Using the Zephyr Extensibility Model to Model Architectural Decisions

Dr. Olaf Zimmermann
IBM Research – Zurich
olz@zurich.ibm.com

Agenda

- **Motivation and case study**
- Usage scenarios for architectural decision modeling
- Scenario 1: After-the-Fact Decision Capturing
- Scenario 2: Active Method Guidance
- Scenario 3: Cross-Role Collaboration and Tool Integration
- Emerging tool support (demo)
- Discussion and summary
Architectural Decisions: Current Trends

- Decision capturing support soon to be mandatory in architecture descriptions conforming to standard
- Popular text books and mature methods promote the concept
- Practitioner demand
- Research prototype tools

ISO/IEC WD4 42010
IEEE P42010/D6

What are Architectural Decisions? Why Bother?

- “The design decisions that are costly to change” (Grady Booch, 2009)
- Our definition (from http://soadecisions.org/soad.htm#wicsa):
  “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”
### 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.**
Agenda

- Motivation and case study
- **Usage scenarios for architectural decision modeling**
  - Scenario 1: After-the-Fact Decision Capturing
  - Scenario 2: Active Method Guidance
  - Scenario 3: Cross-Role Collaboration and Tool Integration
- Emerging tool support (demo)
- 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)

Agenda

- Motivation and case study
- Usage scenarios for architectural decision modeling
  - **Scenario 1: After-the-Fact Decision Capturing**
  - Scenario 2: Active Method Guidance
  - Scenario 3: Cross-Roe Collaboration and Tool Integration
- Emerging tool support (demo)
- Discussion and summary
Scenario 1 (Documentation): A Story from the Case Study (2004-2005)

"According to our method, I have to create a work product called 'Architectural Decisions'. It is supposed to record the key decisions made on the order management SOA project, and the rationale behind them."

"This will help me survive the upcoming technical quality assurance review requested by the world-wide SOA subject matter expert they will be flying in shortly. And hopefully it will stop these endless and pointless discussions 'why SOA' that our developers have been raising since the project start."

So let's create an architectural decisions work product:

1. We decided for process-enabled SOA because the business scenario is a long running, multi channel, multi actor scenario with heavy resource coordination requirements. There are 20 backend systems, only a few of which are transactional.
2. We decided for BPEL because it is a standardized workflow language with emerging tool and runtime support. The out-of-the-box support for compensation will help us meet the coordination requirements.
3. We decided for IBM WebSphere products because this is the preferred vendor of the client. The IBM BPEL engine can easily handle the required load. BPEL is new (July 2004), so lab advocacy will be needed.

This is getting tedious in MS Word although I have all insight I need. Will capture the next few ones later. I really need a tool that supports decision capturing and sharing."

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 AD ID 3</td>
<td></td>
<td></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 Decision Making in Context – Decision Drivers

Technical Drivers

- Reference Information (Industry Models, Enterprise Architecture)
- Functional Requirements (BPM, Use Cases, User Stories)
- Existing Systems (Capabilities, Limitations)
- Non-Functional Requirements (incl. Software Quality Attributes)

Nontechnical Drivers

- Past Arch. Decisions
- Architecture Design Work
- Project Budget and Timelines
- Skills, Experience, Preferences in Team

Stakeholder Goals (Existing Practices, Strategic Directions)

Phase (Ph.) n-1

- 4+1 VPs

Phase (Ph.) n

- Future Arch. Decisions

Ph. n+1

- 4+1 VPs


Valid Justifications… and Counter Examples

- **Convincing rationale:**
  - Direct link to (non-)functional requirements, quality attributes in particular
  - Positive experience on previous project
  - Existing skills, license agreements

- **Poor justifications:**
  - Market momentum (technology or vendor push)
  - Only one alternative known/considered
  - Keep CVs of team members current

Agenda

- Motivation and case study
- Usage scenarios for architectural decision modeling
  - Scenario 1: After-the-Fact Decision Capturing
  - Scenario 2: Active Method Guidance
    - Scenario 3: Cross-role Collaboration and Tool Integration
- Emerging tool support (demo)
- Discussion and summary

Scenario 2 (Guidance): The Case Study Continues

"This is going to be an interesting assignment. According to the PPT charts that outline the architecture, the team decided for SOA but does not know much about the design issues that it has to resolve now. These issues include service granularity, transaction boundaries, message exchange patterns."

"Are we the first IBM client that implements a process-based SOA with BPEL? If not, I’d love to have a look at the architecture design documentation from the previous projects. Which detailed design issues did the team encounter? Which patterns and technology alternatives did they consider? Why did they decide for the ones they eventually picked? Did they work? Do they have other best practices to share?"

"This decision-centric approach to knowledge sharing, architecture design, and reviewing worked really well. In each workshop, we looked at selected issues to be resolved, which apparently came from some reusable asset (called guidance model if I remember correctly). We based the selection (called tailoring) on various factors such as project scope, risk and cost, and amount of experience in the local team. The issues come with alternatives that are known to have worked on previous projects, and with links to additional issues to be investigated. Without this knowledge base, I would have had no idea that I have to worry about the system transaction boundaries inside the business processes and the underlying Web services, let alone where in the IBM BPEL engine to configure these settings. Same for my developers by the way."

"If we succeed with this project, I will harvest and contribute to the guidance model the additional issues that we encountered, the alternatives we chose, and the rationale for these decisions."
From AD Documentation to Active Method Guidance

600+ Decisions in IBM SOA Decision Guidance Model (SOAD)

Architectural Style (SOA or other?)

Service Composition Paradigm (Processes? Classes?)

Message Exchange Pattern (Request-Reply? One Way?)

Transaction Qualifiers in SCA? Operations per SOA? Part types? WSDL or others?

Transaction Boundaries? Service Granularity? Message Confidentiality?

Decision Made/Alternative Selected

Architectural Decision Issue (with Alternatives)

Conceptual Level

Service Composition Paradigm

Process-Enabled SOA

Message Exchange Pattern

Synchronous Request-Reply

Technology Level

Process-Enabled SOA

for each service

Transaction Qualifiers in SCA? Operations per SOA? Part types? WSDL or others?

Transaction Boundaries? Service Granularity? Message Confidentiality?

Vendor Asset Level

Workflow Language (BPEL? Other?)

Transport Protocol (SOAP over HTTP?)

Synchronous Request-Reply

BPEL 2.0

SOAP/HTTP 1.1

BPEL Engine (IBM WPS? Other?)

SOAP Engine (Apache Axis2?)

For each process

Process-Enabled SOA

Service Composition Paradigm

Problem and criteria

Guidance Model

Decisions Required and Candidate Solutions

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

Our extension

Decision Model

Decisions Actually Made on Projects

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

Potential solutions with pros and cons

“…We decided for the MVC alternative to resolve the web page flow issue because we gained positive experience with it on many similar projects.”

Chosen solution and justification

UMF template (ART 0513/ARC 100)
### Integration Layer Design Issue: Message Exchange Pattern

**Issue:** Message Exchange Pattern (Conceptual/Technology Level)

- **Problem Statement:** Do consumer and provider communicate synchronously or asynchronously?
  - Background reading: Hohpe/Woolf “Enterprise Integration Patterns”

**Scope:** Operation

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

**Phase:** Macro Design

**Role:** Integration Architect

**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.

**Alternative 1:** Request-Reply

- **Name:** SOAP/HTTP
- **Description:** Simple to manage, but no guaranteed delivery; so “might” have to deal with undelivered and/or duplicate messages

**Pros:**
- Simple to manage

**Cons:**
- No guaranteed delivery

**Alternative 2:** One Way over Reliable Transport

- **Name:** JMS or WS-RM
- **Description:** Consumer and provider up times can differ; guaranteed delivery (once and only once) when using persistent messages. Must manage dead letter queue.

**Integration Style**

- **MOM:** Message-Oriented Middleware
- **JMS:** Java Messaging Service
- **WS:** Web Services
- **RM:** Reliable Messaging

**Integration Technology**

- **Alternative 2 on underlying transport layer**
- **Alternative 3 on application integration layer**

**Alternative 3:** Pseudo-Asynchronous

- **Description:** Combination of Alternative 1 on application integration layer, Alternative 2 on underlying transport layer, same as Alternative 1, but guaranteed delivery

**Service Model**

- **WSDL**
Issues Dealing with System Transaction Boundaries 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.

**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.

**Alternatives:**
- **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.

**Scope:** Activity/Operation
**Decision drivers:** Resource protection needs, data currency, performance.
**Phase:** Macro Design
**Role:** Application Architect
**Compensation Patterns**
**SCA qualifiers, BPEL server configuration, WS-AT usage**

---

Issues Dealing with Service Granularity Topic

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

**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.

**Alternatives:**
- **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 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.

**Scope:** Service Operation
**Role:** Service Modeler
**Phase:** Macro Design
**Role:** Application Architect
**WSIDL, XML Schema Contracts**
**XML Profiling**
**Out Message Granularity**
**Operation To Service Grouping**
**Component Wrapping**
**WSIDL, XSD Editor Selection**
**Conceptual/Technology Issues about Encryption**

**Issue: Encryption Layer and Technology (Technology Level)**
Where and how should service messages be encrypted?
See book by Weerawarana et al. “Web Services Platform Architecture”

**Alternative 1: Transport-Layer Encryption**
SSL/TLS (HTTPS)
Well adopted on the Web, but no end-to-end security.

**Alternative 2: Message-Layer Encryption**
XML digital signatures (WSSE)
Allows end-to-end security across intermediaries, but is expensive.

**Recommendation:** Stick to SSL/TLS unless message-level security required due to business requirements; use HW accelerator when deciding for WSSE.

**Vendor Asset Level Clustering Issues**

**Issue: WebSphereClustering (Vendor Asset Level)**
Should WebSphere Application Server be deployed in a standalone or in a high availability configuration?
See WebSphere Info Center and platform patterns documents

**Alternative 1: Single Server**
No node manager
Easy to manage, but service consumers can not be served when WAS instance or server HW are down. Server failure leads to loss of session data.

**Alternative 2: Cluster**
Enable node manager, have multiple nodes that can take over from each other
Higher availability from consumers perspective, hot/cold session takeover required. Systems management needs.

**Recommendation:** Stick to Alternative 1 unless business requirements and NFRs force you to use Alternative 2. Design services to be able to run in clustered configuration (e.g., store only serializable Java objects and only a few KB in HTTP session object).
SOA Design Issues Organized by Levels and Layers

<table>
<thead>
<tr>
<th>Business Executive Level</th>
<th>Application Architecture</th>
<th>Infrastructure Architecture</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Requirements Decisions</td>
<td>Executive Decisions</td>
<td>Operational VP Conceptual Decisions</td>
<td>Platform Preferences</td>
</tr>
</tbody>
</table>

Conceptual Level

- Functional VP: Conceptual Decisions
- Consumer Layer
- Process Layer
- Service Layer
- Component Layer
- Resource Layer
- Message Exchange Pattern

Technology Level

- Functional VP: Technology Decisions
- Consumer Layer
- Process Layer
- Service Layer
- Component Layer
- Resource Layer
- Operational VP Technology Decisions
- Transport Protocol

Vendor Asset Level

- Functional VP: Vendor/Asset Decisions
- Consumer Layer
- Process Layer
- Service Layer
- Component Layer
- Resource Layer
- Operational VP: Vendor/Asset Decisions
- DataPower Configuration

Agenda

- Motivation
- Usage scenarios for architectural decision modeling
- Scenario 1: After-the-Fact Decision Capturing
- Scenario 2: Active Method Guidance
- **Scenario 3: Cross-Role Collaboration and Tool Integration**
- Emerging tool support (demo)
- Discussion and summary
Scenario 3 (Cross Role): Desired State in the Case Study

Solution Architect

"I am not supposed to worry about technical details of the SOA solution too much, but quite a few architecturally significant requirements such as confidentiality and 24x7 availability have been stated. I will mark them as such. I’ll be interested to trace whether and how the architecture satisfies them."

"The requirements people are using this new tool which supports decision model tailoring and outcome instance creation. So I already know that two business processes with about 100 Web services activities have to be realized. We’ll have to use HTTPS to connect Web channels to the system. We’ll also pick a clustered topology. This really accelerates the architecture design work."

Business Analyst

"In recent times, developers have criticized my architecture design as too high level and too difficult to implement. I am not always sure how to enforce my architectural decisions so that the implementation reflects them properly. With the new collaboration tooling, I can track them as design work items and link them to development tasks. Some of the enforcement can even be automated by code generators."

"The new collaboration tooling has its goods and its bads from my perspective. On the positive side, I can always find somebody who knows about the design goals and decisions made so far. And we all are informed about project health. However, the architects are much closer involved now; I can no longer create and hide my own designs. And some of the routine coding and configuration steps were taken over by model transformations."

Developer

"The new collaboration tooling has its goods and its bads from my perspective. On the positive side, I can always find somebody who knows about the design goals and decisions made so far. And we all are informed about project health. However, the architects are much closer involved now; I can no longer create and hide my own designs. And some of the routine coding and configuration steps were taken over by model transformations."

From Requirements to Architectural Decisions

<table>
<thead>
<tr>
<th>Functional requirements</th>
<th>Reference information (industry models, enterprise architecture)</th>
<th>Non-functional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes to be automated and integrated</td>
<td>Existing systems (capabilities, limitations)</td>
<td>Software quality attributes</td>
</tr>
<tr>
<td>Services to be built or reused</td>
<td></td>
<td>Service level agreements</td>
</tr>
<tr>
<td>Data to be managed across life cycle</td>
<td></td>
<td>Regulatory constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data management</td>
</tr>
</tbody>
</table>

How to build the processes/services and satisfy their quality requirements and constraints?

Architectural Decisions

Recurring decisions carry best practices regarding pattern adoption
Active Method Guidance (Meta Issue Level): Handshakes between Design Models and Decision Models

- Pattern Selection Decision → Pattern Adoption Decisions → Chosen Alternative → Enforcing DME
- Technology Selection Decision → Technology Profiling Decisions → Chosen Alternative → Enforcing DME
- Asset Selection Decision → Asset Configuration Decisions → Chosen Alternative → Enforcing DME

(Meta) Issue Identification in High-Level Component Models

Legend:
- Design Model Element (Functional, Operational)
- Meta Issue (Decision Required)
- Transition to Next Realization Level (e.g., Conceptual to Specified to Implementation Component Model and Conceptual to Specified to Physical Operational Model?)

- Each concrete component and connector in chosen reference architecture yields concrete issues derived from meta issues
- E.g. Web service component, WebSphere topology unit
Conceptual Operational Model (COM) and Rational Software Architect Topology: Decisions Made/Required

Decision made: Model a single location for entire order management system, rationale: physical separation/mirroring not required

Decision made: Host four Deployment Units (DUs), rationale: UML component layering, best practice stated in SOA Infrastructure Reference Architecture

Decision made: Let this location host two identical conceptual servers, rationale: medium to advanced availability and performance NFRs; requests must not be lost.

Decisions now required (identified in this model):
- Transition to Specified OM, specification of realization links
- Enterprise Architect
- Solution Architect

Scenario 3: Cross Role Collaboration (Continued)

“The order management project was a big success. The team learned a lot about the design of process-driven SOA. We plan to apply this architectural style on several new projects; hence, we would like to establish architectural principles such as loose coupling and share best practices throughout the company. To make this actionable, I will upgrade the decision model from the project to a company-wide guidance model. The issues in the guidance model have to be resolved explicitly on each and every solution development project. If any alternatives are chosen that are not in the guidance model, enterprise-level approval is required.”

“I was initially skeptic about this company-wide guidance model: We have enough policies and principles already. And in the end, the business requirements overrule everything. Finally, who will populate the model and keep it up to date? None of the knowledge management approaches we have tried worked in the long term.”

“I have to admit that they managed to find a balance between freedom-of-choice and freedom-from-choice. The issues and alternatives in the model are all relevant, easy to locate and to digest, but far away from being trivial. With the new tooling, it is really simple to submit desclets from the project to the knowledge harvesting process.”
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

Agenda

- Motivation
- Usage scenarios for architectural decision modeling
- Scenario 1: After-the-Fact Decision Capturing
- Scenario 2: Active Method Guidance
- Scenario 3: Cross-Role Collaboration and Tool Integration
- Emerging tool support (demo)
- Discussion and summary
Architectural Decisions Knowledge Tools Project (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

- Motivation
- Usage scenarios for architectural decision modeling
  - Scenario 1: After-the-Fact Decision Capturing
  - Scenario 2: Active Method Guidance
  - Scenario 3: Cross-Role Collaboration and Tool Integration
- Emerging tool support (demo)
- Discussion and summary
Value of Architectural Decision Modeling

- Architectural decision making is a key responsibility of IT architects which is often underestimated and underrepresented in existing methods and tools.
- In SOA design, 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).
- Prototypical tool support for decision modeling with reuse is available.
- 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?

Summary and Discussion