



Business Integration Technologies

# Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method

Feb 21, 2008  
WICSA 2008

Olaf Zimmermann  
Uwe Zdun  
Thomas Gschwind  
Frank Leymann

## Acknowledgements

- The paper presents results from joint work of several different organizations (and communities):
  - Olaf Zimmermann (presenting) and Thomas Gschwind, Business Integration Technologies Group, IBM Zurich Research Lab, Switzerland
  - Uwe Zdun, Information Systems Institute, Technical University Vienna, Austria
  - Frank Leymann, Institute for Architecture of Application Systems, University of Stuttgart, Germany
  
- Special thanks to the WICSA 2008 reviewers!

## Abstract

When constructing software systems, software architects must identify and evaluate many competing design options and document the rationale behind any selections made. Two supporting concepts are pattern languages and architectural decision models. Unfortunately, both concepts only provide partial support: Extensive upfront education is needed for practitioners to be in command of the full pattern literature relevant in their field; retrospective architectural decision modeling is viewed as a painful extra responsibility without immediate gains.

In this paper, we combine pattern languages and reusable architectural decision models into a design method that is both comprehensive and comprehensible. Our design method identifies the required decisions in requirements models systematically, gives domain-specific pattern selection advice, and provides traceability from platform-independent patterns to platform-specific decisions. We validate our approach by applying it to enterprise applications as an exemplary application genre and a SOA case study from the finance industry.

# Agenda

1. Case study
  - Broker-based integration of core banking application components
2. Existing work
  - Architectural patterns and decision capturing
3. ArchPad walkthrough
  - Refinement stages
  - Reusable Architectural Decision Model for SOA (RADM for SOA)
4. Discussion
  - Tool support
  - Validation
5. Conclusions and outlook

# Agenda

## 1. Case study

- **Broker-based integration of core banking application components**

## 2. Existing work

- Architectural patterns and decision capturing

## 3. ArchPad walkthrough

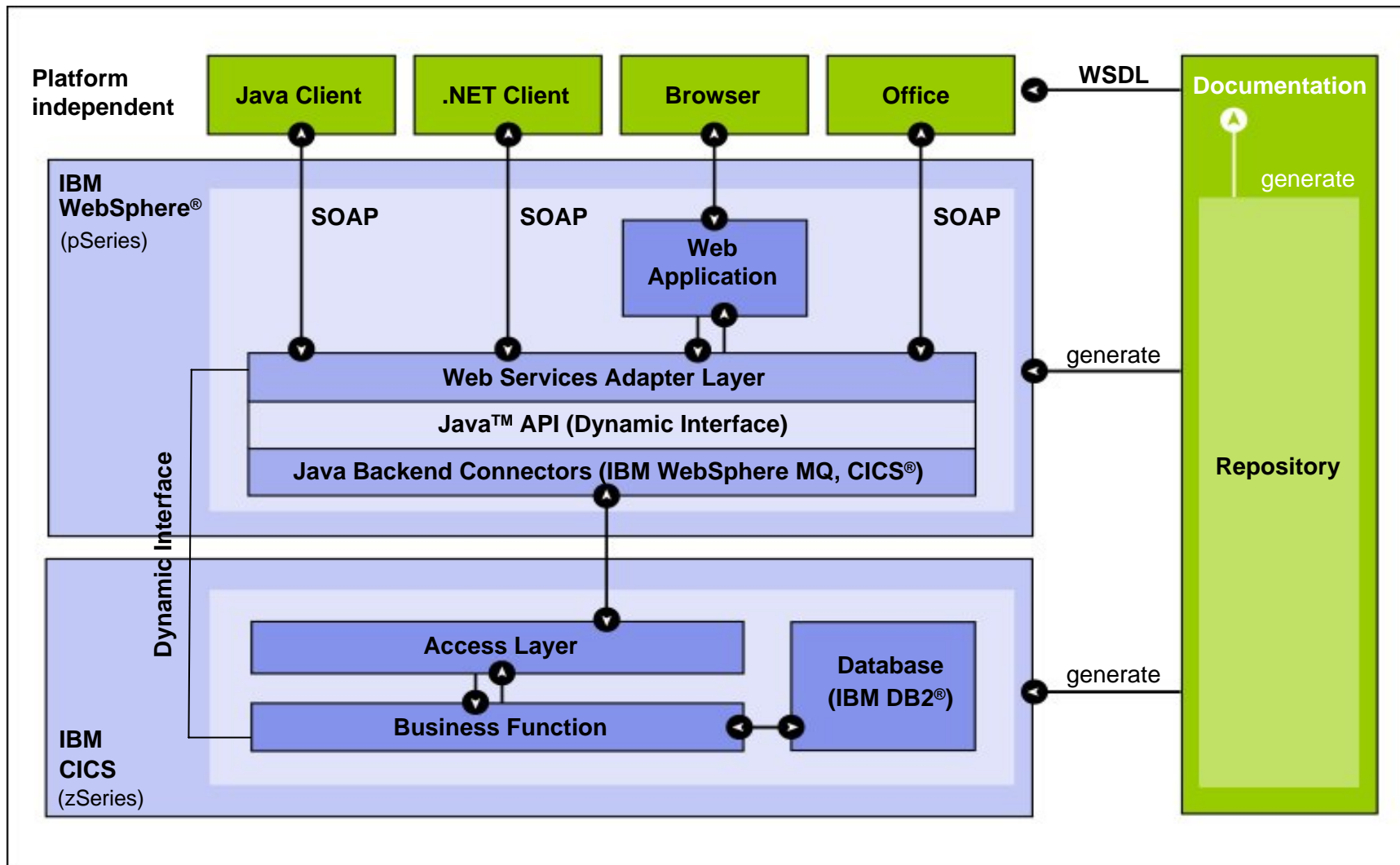
- Refinement stages
- Reusable decision model

## 4. Discussion

- Tool support
- Validation

## 5. Conclusions and outlook

# Case Study: Core Banking SOA with Web Services



Zimmermann, O.; Milinski, M.; Craes, M.; Oellermann, F.: **Second Generation Web Services-Oriented Architecture in Production in the Finance Industry**, OOPSLA Conference Companion, 2004

## 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 (design) patterns are suited to implement the steps? What about the required technology choices? Product selections?*

# Agenda

1. Case study
  - Broker-based integration of core banking application components
2. **Existing work**
  - **Architectural patterns and decision capturing**
3. ArchPad walkthrough
  - Refinement stages
  - Reusable Architectural Decision Model for SOA (RADM for SOA)
4. Discussion
  - Tool support
  - Validation
5. Conclusions and outlook

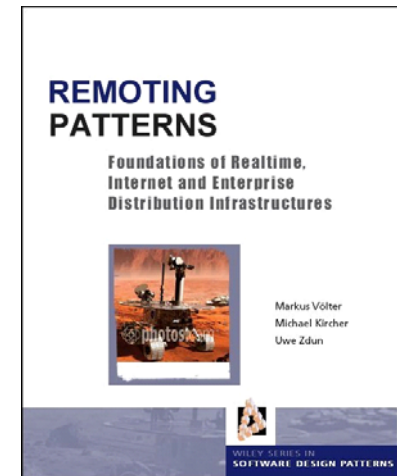
# Architecture and Design Patterns

- In the last couple of years, patterns have become part of the mainstream of object-oriented software development
  - Single pattern is one solution to a particular, recurring problem
  - However, “real problems” are more complex
- Different kinds of patterns:
  - Design patterns (GoF)
  - Software architecture patterns (POSA, POSA2)
  - Analysis patterns (Fowler, Hay)
  - Organizational patterns (Coplien)
  - Pedagogical patterns (PPP)
  - Many others

***“Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.”  
(Coplien summary of Alexander’s definition)***

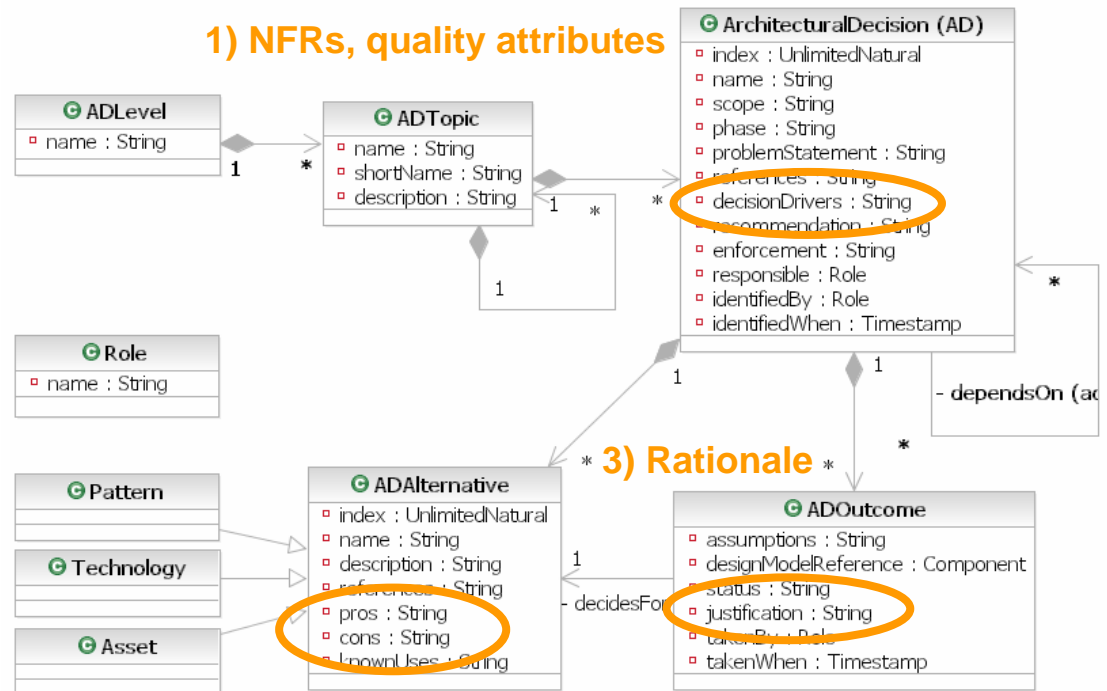
## Remoting Patterns

- Numerous projects
  - *use,*
  - *extend,*
  - *integrate,*
  - *customize,* and
  - *build*
- distributed object middleware
- Goals:
  - illustrate the general, recurring architecture of successful distributed object middleware
  - illustrate more concrete design and implementation strategies
- **Book: published in Wiley’s Pattern Series in 2004**



# Architectural Decision Modeling (ADM) Overview

- Architectural decisions capture key design issues and the *rationale* behind a chosen solution:
  - Conscious design decisions concerning a software system as a whole, or one or more of its core components
  - With an impact on non-functional characteristics and quality factors



- From retrospective, template-based capturing to proactive decision *modeling*
  - Decision drivers include quality attributes (forces) and architectural principles
  - Scope, phase, and role attributes provide method alignment

Zimmermann O., Gschwind T., Küster J., Leymann F., Schuster N., **Reusable Architectural Decision Models for Enterprise Application Development**. In: Third International Conference on the Quality of Software-Architectures (QOSA 2007).

# Patterns and Decisions in Comparison

## Main use cases

Characteristic/Requirement	Pattern Languages	ADMs	Assessment
Intent and main use case	Capture generic architecture and design elements as reusable solutions to commonly occurring problems	Capture and record system-specific decisions justified by project-specific decision drivers	Complementary, strong relationship; pattern application must also be captured as a decision on a project
Standard description format	Several templates, e.g., Portland style	Several meta models, UML profiles	Present in both approaches, overlap
Level of detail	Comprehensive, detailed	Terse, telegram style, details elsewhere	Patterns more detailed by definition
Top-down decomposition of problem into atomic units of design work	Objective of pattern languages and context sections; approach depends on author, often informal	Modeled explicitly in our proposal [26]: AD topic groups, AD levels; AD consists of AD alternatives	Pattern languages often informal, ADMs can add project-specific concretization and modeling rigor
Bottom-up composition of atomic units of design work into solution	Not a design goal	Not a design goal of retrospective AD capturing; supported in our proposal	Additional work for project team, integration effort required
Relationships between atomic units of design work	Informally via consequences, related patterns or pattern language diagrams (design spaces emerging, see [23])	Design goal, addressed by explicit dependency management via associations in UML meta model (directed graph)	ADMs more precise, can be populated from pattern texts
Requirements management link	Context, forces sections	Decision driver attribute in AD, decision identification step	Explicit in our proposal, informal in patterns
Links to software engineering and project management methods	Informal only, through other related patterns, or out of scope	Modeled in our proposal: scope, phase, role attributes	Integration effort required
Separate platform-independent from platform-specific concerns in models, but preserve links between concerns	Most patterns are platform-independent; platform-specific aspects come in informally via known uses and examples	Dedicated vendor asset level exists in our ADM proposal, decision dependencies can model link between levels	AD levels provide explicit support, patterns can be assigned to these levels
Ease of architectural documentation (authoring)	As patterns are publications, significant effort for pattern author (reviews, writer's workshops); easy to reference	Depends on project setup; no formal authoring and review process (yet)	ADMs less rigorous, more project-specific; patterns more sustainable, long lasting (by definition)
Ease of consumption (usability)	Depends on pattern author; time intense due to in-depth education character	Search capabilities, multiple ordering dimensions to support orientation character	ADMs provide orientation, patterns in-depth coverage of single design concern

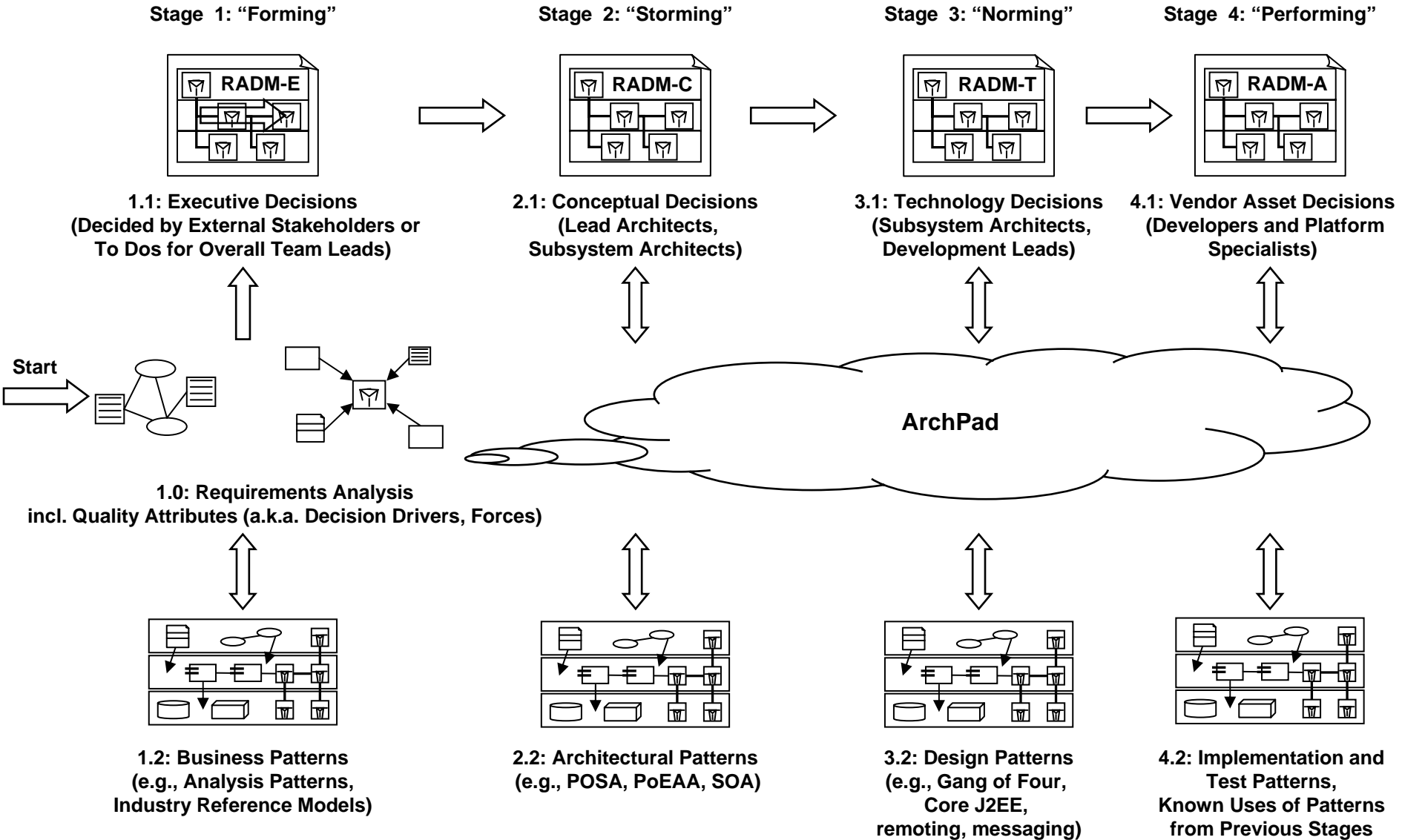
Process alignment?

- Original purpose differs, but concepts overlap
  - Education (patterns) vs. documentation (decisions)... but: pattern selection is a decision!
  - Both concepts have rather weak linkage with software engineering processes/methods

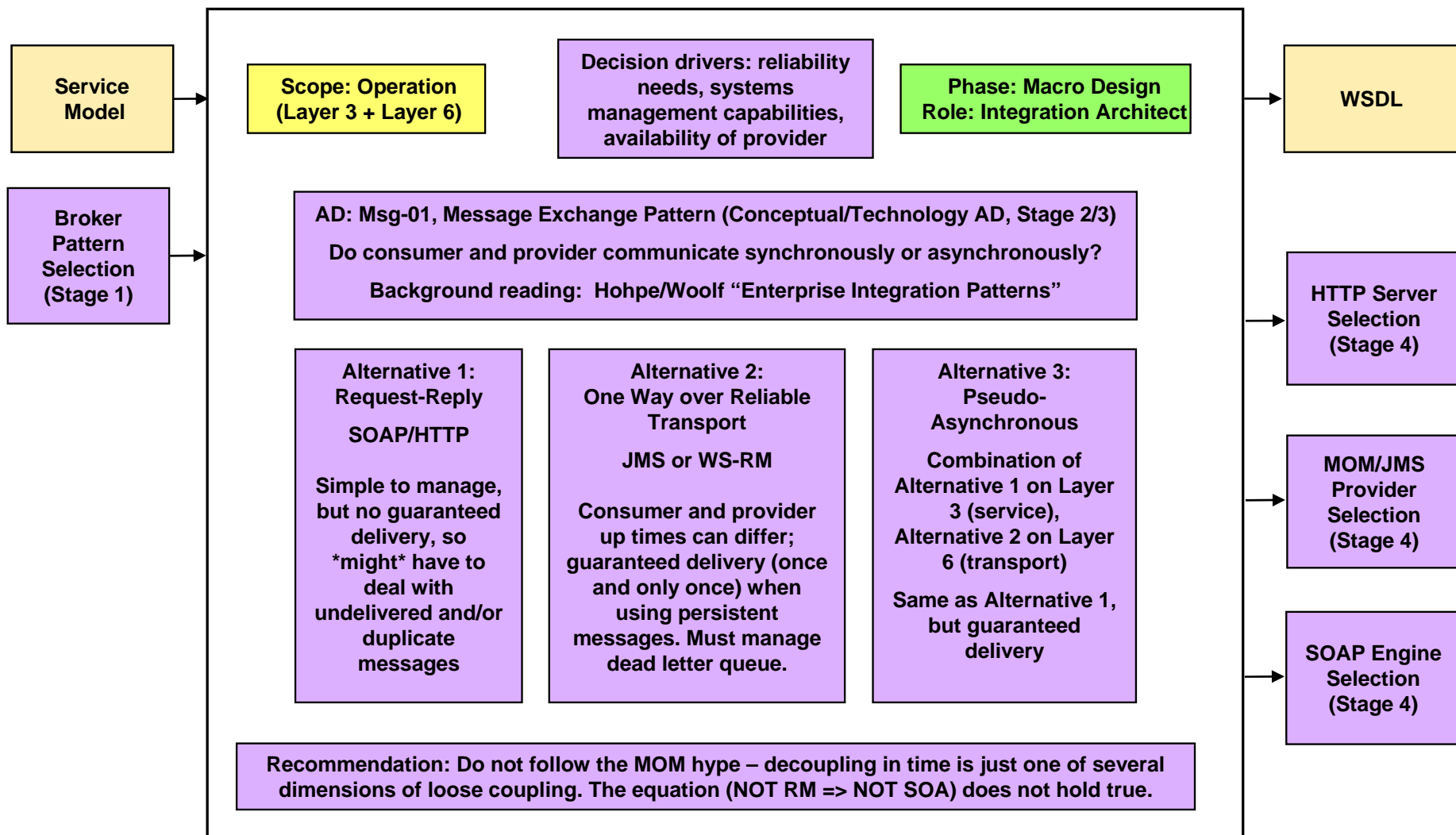
# Agenda

1. Case study
  - Broker-based integration of core banking application components
2. Existing work
  - Architectural patterns and decision capturing
3. **ArchPad walkthrough**
  - **Refinement stages**
  - **Reusable Architectural Decision Model for SOA (RADM for SOA)**
4. Discussion
5. Conclusions and outlook

# Patterns and Decisions Throughout the ArchPad Stages



# Decision (5.1) from Case Study in RADM for SOA



Decision Identification



Decision Making



Decision Enforcement

# Other Decisions in RADM for SOA (300+ Overall)

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	Technologies such as J2EE, .NET, LAMPP, Ruby on Rails
	ED-2 to ED-5: Architectural Principles, Paradigms, Patterns, Processes	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	Front end centric vs. integration centric vs. process centric vs. database centric [13]
	CD-2: Broker Pattern	BROKER [5] (RPC or messaging) vs. direct client/server connectivity
	CD-3: Message Exchange Pattern	Synchronous request-response, POLL OBJECT, RESULT CALLBACK, SYNC WITH SERVER, or FIRE AND FORGET [22]
	CD-4: In and Out Message Parameter Granularity	Deeply structured DOMAIN MODEL [8] elements vs. flat strings
	CD-5: Service Composition Paradigm	Business process engine (following MACRO-MICROFLOW [12] or no separation of flows) vs. custom code; PROCESS-BASED INTEGRATION ARCHITECTURE [12] or not
	CD-6: Presentation Layer Paradigm	Rich client vs. thin client vs. best of both worlds
	CD-7: Conversational State	CLIENT SESSION STATE [8] vs. SERVER SESSION STATE [8] vs. none
	CD-8: Transaction Management	SYSTEM TRANSACTIONS vs. BUSINESS TRANSACTIONS [8]
Stage 3 (RADM-T): Technology Decisions (TDs), dealing with selection of design patterns, technologies	TD-1: Message Exchange Technology and On-The-Wire Protocol [5]	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
	TD-2: Broker Technology	Commercial ESB product vs. custom integration layer
	TD-3: Remoting Patterns	INVOKER, CLIENT PROXY, MARSHALLER, INVOCATION INTERCEPTOR, INVOCATION CONTEXT, use of PROTOCOL PLUG-IN [22] or not
	TD-4: Message Exchange Style and Format	WS-* and SOAP vs. REST and POX/JSON vs. plain TCP/IP and custom strings vs. other
	TD-5: Transport Protocol Binding	HTTP vs. messaging
	TD-6: Service Provider Type and Application Programming Interface (API)	Enterprise Java Bean (EJB) vs. plain Java object vs. other provider in other programming language; JAX-RPC vs. JAX-WS/JAX-B vs. proprietary
	TD-7: Service Provider Component Container Technology	Service Component Architecture (SCA) vs. Java 2 Enterprise Edition (J2EE) vs. Spring vs. Common Object Request Broker Architecture (CORBA) vs. .NET vs. other
	TD-8: Business Process Language	Business Process Execution Language (BPEL), other
	TD-9: Process-based Integration Architecture Design	RULE-BASED DISPATCHER or not; Deployment and Structure of MACROFLOW ENGINES and MICROFLOW ENGINES; CONFIGURABLE ADAPTERS and REPOSITORIES; etc. (see [12])
	TD-10: Presentation Layer Technology	Plain HTML (thin client) vs. portal (thin client) vs. Web 2.0 RIA vs. Eclipse RCP
	TD-11: Presentation Layer Organization	APPLICATION CONTROLLER vs. PAGE/FRONT CONTROLLER [1] vs. HUMAN TASK LIST [18]
	TD-12: Process Layer Interface Granularity	Batch vs. conversational
	TD-13: Presentation/Process Layer Coordination	Pull (presentation leading) vs. push (process leading)
	TD-14: Presentation/Process Layer Protocol	Synchronous RPC API vs. asynchronous messaging
	TD-15: Activation Strategy Patterns	STATIC INSTANCES, PER-REQUEST INSTANCES, or CLIENT-DEPENDENT INSTANCES (all from [22])
TD-16: Resource Management Patterns	LEASING, POOLING, LAZY ACQUISITION, and PASSIVATION (all from [22])	
TD-17: Session Management	Client (e.g., full state in cookie) vs. presentation layer (HTTP session) vs. process layer (BPEL correlation) vs. backend (database)	
TD-18: Compensation Scheme	BPEL vs. vendor-specific spheres vs. custom logic	
Stage 4 (RADM-A): Vendor Asset Decisions (VDs), dealing with product selection and configuration	VD-1: ESB Product	E.g. IBM WebSphere ESB, Progress Sonic ESB, Mule
	VD-2: SOAP Message Exchange Engine	Apache Axis, Codehaus XFire, vendor engines such as IBM WebSphere engine, WSIF and Apache SOAP
	VD-3: Service Provider Container	J2EE application server with(out) EJB support, SCA container such as WebSphereProcess Server
	VD-4: Service Provider Sourcing	Make or buy; adapt or refactor existing asset
	VD-5: Business Process Engine Vendor	E.g. IBM WebSphere Process Server, Oracle BPEL Process Manager), open source (ActiveBPEL)
	VD-6: Presentation Layer Application Server	E.g. various servlet engines and portal servers, application wiki engines
	VD-7: Platform-Specific Transaction Attribute	E.g. various SCA qualifiers and EJB attributes

# Agenda

1. Case study
  - Broker-based integration of core banking application components
2. Existing work
  - Architectural patterns and decision capturing
3. ArchPad walkthrough
  - Refinement stages
  - Reusable Architectural Decision Model for SOA (RADM for SOA)
4. **Discussion**
  - **Tool support**
  - **Validation**
5. Conclusions and outlook

# Tool Support: Decision Modeling Application Wiki AD<sub>kwik</sub>

**QED**
ADkwik : 
You are SoadAdmin [\[settings\]](#) [\[logout\]](#)

text  
ADkwik | QEDWiki | User Settings | Help

**WorkSpace**

[My Pages](#) | [Team Pages](#) | [History](#)

- [-] Driver096Master12
  - [-] AdHistory
  - [-] AdsByRole
  - [-] BusinessExecutiveLev
  - [-] ConceptualLevel
    - [-] EnterpriseApplicati
    - [-] Layer1ResourceRt
    - [-] Layer2ComponentI
    - [-] Layer3ServiceRea
    - [-] Layer4ProcessRea
    - [-] Layer5ApplicationF
    - [-] Layer6IntegrationF
    - [-] Layer7SecurityRea
      - [-] BusinessSecur
      - [-] ItsSecurityServic
        - [-] Its01Identit
        - [-] Its02EndUs
        - [-] Its03Servic
        - [-] Its04EndUs
        - [-] Its05Servic
        - [-] Its06Confid
        - [-] Its07Integr
        - [-] Its08Auditir
        - [-] Its09NonPr
        - [-] Its10Servic
        - [-] Its11Share
        - [-] Seco01Threat
      - [-] SecurityPolicyl
    - [-] Layer8Informatio
    - [-] OperationalModelir
    - [-] TechnologyLevel
    - [-] VendorAssetLevel
  - [-] Eoin Test
  - [-] GtsSoalsraLayer234
  - [-] GtsSoalsraLayer6
  - [-] GtsSoalsraLayer7

## Its-05 ServiceAndEndUserAuthorization

**Scope:** Global
**Phase:** Solution outline
**Role:** Lead architect

**Problem Statement**

Will authorization be required by the service and what level of authorization is required?

Most organizations provide differing classification levels to protect information from unauthorized access. One method for providing the restricted access is to permit or deny access based upon the end-user or their characteristics. In addition, many organizations also permit particular end-users or roles to execute business transactions up to certain financial values. Authorization decisions are used to permit or deny access to intellectual capital and proprietary information to the business. With the legal and regulatory environment in place today authorization is typically required to comply to these laws and regulations.

**Decision Drivers**

Unauthorized access may be permitted to information including PII.  
Unauthorized execution of business transaction may be permitted.

A lack of appropriate authorization can affect the compliance of an organization to such legal or regulatory requirements such as HIPAA, SOX or Safe Harbor. The organization itself must determine its requirement from a legal standpoint and IBM should not provide legal advice in these areas.

Assumptions: Appropriate identification and characteristics are provided for the end user or the service, component or application.

**Alternatives**

- 1 No authorization required
- 2 Coarse grained authorization
- 3 Fine grained authorization re
- 4 Not applicable

**No authorization required**

**Description**

**Pros**

**Go To**

[Recommendation](#)

[Enforcement Recommendation](#)

**Background**

See provided reference information.

**Relationships**

- ◆ influences [AccessManagement](#)
- ◆ is influenced by [PolicyDecisionPoint](#)
- ◆ is influenced by [PolicyEnforcementPoint](#)

**Lifecycle Information**

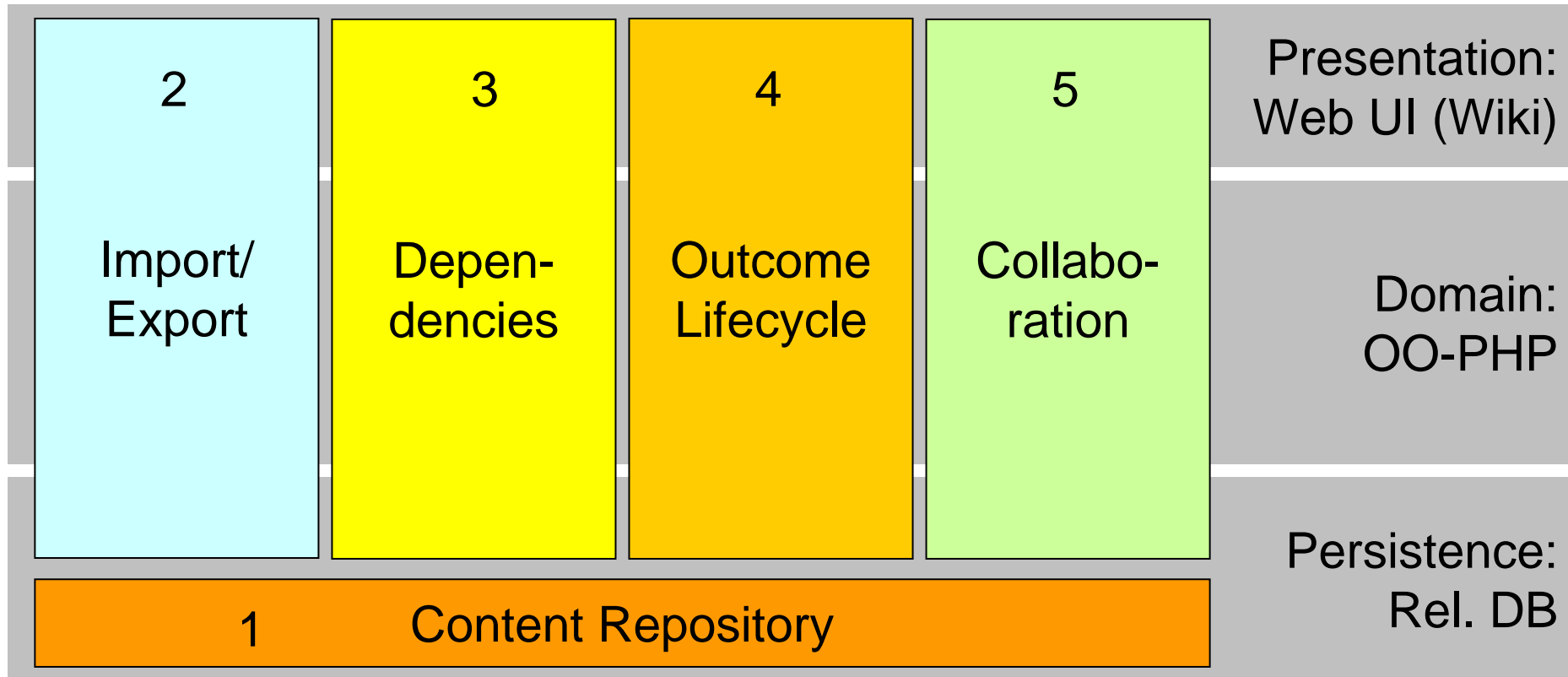
This AD was identified by [AwbUser](#) on 2007-10-01 11:58:10

Last modification of this AD by on 2007-10-01 11:58:10

[Show/Hide editorial information](#)

[Show revisions of this AD](#)

# Architecture of Collaboration Platform AD<sub>kwik</sub>



Schuster N., Zimmermann O., Pautasso C., **ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering**. In: Proc. of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007)

## Validation via Case Studies and Action Research

- Applied ArchPad to own SOA projects retrospectively
  - Case study presented in this paper and BPEL-based telco order management
  - Revised and integrated book content:
    - “Perspectives on Web services”, (Zimmermann/Tomlinson/Peuser)
    - “Remoting Patterns” (Voelter/Kircher/Zdun)
- Method, content, and tool used on industry projects
  - Design method/process support: five industry engagements in several industries (telecommunications, government, retail and distribution)
  - Education, review, and governance workshops (professional services)
- Hosting company-internal instance of decision knowledge wiki
  - 100 registered  $AD_{\text{kwik}}$  users
  - Scheduled for release via IBM alphaWorks emerging technology site

# Agenda

1. Case study
  - Broker-based integration of core banking application components
2. Existing work
  - Architectural patterns and decision capturing
3. ArchPad walkthrough
  - Reusable decision model
  - Reusable Architectural Decision Model for SOA (RADM for SOA)
4. Discussion
  - Tool support
  - Validation
5. **Conclusions and outlook**

## Summary of Key Concepts in ArchPad

- **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 decisions 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

## Advanced Usage Scenarios and Future Research

- **Project planning and health checking**
  - Work breakdown structures can be created from a 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-specific instance of the SOA RADM, consisting of a subset of decisions and alternatives to give freedom of choice to individual project teams without sacrificing overall architectural integrity

# Thank You!

- Questions?
- Comments?
- Architectural decision knowledge to share?
  - Consider authoring a practitioner report for OOPSLA 2008 (Deadline: March 19th)

[http://www.oopsla.org/oopsla2008/cfp\\_development.html](http://www.oopsla.org/oopsla2008/cfp_development.html)

- “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

<http://soadecisions.org>

