Entwickeln und Dokumentieren von Softwarearchitektur

„Best Practices“ in Entwurf und Kommunikation

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

Wintersemester 2020/21

Übersicht

- Entwurf einer Softwarearchitektur
  - Entwicklungsprozess und Architektur
  - „Best Practices“
  - Voraussetzungen
- Vorgehen
- Dokumentation von Softwarearchitektur

Entwicklungsprozess und Softwarearchitektur

- Architektur ist *nicht* Phase des Entwicklungsprozesses, sondern Daueraufgabe
- Architektur als initialer Bauplan
- Interative Entwicklung ≙ Architekturverfeinerung und Architekturrefactoring
- Überprüfung architektonischer Richtlinien ≙ Erosion der Architektur verhindern
- Architektur und agiles Vorgehen – Diskussion, siehe zum Beispiel das Paper von Wils und Van Baelen

Entwurf einer Softwarearchitektur

- Kein „Königsweg“ oder Rezept
- Denn: Softwareentwicklung oft auf Neuland
- Leitlinie: „Best Practices“
- Erfahrungsschatz nutzen
- Architekturstile und -muster kennen
- Mechanismen für Qualitätsmerkmale kennen

Voraussetzungen

- Geschäftsziele
- Funktionale Anforderungen (einigermaßen) vollständig definiert; möglichst samt Anwendungsfälle/User Stories
- Gegebenheiten des Anwendungsgebiets
- Einschränkungen/Festlegungen bezüglich der Infrastruktur
- Gegebenheiten des Softwareproduktionsprozesses definiert

Übersicht

- Entwurf einer Softwarearchitektur
- Vorgehen
  - Komponenten für die Funktionalität
  - Qualitätsmerkmale und Mechanismen
  - Detaillierung und „Refactoring“
- Dokumentation von Softwarearchitektur

Initialen Achitekturentwurf finden

- Systemkontext definieren.
  Grundlage: Funktionale Anforderungen, Infrastruktur
- Domänenmodell erstellen .
  Grundlage: Funktionale Anforderungen, Anwendungsgebiet
  Techniken: Objektorientierte Modellierung,
  Informationsmodellierung, Geschäftsprozessmodellierung, ...
- Typ der Problemstellung $\hookrightarrow$ Architekturstil
  auch: Kombination von Stilen für Subsysteme
- Dekomposition in Komponenten und Konnektoren
  auf Basis der funktionalen Eigenschaften

☞ Initiale Architektur

Qualitätsmerkmale und Mechanismen

- Gewünschte Qualitätsmerkmale ermitteln
  Qualität für den Benutzer; Qualität der Entwicklung
- Qualitätsszenarien ermitteln
- Szenarien gegeneinander abschätzen:
  Trade-offs? Prioritäten?
- Mechanismen für Kernszenarien entwickeln und in die Architektur einbringen.

Detaillierung und „Refactoring“

- Anwendung von Mechanismen
  ⇒ Transformation von Qualitätsanforderungen in funktionale Komponenten
  ⇒ neue Komponenten, neue Konnektoren
- Anwendung von Entwurfsmustern
  ⇒ Einfluss auf Designzentren?
- Festlegung der Codestruktur: Abhängigkeiten von Code-Komponenten
  ⇒ Planung und Überwachung von Struktur im Code
  ⇒ Gleichzeitig Überprüfung der Architektur

Risiko im Vordergrund

- Architekturentwurf und agiles Vorgehen – Lange Architekturphase „up front“ nicht notwendig
- Stattdessen: Risiko im Fokus
- Identifizieren von Risiken, Priorisierung
- Mechanismus überlegen, der das Risiko vermindert
- Noch ein wichtiges Risiko?

Methode: Attribute-Driven Design

- ADD ist eine systemtische und iterative Methode für die Entwicklung einer Softwarearchitektur
- Pro Interationsschritt:
  - Wähle zu betrachtende Komponenten und definiere Ziel
  - Identifiziere relevante Geschäftsziele, Qualitätsattribute und Constraints
  - Finde geeignete Mechanismen dafür und verfeinere die Architektur entsprechend
- Resultat jeweils: Skizzen der Konzepte
- Der iterative Ansatz eignet sich zu Verwendung der Methode in agilen Prozessen

Literatur und Links zu ADD

Humberto Cervantes, Rick Kazman
Designing Software Architectures
Harlow, UK: Addison-Wesley, 2016.

Humberto Cervantes et al.
Smart Decisions: A game about architecting modern software systems
https://smartdecisionsgame.com

Literatur


Jan Bosch
Design and Use of Software Architectures
Harlow, UK: Addison-Wesley, 2000.


Rick Kazman, Paul Clements, Len Bass
Software Architecture in Practice Part Two
Boston: Addison-Wesley, Third Edition 2012.


George Fairbanks
Just Enough Software Architecture: A Risk-Driven Approach
Boulder, CO: Marshall & Brainard, 2012.


Michael Stal, Stefan Tilkov, Markus Völter, Christian Weyer
SoftwareArchitekTOUR - Podcast für den professionellen Softwarearchitekten – Episode über Architektur-Refactoring
www.heise.de/developer/podcast/

Übersicht

- Entwurf einer Softwarearchitektur
- Vorgehen
- Dokumentation von Softwarearchitektur
  - Perspektiven und Sichten
  - Notationen
  - Beispiele

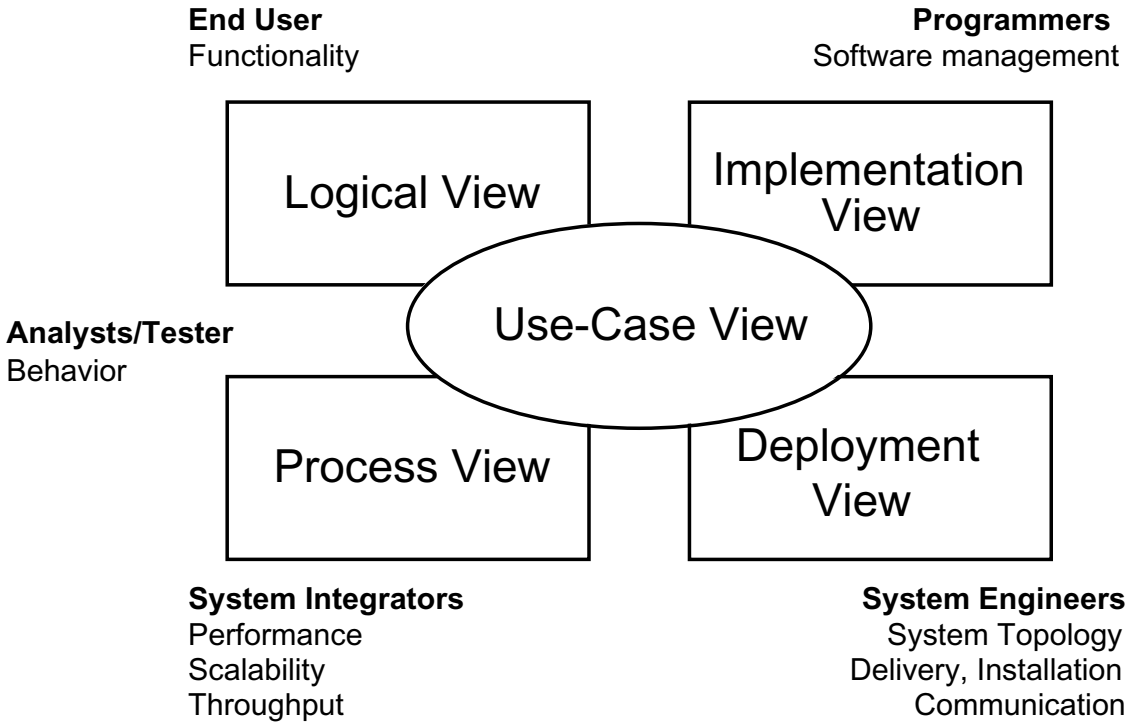
Sichten der Softwarearchitektur

Das Wesentliche der Architektur lässt sich in der Regel nicht in *einer* Sicht allein darstellen. Man unterscheidet deshalb verschiedene Sichten.

Im konkreten Fall wählt man die „passenden“ Sichten, um die Architektur darzustellen.
Es gibt verschiedene Methoden, Softwarearchitektur darzustellen:

- Das 4+1-Sichten-Modell von Phillipe Kruchten, verwendet im (Rational) Unified Process
- 4 Sichten von Hofmeister, Nord und Soni
- Viewtypes und Styles des SEI
- Canonical Model Structure von George Fairbanks
- FMC Fundamental Modeling Concepts, gelehrt am Hasso-Plattner-Institut

Kruchten & UML



The “4+1” view model (Phillipe Kruchten 1995)

Charakteristik der 4+1-Sichten

Für jede Sicht wird angegeben:
I Inhalt, **K** Komponenten, **B** Beziehungen und **S** Stakeholder.

- Use Case Sicht
  - I: Verhalten des Systems
  - K: Akteure, Anwendungsfälle
  - B: Interaktion, Verwendung, Vererbung
  - S: Anwender, Analytiker, Tester
- Logische Sicht
  - I: Vokabular des Gebietes, Funktionalität
  - K: Klassen, Verantwortlichkeiten, Kollaborationen
  - B: Assoziation, Vererbung, Abhängigkeit, Steuerung
  - S: Anwender, Analytiker, Designer, Bereichsexperte

Charakteristik der 4+1-Sichten

- Prozess-Sicht
  - I: Performanz, Skalierbarkeit, Verfügbarkeit
  - K: Prozesse, Threads
  - B: Aktivierungssteuerung, (gemeinsame) Resourcen
  - S: Designer, Deployer
- Implementierungs-Sicht
  - I: Systembestandteile, Konfigurationsmanagement
  - K: Dateien, Repositories
  - B: Enthaltensein, Abhängigkeit
  - S: Designer, Entwickler, Konfigurationsmanager
- Physische Sicht
  - I: Hardwaretopologie
  - K: Hardwareresourcen
  - B: Kommunikationskanäle, Abhängigkeit
  - S: Hardwareingenieur, Deployer.

Hofmeister, Nord und Soni

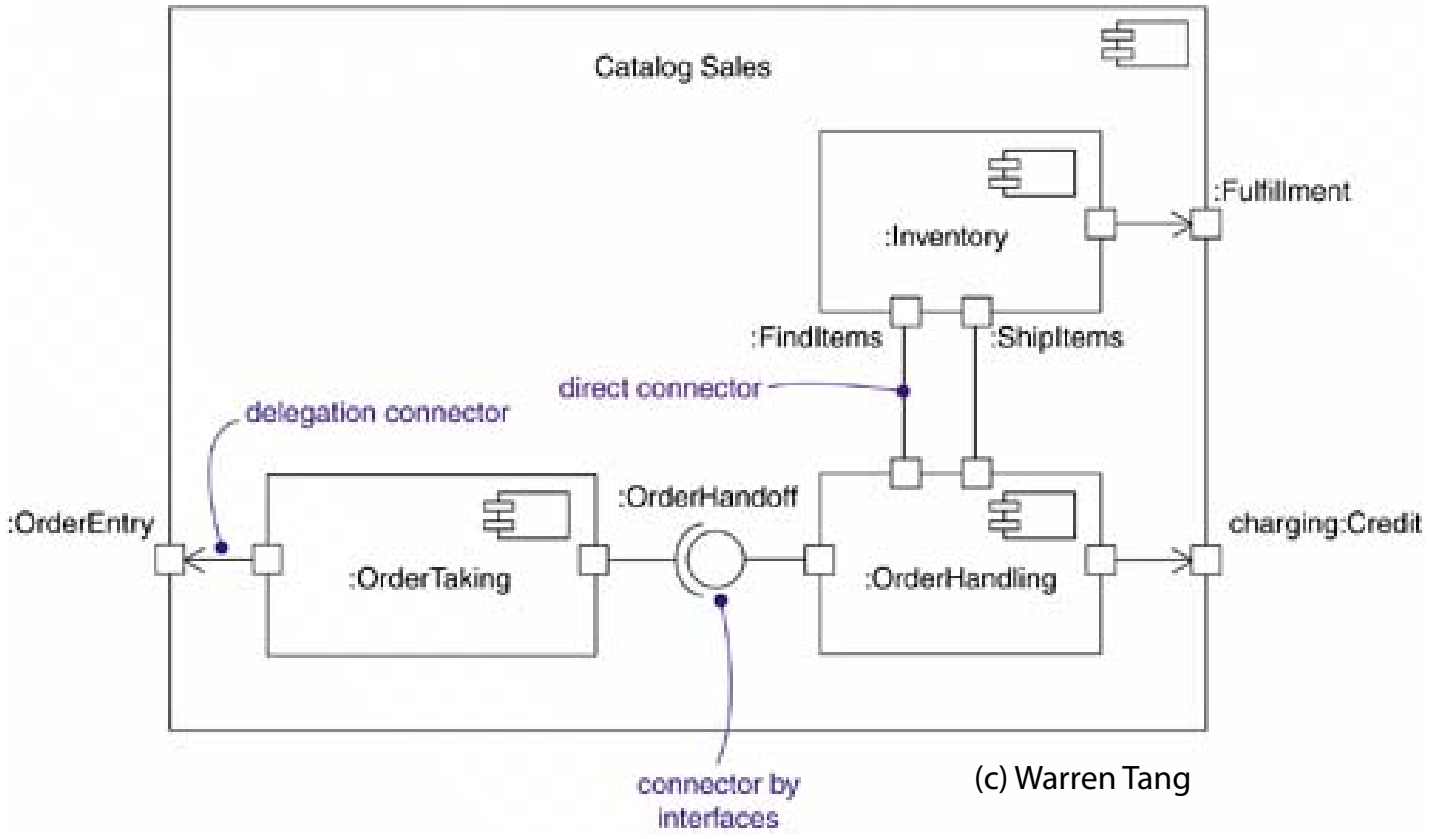
Konzeptionelle Sicht beschreibt Komponenten und Konnektoren und wie sie zusammenarbeiten

Modulsicht beschreibt Subsysteme, bestehend aus Modulen mit ihren Schnittstellen, eventuell angeordnet in Schichten

Ausführungssicht beschreibt Ausführungseinheiten (z.B. Prozesse), die auf einer bestimmten Plattform laufen und kommunizieren

Codesicht beschreibt Quelldateien, binäre Komponenten, Bibliotheken, ausführbare Programme und weitere Dateien

UML Composite Structure Diagram



Elemente des Composite Structure Diagrams

- *Structured Class*
  Klasse, die eine interne Struktur hat, bestehend aus Eigenschaften (properties), Teilen (parts) mit bestimmten Rollen (roles) und Konnektoren (connectors), sowie Ports.
- *Property, Part, Role*
  Properties sind Instanzen, die Teil der Structured Class sind, Parts speziell Aggregationen von Properties, sie können bestimmte Rollen spielen
- *Connectors*
  sind Assoziationen innerhalb der Komponente, die einen Kommunikationskanal repräsentieren
- *Ports*
  sind Interaktionspunkte einer strukturierten Klasse
  (1) nach außen (*service port*), oder
  (2) zu inneren Teilen (*behavior port*).

Viewtypes des SEI

Viewtype = Perspektive auf das System

A viewtype defines the element types and relationship types used to describe the architecture of a software system from a particular perspective

Drei Viewtypes

1. Module viewtype
2. Component-and-connector viewtype
3. Allocation viewtype

Styles

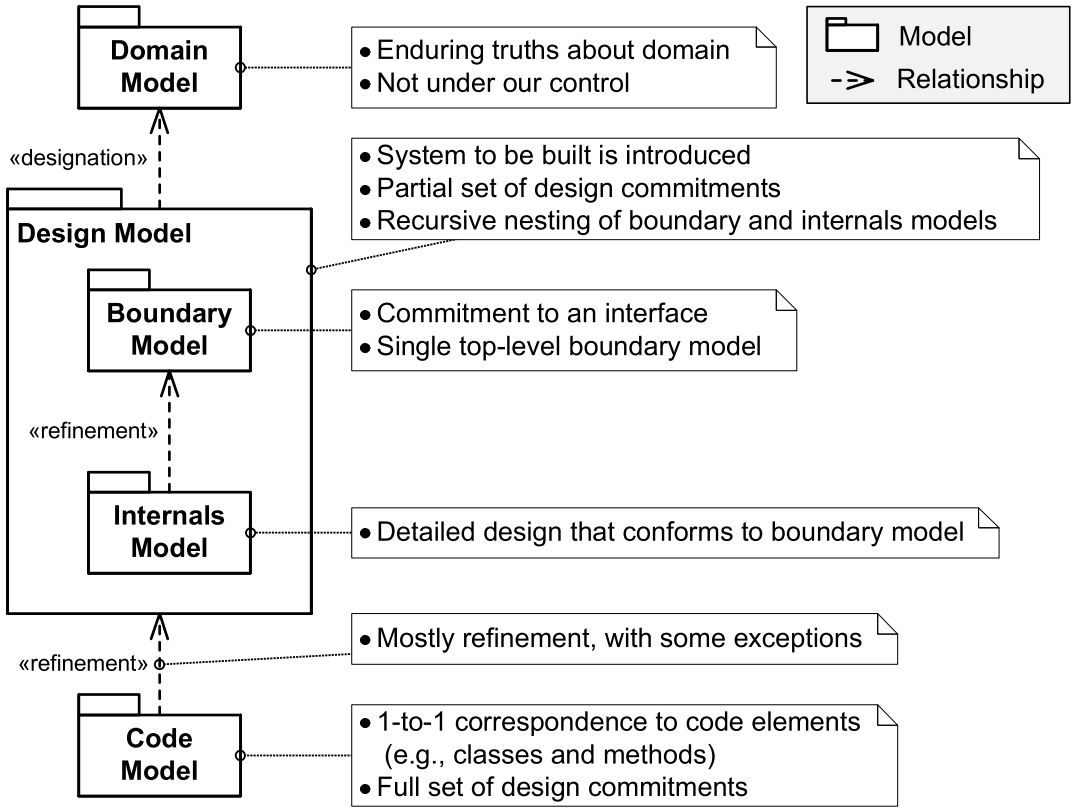
Style = Spezielle Ausprägung/Muster der Perspektive

A style guide is the description of an architectural style that specifies the vocabulary of design (set of element and relationship types) and the rules (sets of topological and semantic constraints) for how that vocabulary can be used.

Beispiel

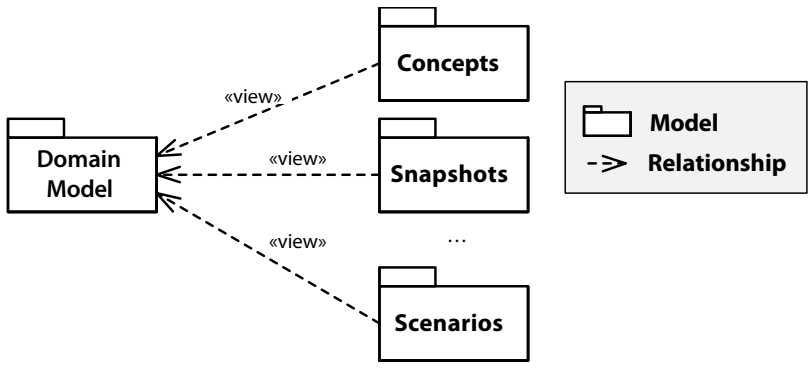
Pipes & Filters ist ein Style des Component-and-Connector Viewtypes.

Canonical Model Structure (George Fairbanks)



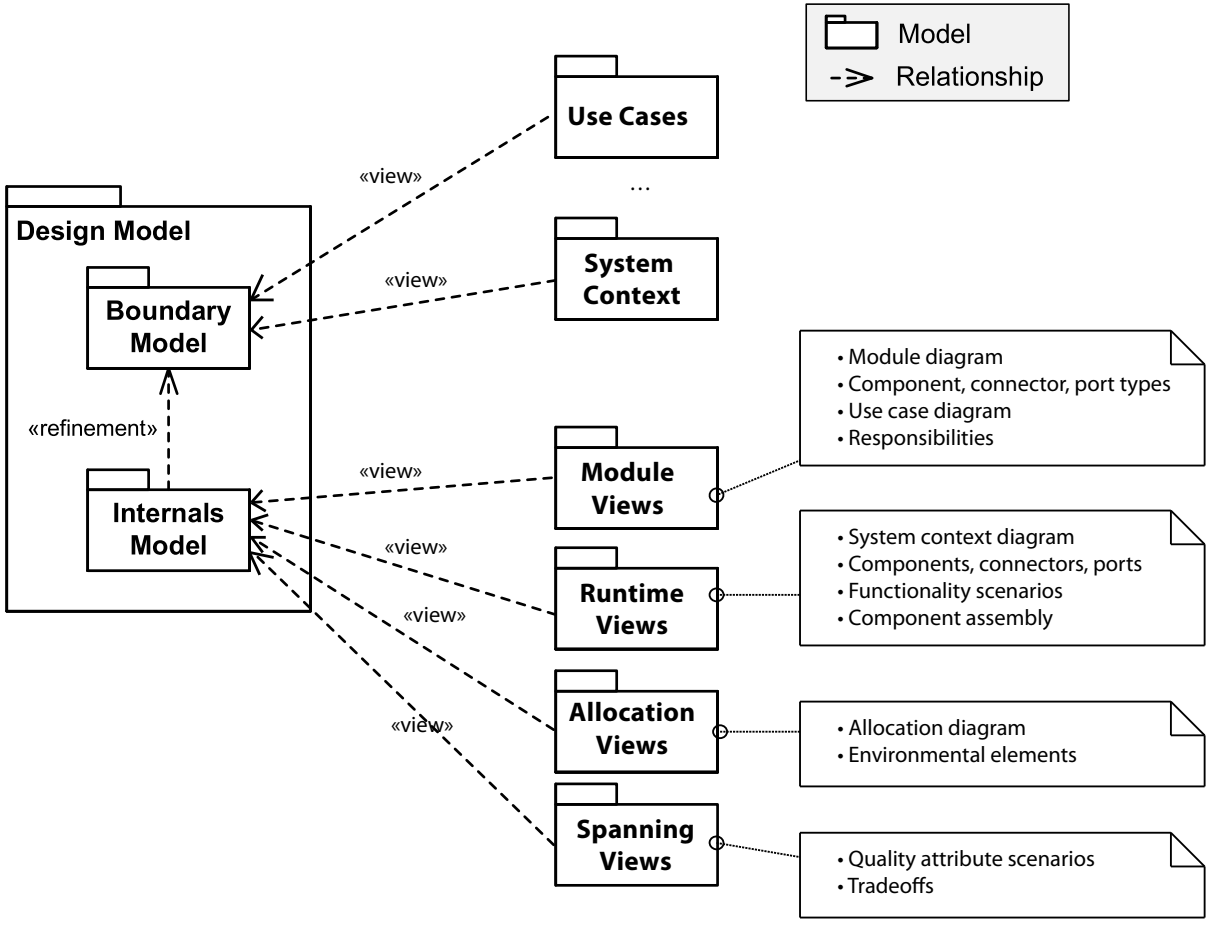
George Fairbanks: Just Enough Software Architecture, S.116

Sichten auf das Domain Model



George Fairbanks: Just Enough Software Architecture, S.119

Sichten auf das Design Model



George Fairbanks: Just Enough Software Architecture

Fundamental Modeling Concepts

Ausgehend von einer Klassifikation dynamischer Systeme schlägt Siegfried Wendt eine Darstellung der fundamentalen Strukturen eines Softwaresystems vor, die die grundlegende konzeptionelle Architektur des Systems verständlich machen

Drei fundamentale Strukturen in informationellen dynamischen Systemen:

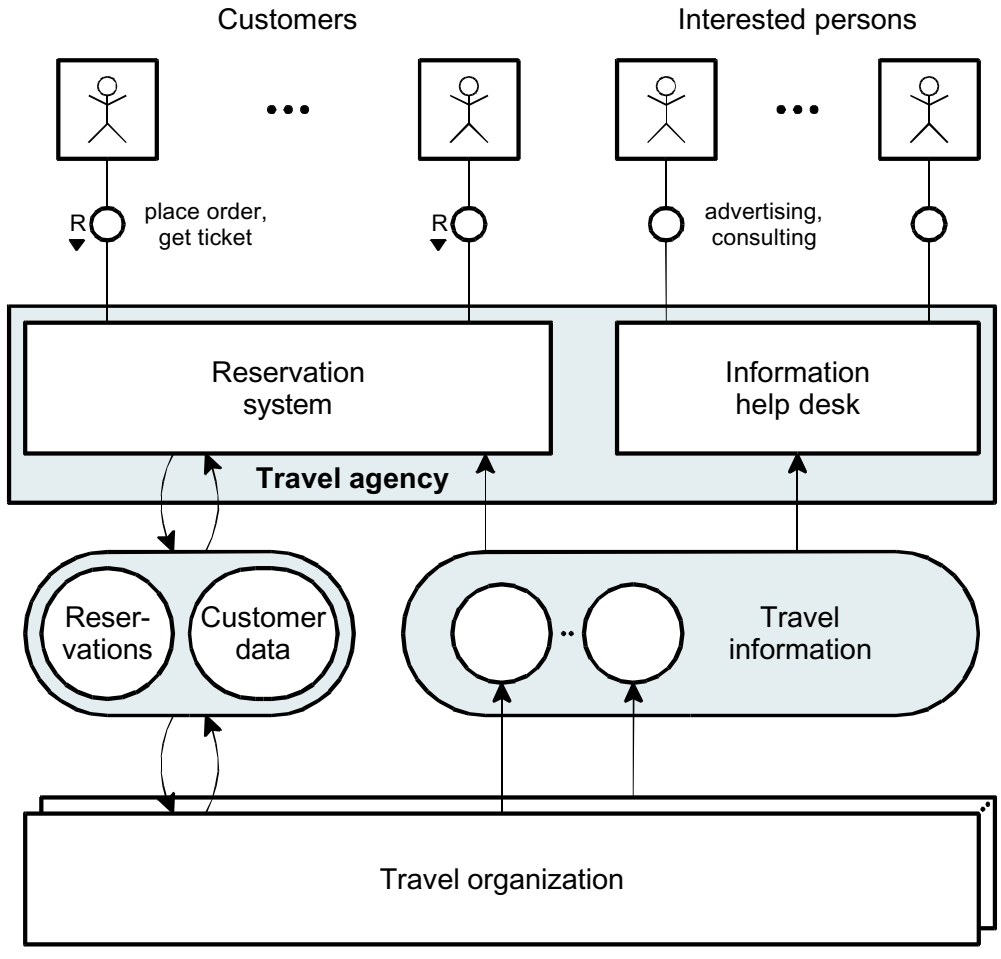
- Aufbaustrukturen
- Wertebereichsstrukturen
- Ablaufstrukturen

Notation von FMC

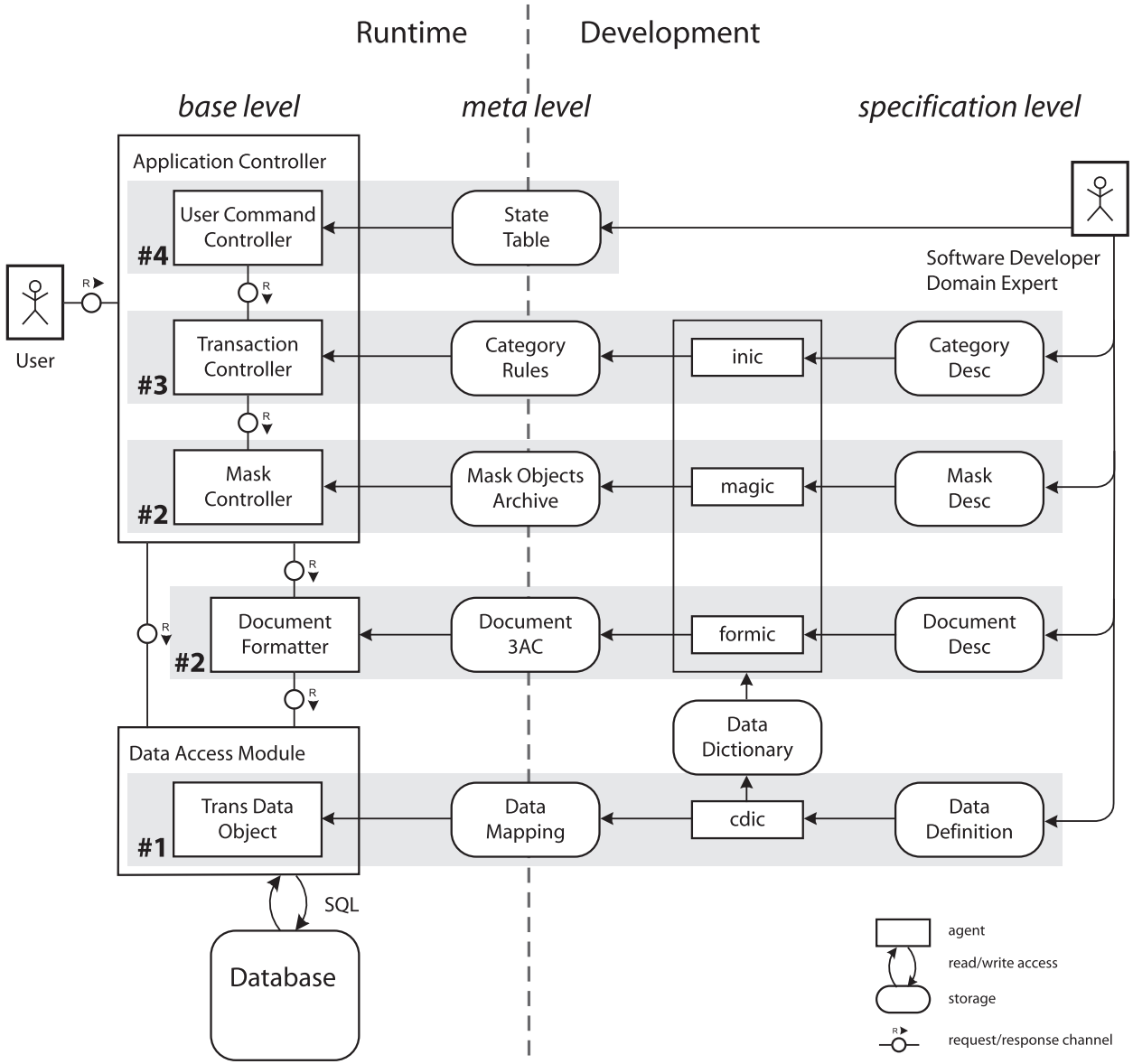
Bipartite Graphen

- Aufbaustrukturen bestehen aus
  - aktiven, verarbeitenden Komponenten (Agenten)
  - passiven, datenhaltenden Komponenten (Kanälen und Speichern)
  - Struktur: Agenten verarbeiten Daten, Ergebnisse werden an Kanälen oder in Speichern beobeachtbar
- Darstellung der Wertebereichsstukturen durch Mengen und Relationen, optisch ähnlich den Venn-Diagrammen, inhaltlich Entity-Relationship-Modelle
- Ablaufstrukturen durch Petri-Netze, genauer Stellen-Transitions-Netze

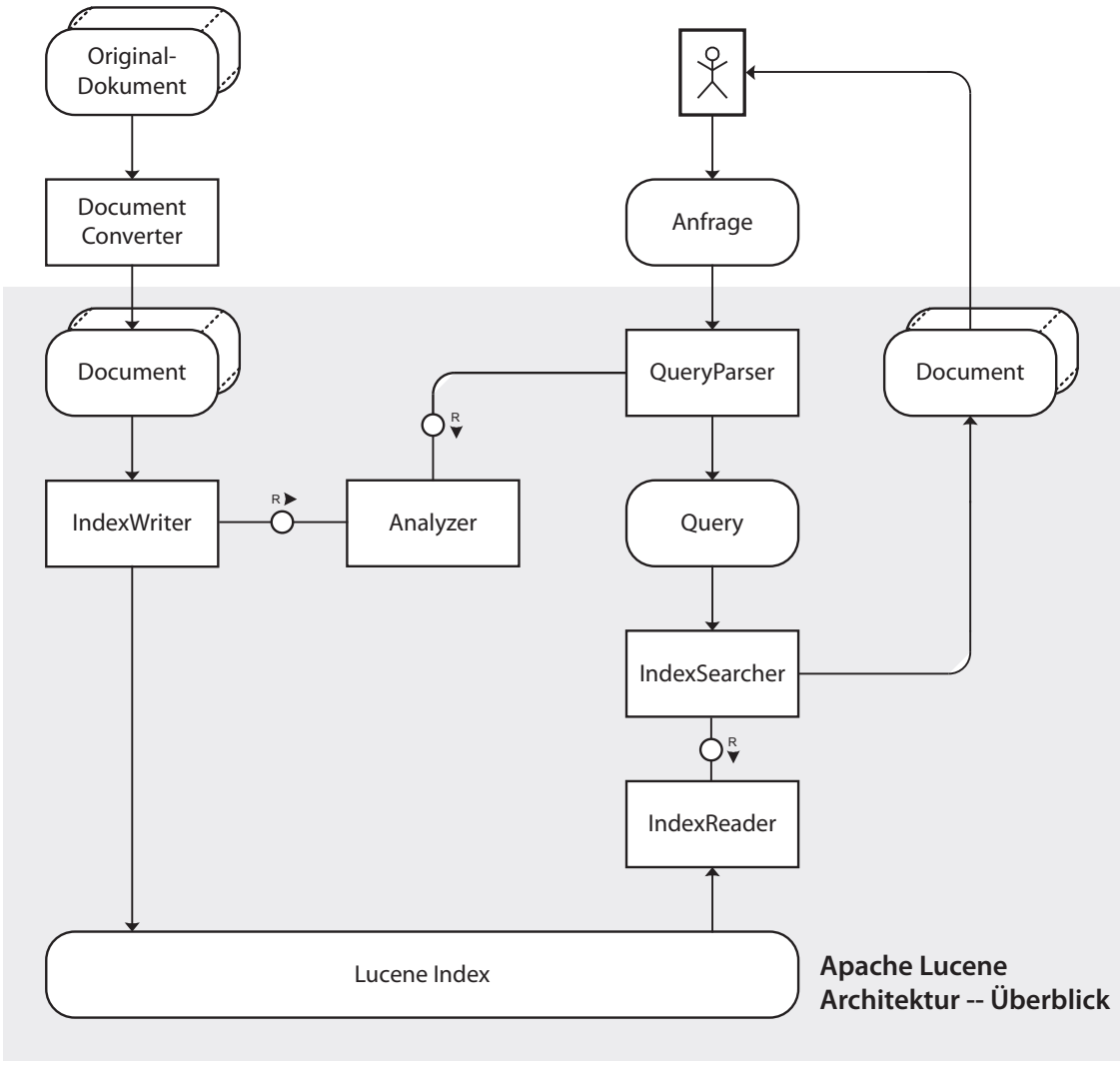
Beispiel Aufbaustruktur in FMC



Beispiel Architektur von AutiSta



Beispiel Architektur von Apache Lucene



Literatur


Philippe Kruchten
Architectural Blueprints – The „4+1“ View Model of Software Architecture
IEEE Software 12(6), November 1995


Christine Hofmeister, Robert Nord, Dili Soni
Applied Software Architecture
Reading, MA: Addison-Wesley, 2000.


Paul Clements et al.
Documenting Software Architectures: Views and Beyond
Boston: Addison-Wesley, 2003.

Literatur

George Fairbanks
Just Enough Software Architecture
Boulder, CO: Marshall & Brainerd, 2012.

Andreas Knöpfel, Bernhard Gröne, Peter Tabeling
Fundamental Modeling Concepts
John Wiley & Sons, 2005.
http://www.fmc-modeling.org/