Model-Driven Software Engineering:
Models in Software Architecture Design

Dr. Olaf Zimmermann, Executive IT Architect (olz@zurich.ibm.com)
Learning Objectives

- Understand selected software architecture concepts, and see them being applied to practical examples and industrial case studies
  - Software quality attributes
  - Architectural patterns (for application development and integration)
  - Architectural decisions

- Understand role of models in software architecture design, and be able to consume and create such models
  - Motivation for modeling
  - Types of models
  - Languages and notations
  - Tool support
  - Practical hints
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- Case study: order management in telecommunications
- Models in software architecture design: UMF and UML
- Concluding thoughts
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- Case study: order management in telecommunications
- Models in software architecture design: UMF and UML
- Concluding thoughts
A First Example: Core Banking SOA with Web Services

Platform independent

IBM WebSphere®
(pSeries)

Dynamic Interface

Java Client

.NET Client

Browser

Office

Web Services Adapter Layer

Java™ API (Dynamic Interface)

Java Backend Connectors (IBM WebSphere MQ, CICS®)

Access Layer

Business Function

Database

(IBM DB2®)

Repository

Documentation

generate

WSDL

generate

Generate
Architecture and Software Architecture Defined

Most definitions indicate that an architecture is concerned with both structure and behavior, is concerned with significant elements only, may conform to an architectural style such as SOA, is influenced by its environment, and embodies important decisions [Eel11].

Three commonly used definitions are:

- Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IEEE1471]
- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [BCK03]
- Software architecture as a set of architectural design decisions. [JB05]

Architecture vs. IT architecture vs. software architecture – all in scope in this lecture
Software Architecture Fundamentals

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

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

- **Foundations** date back to late 1990s (industry and academia)
  - Bass/Clements/Kazman “Software Architecture in Practice” [BCK03]
  - Shaw/Garlan “Software Architecture – Perspectives on an Emerging Discipline” [SG96]
  - Buschmann et al. “Patterns of Software Architecture” (POSA) [BMR+96]
Software 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 example deal with QAs tacitly or explicitly:
  - QA “Accuracy”: orders must not be lost, resource reservations must be undone
  - QA “Efficiency” (performance): sub-second response times specified
  - QA “Interoperability”: multiple platforms to be supported

- 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/step)
Architectural Patterns

- Proven, common solution to known and recurring architecture design problem:
  - 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!)
  - See e.g. http://hillside.net/index.php/ag-template and http://hillside.net/index.php/a-pattern-language-for-pattern-writing for pattern writing support and advice

- Top-level pattern in first example and in SOA case study:
  - “Layers” pattern from [BMR+96] 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)
No industry consensus on SOA principles and patterns yet
Each author defines his/her own – many terminology mismatches
A Closer Look at the 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.”

Source: [HW04] and http://www.eaipatterns.com/ProcessManager.html

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

- Layers, Pipes & Filters, Broker – POSA book series [BMR+96]
- Patterns of Enterprise Application Architecture such as Model-View Controller, Domain Model, Active Record [Fow03]
- Domain-Driven Design patterns such as Bounded Context [Eva03]
- Enterprise Integration Patterns such as Messaging, Channel [HW04]
- Adapter, Observer and other design patterns [GHJ+95]

“Handbook of Software Architecture” collects such architectural patterns
  – Compiled by Grady Booch (one of the original authors of UML):
    http://www.handbookofsoftwarearchitecture.com/index.jsp?page=Main
    (registration/login required)
Architectural Decisions

- Capture the *rationale* justifying a design, in addition to design itself (answers to “why” questions)

- Example:
  - “We selected the Layers pattern to make the core banking SOA future proof, e.g., to be able to add user channels in a flexible manner”

- Practical challenges:
  - Retrospective decision capturing takes time and does not yield sufficient benefits
  - Relation to other architectural concepts and viewpoints (quality attributes, patterns) not understood well and not supported in methods and tools
Input and output of artifacts are documents or models

Widely recognized and/or commonly practices methods:
- ADD (SEI CMU), Siemens’ 4 Views, RUP (Rational/IBM), BAPO/CAFCR (Philips), ASC/ARES (Nokia) [HKN+07]
- IBM Unified Method Framework (UMF), e.g. Application Development (AD) 2.0 [EC10]
- Object-Oriented Analysis and Design (OOAD), e.g. codified in Open Unified Process (Open UP) [OUP]. Sample technique (a.k.a. method guidance):
  Focus on the architecture early to minimize risks and organize development
- Domain-specific methods, e.g. for SOA design
The architect (a.k.a. technical lead) plays many roles.

- Project Management Software
  - Project Plan incl. Work Breakdown Structure
  - Executive ADs (Project Scoping)
- Method Browser
  - Roles
  - Process Phases/Activities/Tasks
  - Artifacts
  - Technique Papers
  - Sample Content
- Office Suite
  - NFRs
  - Decision Log
- Analysis Modeling Environment
  - Analysis-Phase BPM
  - Business Processes
  - Business Activities
- Design Modeling Environment
  - Design Model (e.g., UML)
  - Conceptual Workflows
  - Service Contracts
  - Tacit ADs
- Development Environment
  - BPEL
  - WSDL
  - Java
  - Issue List (ADs)
  - NFRs
- Traceability Management Tool
  - Setup, review
  - NFRs

Software Architect

- Consult, tailor
- Create, review
- Review
- Create
- Maintain
- Setup, review

Source: [Zim09]
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- Case study: order management in telecommunications
- Models in software architecture design: UMF and UML
- Concluding thoughts
What is a Model (Engineering Definition)?

- From 1st lecture (MDSE fundamentals): “A model is an abstraction of something for the purpose of understanding it before building it” [RBT+91]

- Compare with definition by Bran Selic (an author of UML2): “abstraction is an intellectual defense mechanism for coping with overwhelming complexity” [Sel11]:
  - “Selective reduction of information of a system which preserves its salient properties relative to a given set of concerns”
  - “Refinement is the inverse process”

- In engineering, abstractions are gained through modeling. An Engineering Model is [Sel11]:
  - “A selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns”
  - “We don’t see everything at once”
  - “What we do see is adjusted”
Why do Architects Model?

- Modeling forces you **think** deeply about the requirements and the emerging design – and **understand** and **anticipate** their ramifications (consequences)
  - Reason about choices
  - Trigger creativity, force certain decisions (e.g. type and cardinality of associations)
  - Abstract (leave out unnecessary details)

- Models **communicate** (an architectural vision/intent)
  - Within team of architects (same or different areas of specialization/expertise)
  - Within project team (architects, developers, testers, platform specialists, administrators)
  - With external stakeholders (sponsors, end users, maintenance teams)

- Models **document** a design and allow to automate tasks, e.g. test case creation
  - Simulations (models often are cheaper/less risky to build than the real system)
  - Automation of routine tasks

Compare with Bran Selic’s view:
“Model to understand, to predict, to communicate, to specify” [Sel11]

*Do not model without a purpose and a target audience in mind!*
Selected Models as UMF Work Products (Artifacts)

Global Viewpoint
- Use Case Model (UCM)
- Non-Functional Requirements (NFRs)
- Architecture Overview Diagram (AOD)

Logical Viewpoint
- Component Model (CM) L1
- Component Model (CM) L2
- Component Model (CM) L3

Physical Viewpoint
- Operational Model (OM) L1 (Conceptual/ALOM)
- Operational Model (OM) L2 (Specified/LOM)
- Operational Model (OM) L3 (Physical)

Adapted from: [ZKP09]
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- **Case study: order management in telecommunications**
- Models in software architecture design: UMF and UML
- Concluding thoughts
“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

VSP 1 (Browser)
- Move
- Create

VSP 2 (Other System via Web Service)
- Move
- Create

Internal Channel for T Staff
- Move
- Create

T

Order Management System

Customer Database

Telephony Network

Billing System
Functional Requirements

PSTN – Public Switched Telephone Network

VSP 1 (Browser)

Move
Create

VSP 2 (Other System via Web Service)

Move
Create

Internal Channel for T Staff

Move
Create


About 20 steps per process, taking up to 24 hours to complete. Steps include:

1. Address validation – complex and requiring several user interactions
2. Resource reservation, e.g. copper transmission path, telephone number

Customer Database
Telephony Network
Billing System
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.
Solution: **Service-Oriented Architecture (SOA) with Process & Service Layers**

- **Process Layer**:
  - "Create" Process
  - "Move" Process

- **Service Layer**:
  - "Validation" Service
  - "Pricing" Service
  - "Transmission Reservation" Service

**Components**:

- **Customer Database**
- **Telephony Network**
- **Billing System**

**Interface**:

- **VSP 1 (Browser)**
  - Move
  - Create
- **VSP 2 (Other System via Web Service)**
  - Move
  - Create
- **Internal Channel for T Staff**
  - Move
  - Create

**Integration**:

- **T**
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- Case study: order management in telecommunications
- Models in software architecture design: UMF and UML
- Concluding thoughts
The term “model” is overloaded – two primary meanings exist (MDSE vs. method)
- Instance of a metamodel defining a modeling language (e.g., UML model)
- Type of artifact in a method and instance thereof (e.g., UMF component model)

Several UML profiles for architects exist
- IBM Architecture Description Standard (ADS)
- IBM Rational UML Profile-Based Integrated Architecture (UPIA) V7.5.2
- Profiles supporting (enterprise) architecture frameworks such as DoDAF and MoDAF

UML concepts frequently used in architecture design:
- Use cases, stereotyped class diagrams (e.g., domains models, logical/functional architecture viewpoint)
- Sequence diagrams (for component interactions, system walkthroughs), and state machines (e.g., protocol specifications)

Architecture Description Languages increasingly popular, e.g. ArchiMate
(subgenre of domain-specific languages)
Sample Architecture: 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

- **Note**: Architecture Overview Diagram (AOD) – drawn in an ADL, not in UML!
  - Proprietary ADL, first defined in an article on ADS [YRS+99]
  - Supported by IBM Rational Software Architect
UMF Use Case Diagram and Component Model

- Traceability to functional requirements and NFRs required (a non-trivial task in practice)
- Dynamic view also to be provided (component interaction diagrams)
- Sunny day vs. rainy day design (error handling!)

**Note:** Both UMF deliverables contain a UML model/diagrams – and more!
- Open UP/RUP give similar advice
- E.g. Use case template to be filled out (structured text)
  - Preconditions, post conditions, scenario walkthroughs, etc.
UMF Operational Model

- **Note:** Like in the AOD, the model elements (i.e., location node, deployment units, relations) and are defined in an ADL [YRS+99], not in UML.
- Design rationale is attached in the form of unstructured comments (free form).
Important Model Types (Work Products/Artifacts) in UMF

- CIM (Analysis)
  - Process Definition (+Business Use Cases)
  - Current Process Assessment (+Reference Model Alignment)
  - Current IT Environment
  - Architectural Template
  - Nonfunctional Requirements

- PIM 0
  - Process Definition (Design Model)
  - System Context
  - Architecture Overview Diagram

- PIM 1
  - Use Case Model 1 (external)
  - Service Model 1 (incl. WSDL-i)
  - Component Model 1 (UML outline)
  - Operational Model 1 (conceptual)

- PIM 2 / PSM 1 (hybrid)
  - Use Case Model 2 (+internals)
  - Service Model 2 (incl. WSDL-b)
  - Component Model 2 (UML refined)
  - Operational Model 2 (WPS, specified)

- PSM 2
  - Deployment Units (SCA)
  - Component Model 3 (WPS, JEE)
  - Operational Model 3 (WPS, physical)

Analysis to Design (manual) High to Low Level Design (manual, tool supported) Design to Implementation (possibly automated)
More Information on Architecture Modeling in UMF/UML

- IBM Architectural Thinking class (subset also available in lecture form) [Eel11]
- IBM Architecture Description Standard (ADS) [ZRS+99]
- “The Software Architecting Process” by Eeles/Cripps is a definite reference [EC10]:
  - http://www.architecting.co.uk/index.php
- Also see Unified Process (an Open UP is available from Eclipse) [OUP]:
  - http://epf.eclipse.org/wikis/openup/

- Many commercial UML tools, e.g. IBM Rational Software Architect and Design Manager are available, as well as open source assets
- Vision stencils (for UML and ADLs)
- Quick reference guides (for UML and ADLs)
Contents

- Software architecture fundamentals
- Motivation for architectural modeling
- Case study: order management in telecommunications
- Models in software architecture design: UMF and UML
- Concluding thoughts
Why Model (Revisited)?

- Abstraction helps the architect to *deal with complexity*

- Support a *divide-and-conquer* strategy to problem solving
  - Componentization supports division of labor
  - Encapsulation, esp. layering eases maintenance (swap parts in/out)
  - Traceability and auditability (think of recent financial crisis/crises)

- Promotes *accuracy and precision* (faithful to engineering spirit)
  - Example: Cardinalities of relations between domain entities and architectural components may make or break your project
    - E.g. database design (normalization?)
    - E.g. API design (method/operation signatures?)
    - E.g. number of test cases and structure of test data
    - E.g. security concerns (denial of service attacks)
Take Aways and Additional Information

- **Software architecture**
  - Start from functional requirements and – even more importantly – NFRs, particularly the explicit and tacit software quality attributes
  - Find architectural patterns which resolve the forces underneath the NFRs, e.g., in the POSA book series [BMR+96] and the [Fow03] and [HW04] books
  - Make conscious pattern selection decisions and follow-on decisions about technologies and products
    - Based on experience or reusable asset

- **Full lecture on enterprise-wide IT architectures available locally!** (Hoidn/Schlatter/Schwidder)
  - Link to 2010 edition
Recommended Reading: Fundamentals and Process


Recommended Reading: Patterns and SOA Design


Recommended Reading: Agile and Miscellaneous

[Amb] Ambler S., articles and book references available via his blog:  


[Sel11] Selic B., Making Abstraction Concrete, CompArch/WICSA 2011 keynote


- Software Architecture and Design at USI Lugano (link to 2011 edition)
- Also see the recommended reading list in lecture “enterprise-wide IT architectures” (link to 2010 edition)