work for programmer

**Alternative 2:** Bar Pattern  
Single complex parameter  
Deep structure and exotic types can cause interoperability issues.

**Alternative 3:** Dotted Line Pattern  
Multiple scalar parameters  
Handled by all common engines, some programmer convenience.

**Alternative 4:** Comb Pattern  
Multiple complex parameters  
Combination of options 2 and 3, biggest overhead for processing engines.

**Recommendation:** All alternatives have their place; alternatives 2 and 3 are often chosen. Base decision on layer and service type. Avoid overly deep nesting of data structures unless you want to stress test the XML processing. Minimize message verbosity.

---

**Agenda**

- Architectural decision modeling concepts
- Recurring SOA design decisions
- **Guidance models for other domains**
  - Cloud computing
  - Strategic outsourcing
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary
Cloud Design Issue #1 – Cloud Virtualization Layers

**Scope:**
Project, Program, or Enterprise

**Decision drivers:**
Projects, goals, e.g., service management engagements striving to cut infrastructure cost and time to deployment vs. development and integration projects looking for utility-based sourcing options.

**Issue:** Cloud Virtualization Layers

Which layers of the general computing stack (hardware -> operating system -> middleware -> application) should be "cloudified" (virtualized)?

*Background reading: IBM Global Technology Outlook (GTO) 2009, NICS*

**Alternative 1:**
Infrastructure as a Service (IaaS)
Storage, server, and/or network virtualization

**Alternative 2:**
Platform as a Service (PaaS)
Database, integration broker, application server, portal server

**Alternative 3:**
Software as a Service (SaaS)
Application components and/or SOA services

**Recommendation:** Follow a crawl-walk-run approach, e.g., bottom up from infrastructure to platform to application (or top-down from application to platform and infrastructure). For infrastructure providers and consultants, IaaS is typically the first because its at the bottom of the stack and, from an operational standpoint, has the most cost savings to contribute. But it's not necessarily the easiest. An additional primary "pro" for IaaS is also the savings that come from standardization on automation, standard stacks, etc.

Cloud Design Issue #2 – Workload Type

**Scope:**
Project, Program, or Enterprise

**Decision drivers:**
Risk and migration cost (transformation effort and complexity effect); standardization of workload, footprint (CPU and storage resource consumption, network communication channels).

**Issue:** Workload Type

Which application genres and/or architectural styles and/or application and middleware components and/or presentation/business logic/data deployment units should be deployed into the cloud?

*Background reading: See Booch's Software Architecture Handbook project for a classification of application genres as a source of workload types.*

**Alternative 1:**
Development tools and test environments (SaaS)

**Alternative 2:**
ERP workloads, Web applications, collaboration infrastructures, high volume/low cost analytics

**Alternative 3:**
Other workloads that can be standardized for cloud deployment or are made possible by cloud computing

**Recommendation:** Workloads which depend on sensitive data normally restricted to the enterprise often can not be moved into a cloud. E.g., most companies are not ready to move their LDAP server into a public cloud because of sensitivity of employee information. Same for health care records (unless security of cloud provider is well established); workloads composed of multiple, co-dependent services, high throughout online transaction processing; workloads requiring a high level of auditability and accountability; workloads based on 3rd party software which does not have a virtualization or cloud-aware licensing strategy; workloads requiring detailed chargeback or utilization measurement as required for capacity planning or departmental level billing; adaptive workloads requiring runtime flexibility and customization.
Cloud Design Issue #3 – Cloud Exposure Type

Issue: Cloud Exposure Type
Which human users and systems will the cloud infrastructure be exposed to?

Background reading: See introduction to cloud computing and other cloud computing resources that is available at http://www.ibm.com/ibm/cloud/

Recommendation: Use public clouds to host situational applications; use private clouds for company-internal environments such as test beds. Attributes of a private cloud: customization, efficiency, security and privacy, availability. Attributes of a public cloud: standardization, capital, preservation, flexibility, reduced time to deploy.

Cloud Design Issue #4 – Server Availability Class

Issue: Server Availability Type
Which conceptual Operational Model (OM) should be supported by the virtualized IT infrastructure?

Background reading: General OM and topology modeling literature

Recommendation: The final decision per IaaS/PaaS service and workload type is made by the cloud user during cloud operations; however, the IT service types made available to the user must be decided during cloud design. A generic best practices recommendation is not appropriate for this issue.
Cloud Design Issue #5 – Service Automation Patterns

Issue: Service Automation Patterns (Conceptual Level)
How are virtualized services deployed into a cloud platform?
How are they managed throughout their lifetime?

Background reading: Tivoli Service Automation Manager documentation

Recommendation: Combine alternatives 1 and 2 to achieve cloud-scale efficiencies (e.g., with respect to time-to-deploy and admin-per-managed element ratio). Consider to fall back to alternative 3 for low-end, entry point cloud offerings with moderate NFRs.

Component Model

Decision Identification Decision Enforcement Decision Making

Cloud Computing Go/No Go Decision

Cloud Virtualization Layers

Workload Type

Cloud Exposure Type

Scope: Project, Program, or Enterprise

Decision drivers: Project goals, amount of images and users, change dynamics, standardization of workload

Phase: Solution Outline Role: Lead Architect

Component Model

Issue:

Alternative 1: Image-Based Build Template

Alternative 2: Image-Based Management Template

Alternative 3: Manual or Traditional Service/Systems Management

Recommendation: Combine alternatives 1 and 2 to achieve cloud-scale efficiencies (e.g., with respect to time-to-deploy and admin-per-managed element ratio). Consider to fall back to alternative 3 for low-end, entry point cloud offerings with moderate NFRs.

Cloud Design Issue #6 – VMWare Resource Pool Layout

Issue: VMWare Resource Pool Layout (Tech./Vendor Asset Level)
How many Virtual Machines (VMs) should be defined in each resource pool? How many resource pools should be defined in a cluster?

Background reading: VMWare documentation, http://www.vmware.com/technology/

Recommendation: Stay in the middle of the road, do not stress VMware's capabilities by defining an excessive amount of virtual machines in a resource pool and by defining an excessive amount of resource pools per cluster. For quantitative information, consult the vendor documentation.

Cluster Design

Scope: Project, Program, or Enterprise

Decision drivers: Usability of system administrator interface, variability of requirements, access patterns, security principles

Phase: Solution Outline Role: Lead Architect

Architecture Overview Diagram

Image File Placement

Scope: Project, Program, or Enterprise

Decision drivers: Usability of system administrator interface, variability of requirements, access patterns, security principles

Phase: Solution Outline Role: Lead Architect

Architecture Overview Diagram

Image File Placement
Cloud Design Issue #7 – Asset Management Approach

Issue: Asset Management Approach
How should the portfolio of provisioned VMs and their owners be kept track of?
Background reading: See ITIL for information about asset management; see COBIT and other standards for information on IT audits.

Recommendation: Treat VMs like regular machines initially (to avoid extra effort for audit setup) – assuming that this does not compromise project goals such as reaching cloud-scale efficiencies. If so, perform a tradeoff analysis.

Alternative 1:
Treat a VM like a physical asset (asset management database entry, audits, etc.)
No changes in admin procedures needed, but might not be able to reduce admin-per-server and time-to-deploy goals.

Alternative 2:
Treat a VM as a software application
No serial numbers have to be assigned so easy to realize, but might violate user access management, data privacy, and other regulatory compliance requirements. Might also limit ability to track VM usage (in audit trail).

Decision Identification -> Decision Making -> Decision Enforcement

Agenda

- Architectural decision modeling concepts
- Recurring SOA design decisions
- Guidance models for other domains
  - Cloud computing
  - Strategic Outsourcing (SO)
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary
We investigated the applicability of SOAD concepts for SO...

- Design of Strategic Outsourcing (SO) solutions can be seen as a superset of software architecture design:
  - Server and storage infrastructures, human resources, real estate in scope
  - Many business and technical decisions must be made prior to contract signature (due to market forces)
  - Sales executives, project managers, and domain SMEs (e.g., for legal matters) are also involved in the solution design

- Five immanent knowledge management challenges identified through stakeholder interviews:
  1. Scope and scale
  2. Priority and order
  3. Data quality and uncertainty
  4. Consistency and efficiency
  5. Reuse and education

... and engineered a decision guidance model for deal design.

- Metamodel adjustments (e.g. improved workflow)
- 100+ decision points modeled
- Contributions and reviews by domain subject matter experts (SMEs)
- Deal replays followed by pilot releases

- Novel, standalone Solution Decision Advisor (SDA) based on Eclipse Rich Client Platform (RCP)
- A WICSA 2011 paper provides more information on SO domain and knowledge management solution for it:

Agenda

- Architectural decision modeling concepts
- Recurring SOA design decisions
- Guidance models for other domains
  - Cloud computing
  - Strategic outsourcing
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary

Three Usage Scenarios for Architectural Decisions

**Scenario 1: After-the-Fact Decision Capturing**
1.1 Document decisions made and their rationale (supporting decision log report generation)

**Scenario 2: Active Method Guidance**
2.1 Distinguish decisions required (issues) from decisions made (outcomes)
2.2 Share issues and related best practices via guidance models
2.3 Assign design work items (issues with open outcomes) to team members and track decision making progress (a.k.a. "backlog")

**Scenario 3: Cross Role Collaboration**
3.1 Identify issues in requirements artifacts and trace their resolution
3.2 Bind architectural decisions to enterprise-wide architectural principles and reusable assets such as pattern repositories
3.3 Enforce decisions in UML and topology models
3.4 Integrate with process tasks
3.5 Measure decision making practices (model analysis)
AD Tooling Options – Overview

- No modeling tool (do-it-yourself)
  - E.g. word processor or spreadsheet (template-based)
  - E.g. wiki, Web 2.0 community software, groupware team rooms
- Architectural Decision Knowledge Wiki/Web Tool on IBM alphaWorks
  - No longer actively developed, but decision content (samples) still relevant (full SOA guidance model available via services engagements)
- Architectural Decision Knowledge tools by IBM Rational/IBM Research (emerging)
  - Eclipse/Jazz-based version first presented at Innovate 2010
  - Solution Decision Advisor
- Several more research prototypes from universities exist
  - See Philippe Kruchten’s SATURN 2010 tutorial and ECSA 2010 keynote on Architectural Knowledge Management (pages 72-73)

Architectural Decisions Knowledge tools (Rational/RES)

- 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
Agenda

- Architectural decision modeling concepts
- Recurring SOA design decisions
- Guidance models for other domains
  - Cloud computing
  - Strategic outsourcing
- SOA Decision Modeling (SOAD) tooling
- Discussion and summary

SOA Lessons Learned

- SOA concepts and Web services technologies work
  - Interoperability proven
  - Performance not an issue
- Same old architecture?
  - Broker vs. Enterprise Service Bus
  - Workflow vs. service composition
  - Directory service vs. registry
- Not all patterns always have to be used
  - Judge and decide on a pattern-by-pattern base
  - Decision making driven by (non-)functional project/program requirements
  - Don’t confuse concepts and technology (SOA vs. Web services, REST)
- Follow a crawl, walk, run approach
  - As for any other non-trivial problem/solution
  - Don’t over-engineer (e.g. complex XML schema)

www.soadecisions.org
Summary and Discussion

- Architectural decision making is a key responsibility of IT architects which is often underestimated and underrepresented in existing methods and tools.
- In SOA design and other domains, many decisions recur. This makes it possible to reduce the documentation effort and to share architectural decision knowledge including best practices (design acceleration and quality assurance).
- Tool support for decision modeling with reuse is available/emerging.
- We would like to hear from you now…
  - … are the presented scenarios, concepts, and tool features useful and usable?
  - … would you have additional requirements, e.g. collaboration and integration needs?

More Information and Upcoming Events

- SOAD overview article and full coverage:

- SOAD tutorial and case study reports:
  - http://soadecisions.org/soad.htm

- Upcoming conferences and workshops with focus on architectural knowledge and SOA/Web services design:
  - ICSE SHARK 2011 (May 24, Honolulu, Hawaii, USA)
  - IEEE WICSA 2011 (June 20-24, Boulder, Colorado, USA)
  - IEEE ECOWS 2011 (Sept 14-16, Lugano, Switzerland)
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. server) and a service requestor (a.k.a. consumer or client).

- 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).
SOA Patterns Overview

- No industry consensus on SOA principles and patterns yet:
- Each author defines his/her own – many terminology mismatches
Case 2: Multi-Channel Order Management (B2B)

- **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

Vision: Enterprise-Wide Guidance Model

- Enterprise architecture group owns and maintains guidance model
  - Input comes from solution architects on development/integration projects
  - Quality assured, aligned with enterprise IT strategy
- Does not mandate a particular architecture, but frames design work
  - Recommend certain alternatives:
    - E.g. "always use document/literal SOAP"
  - Ban others:
    - E.g. "no open source assets can be used due to open legal issues"
  - Finding a balance between freedom-of-choice and freedom-from-choice
- Allows project teams to share lessons learned and best practices
  - Actionable enterprise architecture
  - Enterprise architects perceived as friends, not foes