Diplomarbeit

„Simulationssystem zur Analyse von Fehlertoleranzverfahren in physiknahen service-orientierten Architekturen am Beispiel einer virtuellen Förderanlage“

Jan Krüger, jan.krueger@tu-dortmund.de

10. März 2008

Betreuer: Prof. Dr. Heiko Krumm, Dipl.-Inform. Andre Pohl

(0) Übersicht

1. Einleitung / Motivation
2. Grundlagen: Fehlertoleranz
3. Grundlagen: service-orientierte Architekturen
4. Grundlagen: Simulation
5. Zu simulierendes (Beispiel-) System
6. Anforderungen an die Software
7. Grundlagen der Implementierung
8. Modell-Framework
9. Simulations-Umgebung

---

10. Anwendungsbeispiel

1. Einleitung / Motivation

(1.1) Einleitung / Motivation

„Trend“ zu service-orientierten Architekturen

und parallel

Leistungsstärkere, netzwerkfähige eingebettete Systeme + Einsatz dieser bei der Automatisierung

Übertragung des SOA-Konzeptes auf automatisierte Systeme

➔ vielfältige neue Kontroll- und Steuerungsmöglichkeiten mittels von Komponenten des automatisierten Systems bereitgestellter Dienste

(1.2) Einleitung / Motivation

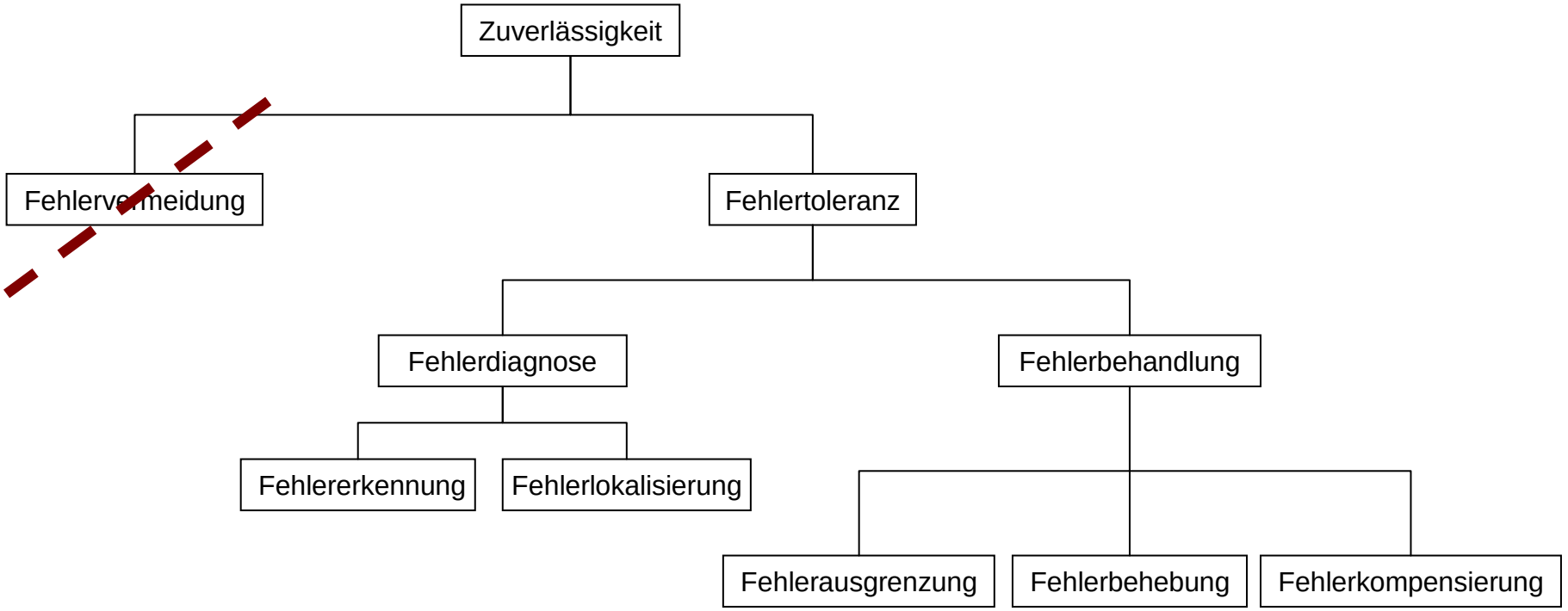
- Problem bei Automatisierung: Ausfälle von bzw. Fehler an Komponenten im automatisierten System
- Lösungsansatz: Fehlertoleranzverfahren
- SOA-Konzept ermöglicht hier flexibleren Einsatz derartiger Verfahren als bisher
- Aber: bisher kaum praktische Erkenntnisse über den Einsatz von FT-Verfahren in SOA-konformen, automatisierten Systemen
- Also: Untersuchungen und Tests notwendig

➔ Ziel hier: Entwicklung / Implementierung eines Werkzeuges zur Simulation von gemäß dem SOA-Konzept aufgebauten automatisierten Systemen und zum Testen von Fehlertoleranzverfahren mittels dieser Simulationen

2. Grundlagen: Fehlertoleranz

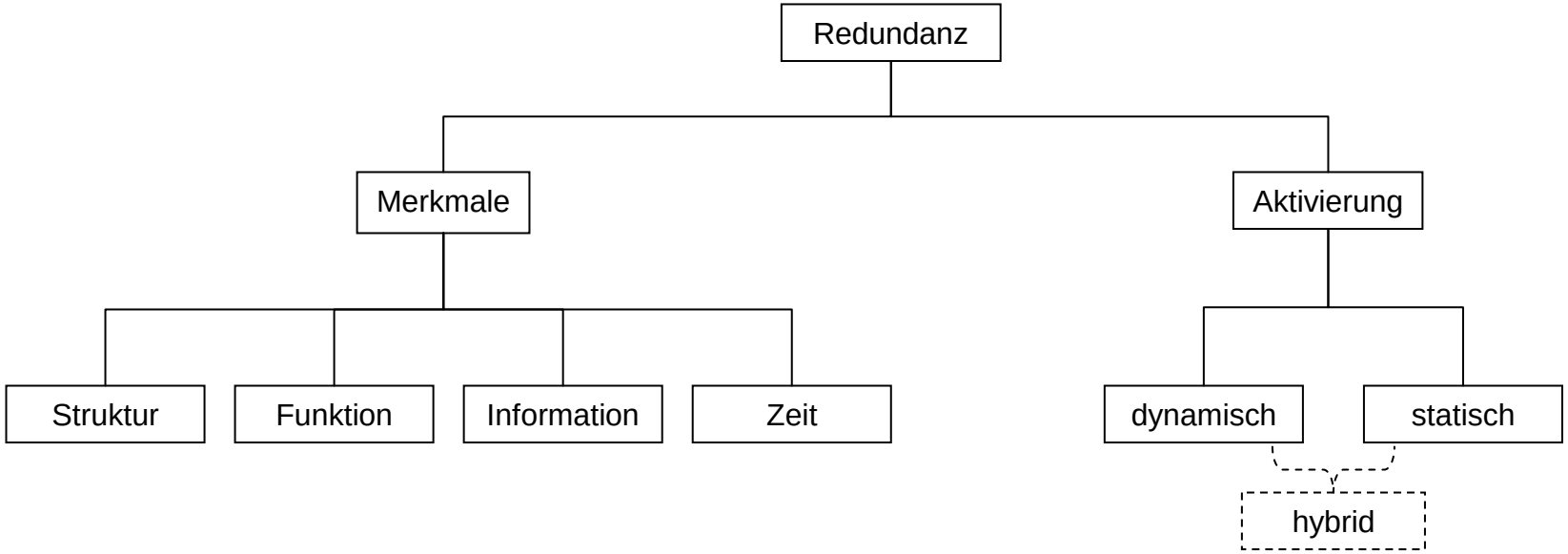
(2.1) Grundlagen: Fehlertoleranz

- Fehlertoleranz: „Die Fähigkeit eines Systems, auch mit einer begrenzten Anzahl fehlerhafter Komponenten seine spezifizierte Funktion zu erfüllen.“
- Ziel: Erhöhung der Zuverlässigkeit des Systems



(2.2) Grundlagen: Fehlertoleranz

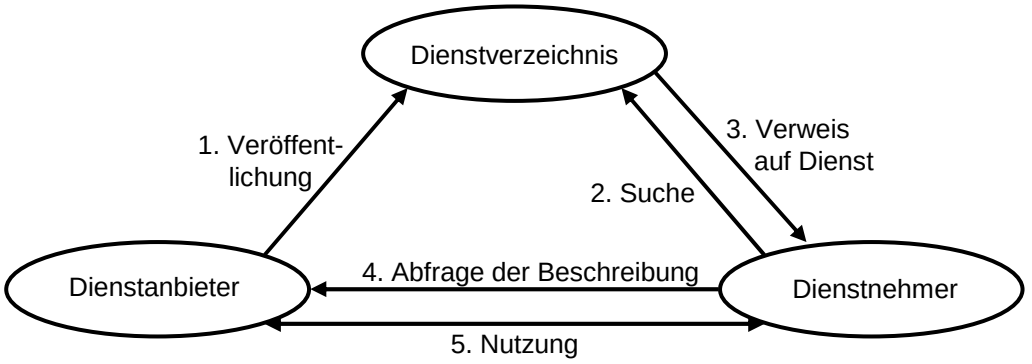
- Zentral für Gewährleistung von fehlertolerantem Verhalten: Redundanzen im System
- Redundanz = „Funktionsbereites Vorhandensein von mehr technischen Mitteln, als für die spezifizierte Nutzfunktion eines Systems benötigt werden.“



3. Grundlagen: service-orientierte Architekturen

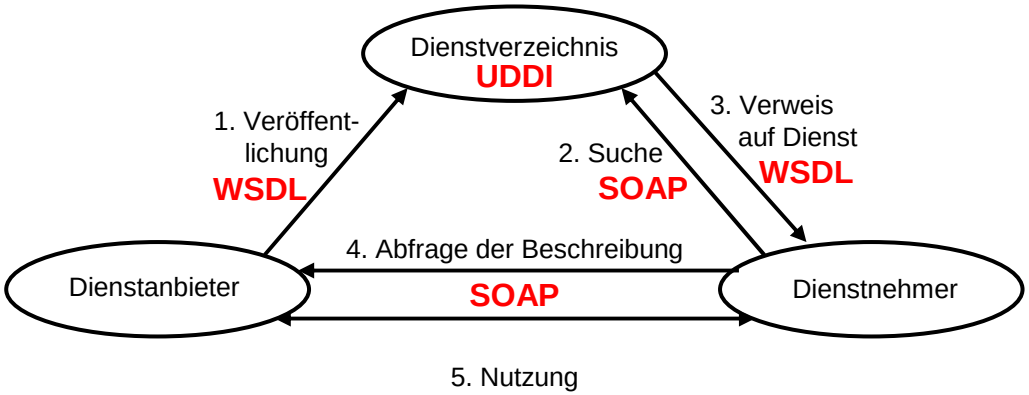
(3.1) Grundlagen: service-orientierte Architekturen / Konzept

- Abstraktes Konzept zum Anbieten, Suchen und Nutzen von Diensten
- Anforderung: Diensten zugrundeliegende Hardware / Betriebssysteme / Programmiersprachen für Dienst-Interaktionen unerheblich
  ➔ Dienst-Zugriff über öffentlich beschriebene Dienst-Schnittstelle
- Zentrale Merkmale: lose Kupplung / dynamisches Binden
- Rollen in einer SOA: Dienstverzeichnis, Dienstanbieter, Dienstnehmer



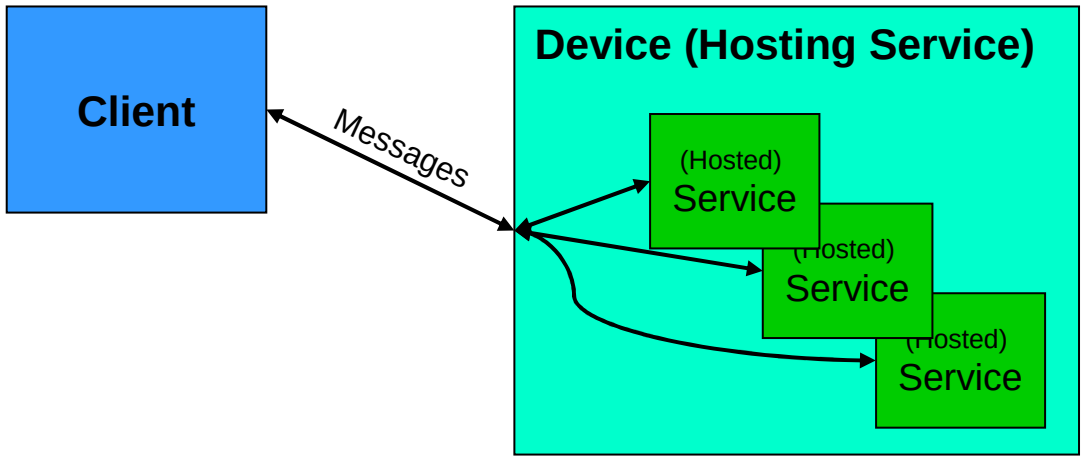
(3.2) Grundlagen: service-orientierte Architekturen / Web Services

- Web Services (WS): Implementierungsansatz einer SOA
- XML als standardisiertes Datenformat
- Schnittstellenbeschreibungen mittels WSDL
- Realisierung des Dienstverzeichnisses mittels UDDI
- Datenübertragungen mittels SOAP



(3.3) Grundlagen: service-orientierte Architekturen / Devices Profile for Web Services (DPWS)

- DPWS als Erweiterung des Konzeptes der Web Services
- WS-Eventing zum Empfangen von durch sog. Event Sources ausgelösten Events an sog. Event Sinks
- WS-Discovery zum Suchen und Lokalisieren von verfügbaren Diensten, sowie zum An- / Abmelden von Diensten („Hello“ / „Bye“)



4. Grundlagen: Simulation

(4.1) Grundlagen: Simulation

- Simulation = Reproduktion eines *Systems* anhand eines *Modells*, welches für *Erkenntnisgewinn* wichtige System-Aspekte *nachbildet*

- Ziel: aus dem Simulationsergebnissen Rückschlüsse auf System-Eigenschaften zu ziehen

---

➔ warum simulieren? Vorteile finden sich z. B. bezüglich

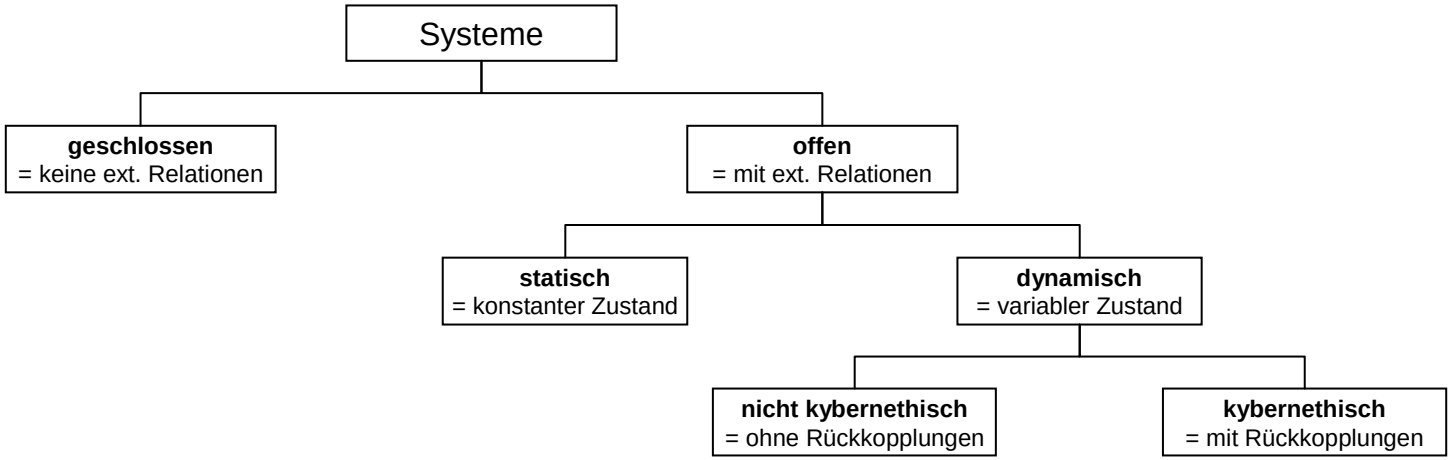
- Zeit
- Kosten
- Risiken
- ...

(4.2) Grundlagen: Simulation / System-Begriff

- System-Merkmale:
  - erkennbare Funktion / Systemzweck
  - wirkungsverknüpfte Systemelemente mit individuellen Attributen
  - Systemidentität geht bei System-Teilung verloren
- System-Zustand = Kombination aller Systemelement-Zustände
- System-Verhalten = Abfolge der System-Zustände über die Zeit

---

- System-Ausprägungen:

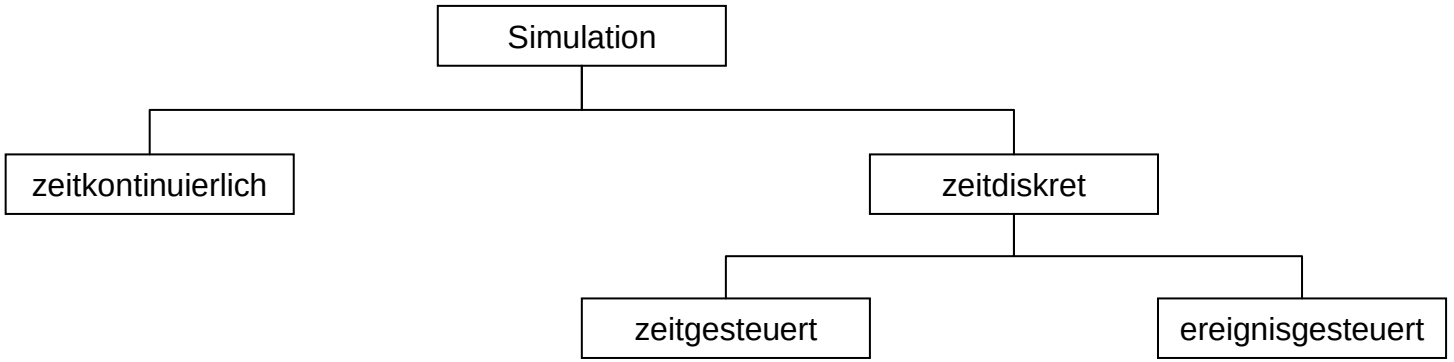


(4.3) Grundlagen: Simulation / Modell-Begriff

- Modell = Nachbildung eines (Real-) Systems
- Bei Modellbildung: Abstraktion und Idealisierung des Systems zur Vereinfachung komplexer Systeme
- Problem: richtiges Maß an Vereinfachung muss bestimmt werden
- Modell-Sichtweisen:
  - „Black-Box“: Systemverhaltens-Nachbildung
  - „White-Box“: Systemstruktur-Nachbildung
  - „Grey-Box“: Mischform
- Vorteile: weniger Aufwand, keine Gefahren, größerer zu testender Parameterbereich denkbar, ...
- Nachteile: Modell-Entwicklung notwendig, Unsicherheit bzgl. der Modell-Genauigkeit, reale Gefahren bei falschen Modell-Nachbildungen

(4.4) Grundlagen: Simulation / Verfahren

- Einteilung der Simulations-Verfahren:



- Zeitkontinuierliche Verfahren: keine Zeitsprünge, simuliertes Modell ist zu jedem Zeitpunkt in genau definiertem bzw. berechenbarem Zustand

- Zeitdiskrete Verfahren: Modell-Zustand ändert sich sprunghaft, zwischen zwei Zeitpunkten t_i und t_{i+1} lässt sich kein Zwischenzustand bestimmen
  ➔ zeitgesteuert = feste Zeitsprünge Δt
  ➔ ereignisgesteuert = Betrachtung der Zustandsänderungs-Zeitpunkte

5. Zu simulierendes (Beispiel-) System

(5.1) Zu simulierendes (Beispiel-) System

- Vorbemerkung: hier nicht Untersuchung *des* realen Systems, sondern des *allgemeinen Systemkonzeptes*
- Daher auch: keine Kenntnisse über konkrete Wirkungsbeziehungen zwischen konkreten Systemelementen
- Systemzweck:
  - Durchschleusung von zu transportierenden Elementen („Payloads“) durch das System
  - Payloads betreten das System, werden innerhalb dieses evtl. „bearbeitet“, und verlassen das System schließlich

(5.2) Zu simulierendes (Beispiel-) System / Systemkomponenten

- Systemkomponenten:
  - Eintritts- / Austritts-Komponenten für Payloads
    ➔ InPort, OutPort
  - innere Transport-Komponenten
    ➔ Conveyer, Gate, RobotGrabber
  - innere Bearbeitungs-Komponenten
    ➔ WorkStation
  - Zusatz-Komponenten zur Unterstützung anderer Komponenten
    ➔ LightBarrier
- Systemkomponenten sind service-orientiert, d. h. stellen Devices bzw. Services zur Steuerung, Kontrolle und Informationsabfrage bereit

(5.3) Zu simulierendes (Beispiel-) System / reales Beispiel

Beispielhafte reale Förderanlage / reale Anlagenkomponenten

(5.4) Zu simulierendes (Beispiel-) System / Simulationsmodell + Verfahren

- Resultierendes Modell: Konstruktion „des“ Simulations-Modells nicht möglich
- Stattdessen: Konstruktion der einzelnen Komponenten eines jeden Modells
- Anforderungen an die einzelnen Komponenten z. B.:
  - Verwaltung der Relationen bzw. Interaktionen,
  - Durchführung der komponenten-spezifischen Aktionen,
- Anforderungen an das Umfeld eines jeden gebildeten Modells z. B.:
  - Verwaltung des Fortschreitens der Simulation,

➔ Für derartiges Modell-Prinzip gewähltes Simulations-Verfahren bzw. gewähltes Vorgehen beim Simulieren:

zeitdiskret und dabei zeitgesteuert

6. Anforderungen an die Simulations-Software

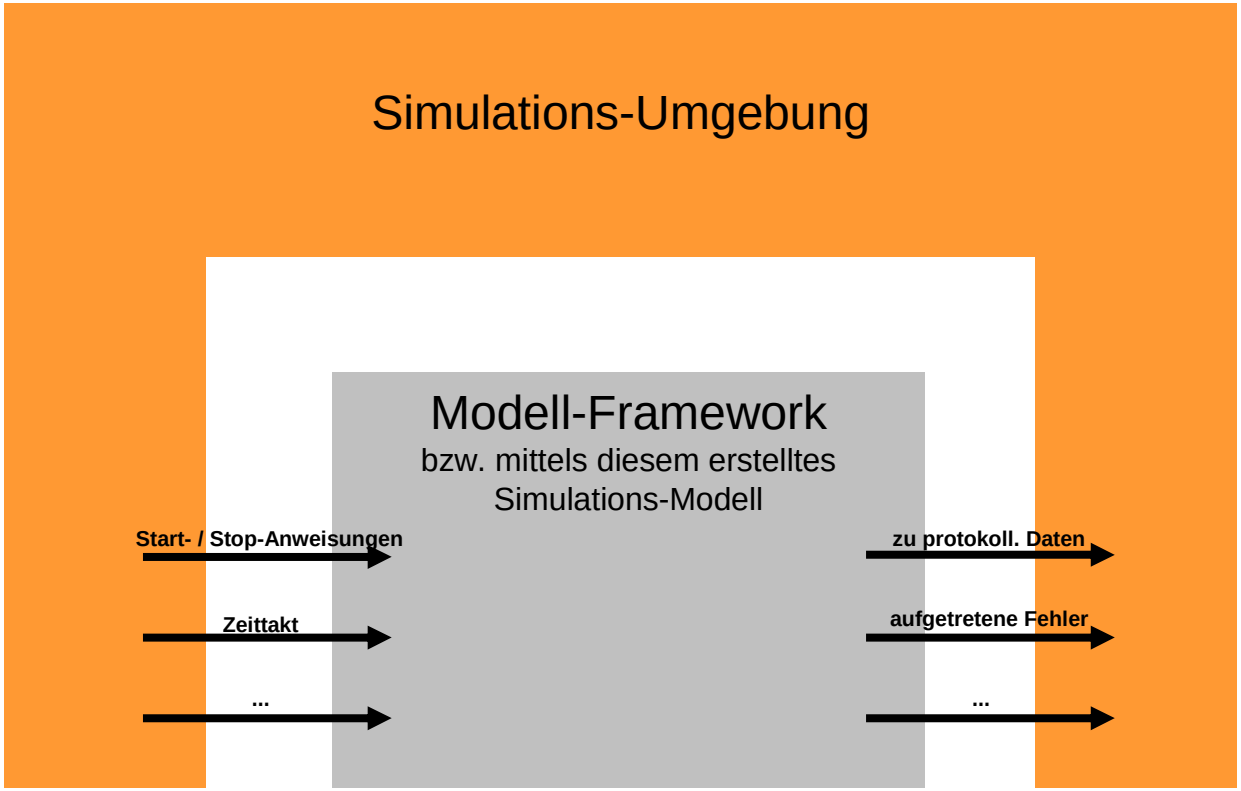
(6.1) Anforderungen an die Simulations-Software

- Unterstützung beliebiger (zulässiger) Modelle
- Erstellen, Speichern und Laden von Modellen in leicht verständlicher Struktur
- Unterstützung von Echtzeit und simulierter Zeit
- Möglichkeit zur externen Steuerung und Kontrolle sowohl der Software als auch der Modelle (mittels DPWS)
- Übersichtlichkeit bzw. Einfachheit der Benutzerschnittstelle
- Protokollierung sowohl von Simulations-Verwaltungsdaten als auch von Zuständen der Modell-Komponenten über die Zeit
- Modifizierbarkeit bzw. Modularität
- ...

7. Grundlagen der Implementierung

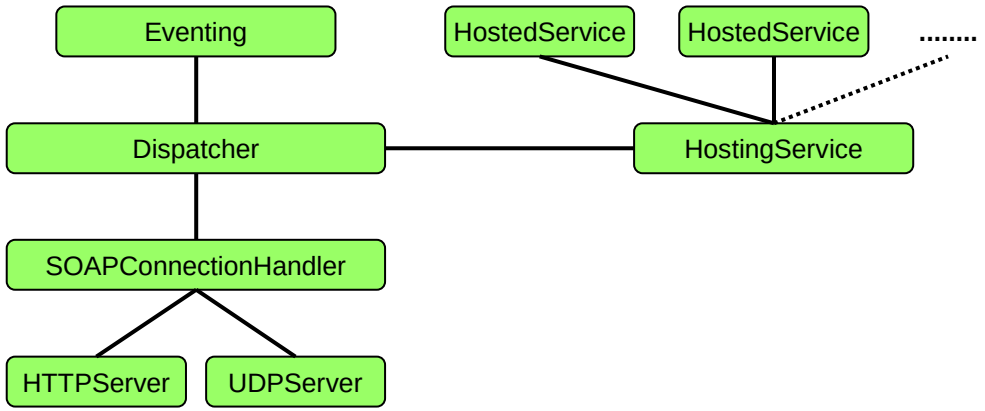
(7.1) Grundlagen der Implementierung / Unterteilung der SW

- Auftrennung der Software in 2 logische Teile:
  - Modell-Framework zum Zusammenstellen von Modellen
  - Simulations-Umgebung, in welche ein Modell eingebunden wird



(7.2) Grundlagen der Implementierung / Programmiersprache + benutzte Bibliotheken

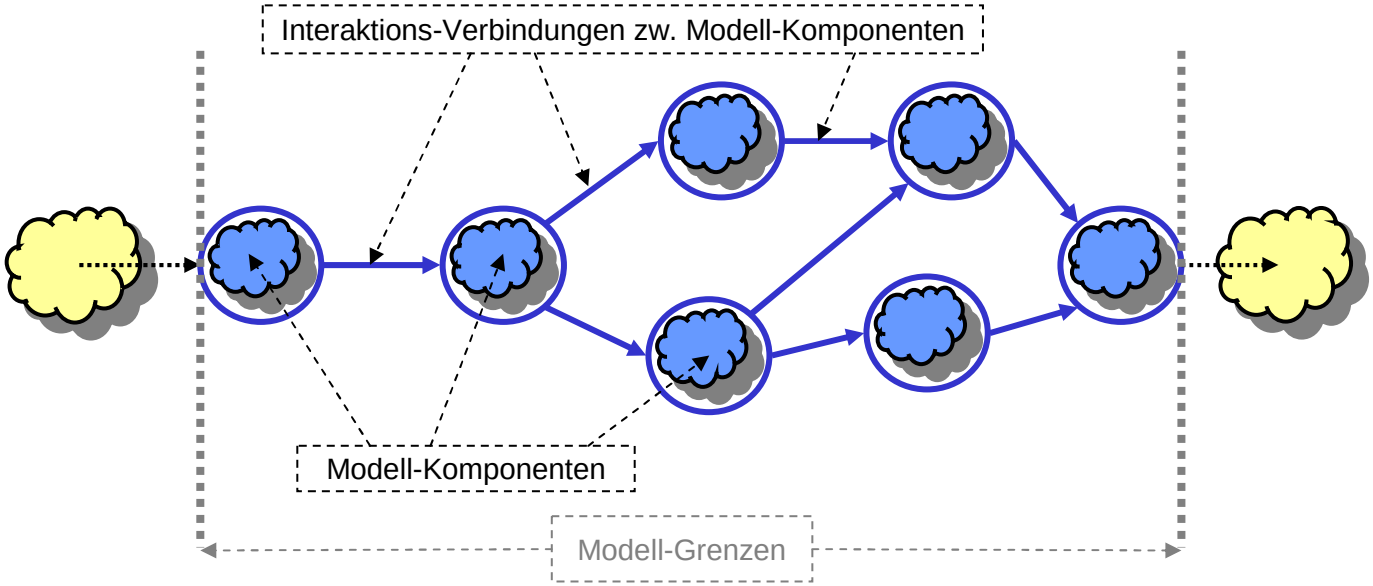
- Implementierung in Java
- Verwendung des Standard Widget Toolkit / SWT für die Gestaltung der graphischen Benutzerschnittstelle
- JDOM für die XML-Verarbeitung
- Realisierung der DPWS-Konformität mittels Verwendung des WS4D-J2ME – Stacks



8. Modell-Framework

(8.1) Modell-Framework: Grundlagen / Problemstellung

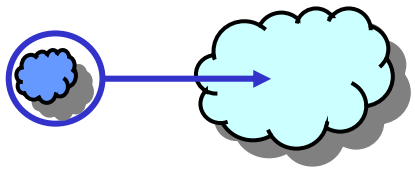
- Allgemeine Struktur eines Simulations-Modells:



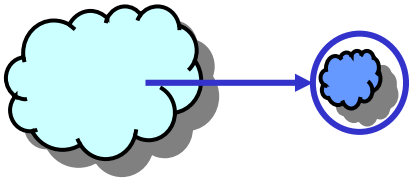
- Zwei mittels dem Framework zu lösende Teilaufgaben:
  - Realisierung der verschiedenen Modell-Komponenten
  - Realisierung der Verbindungen zwischen Komponenten

(8.2) Modell-Framework: Realisierung der Komponenten-Verbindungen

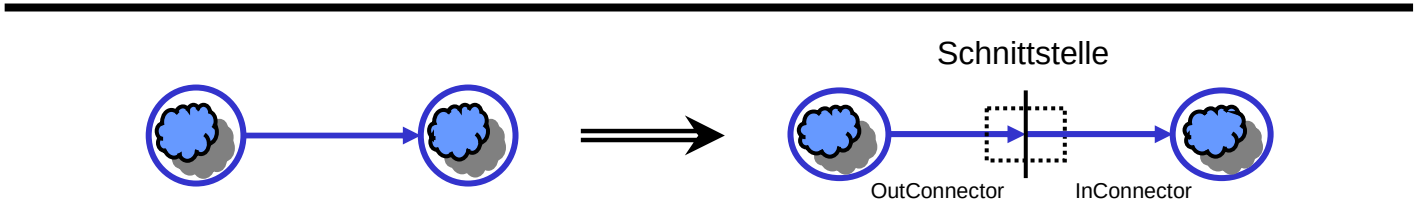
- Überlegung: jede konkrete Modell-Komponente verwaltet selbstständig ihre Interaktions-Verbindungen
- Dafür: Einführung spezieller Verbindungs-Elemente; sog. Connectoren
- Überlegungen für diese Connectoren:



OutConnector



InConnector



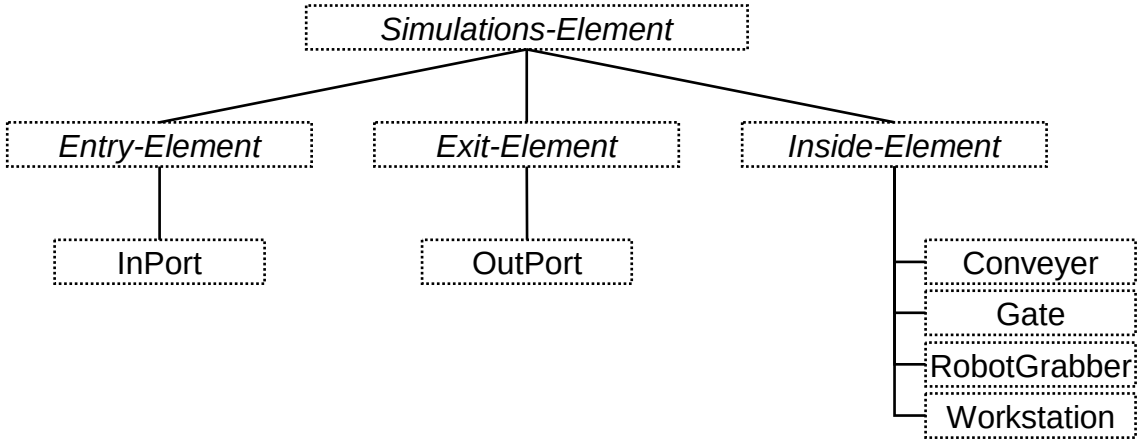
(8.3) Modell-Framework: Realisierung der Komponenten-Verbindungen

- Also: Einführung jew. einer Klasse `InConnector` und `OutConnector`
- Jede konkrete Modell-Komponente verwaltet je eine Liste ihrer In- und / oder OutConnectoren; diese werden ausschließlich dort verwaltet!
- Die Connector-Klassen stellen alle benötigten Funktionen bzw. Methoden zur Verwaltung bzw. Interaktion bereit, z. B.
  - Ermitteln des Besitzers
  - Verbinden des Connectors mit einem korrespondieren Connector (OutConnector ⇔ InConnector)
  - Ermitteln des verbundenen Connectors und somit der verbundenen Modell-Komponente
  - Übertragen von Payloads
  - ...

➔ *Connectoren ermöglichen somit das flexible Verwalten von Verbindungsstrukuren und den Payload-Transfer zwischen Modell-Komponenten!*

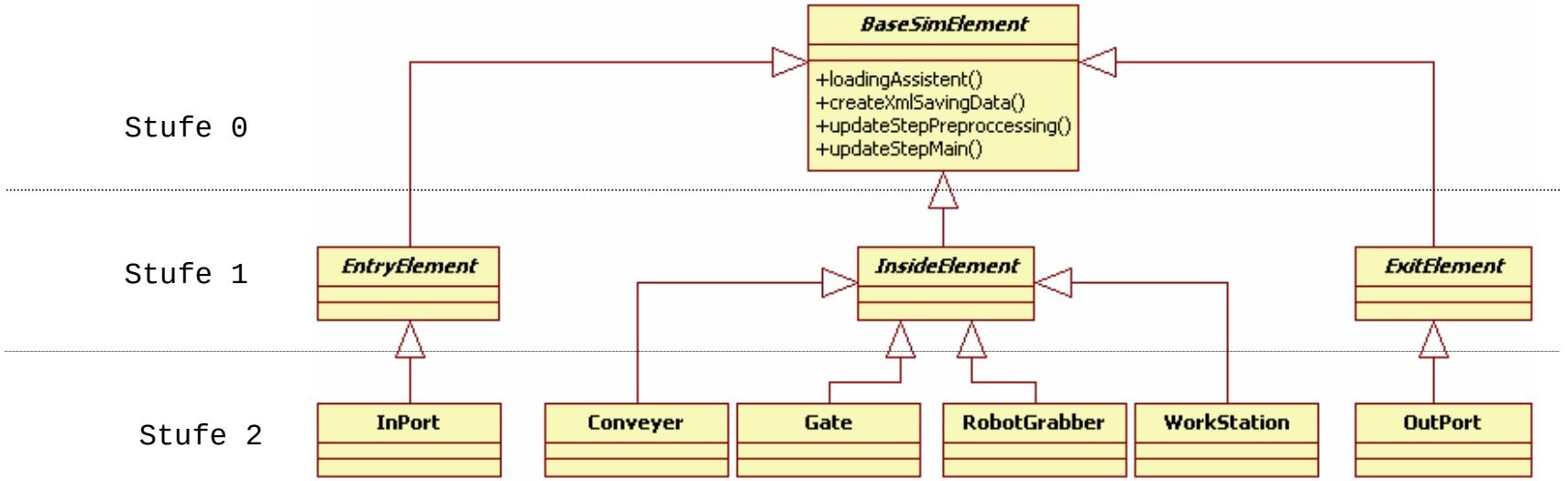
(8.4) Modell-Framework: Realisierung der Modell-Komponenten

- Erinnerung: existierende System-Komponenten = InPort, OutPort, Conveyer, Gate, RobotGrabber und Workstation
- Komponenten lassen sich in Gruppen mit „ähnlichen“ Eigenschaften bündeln
- Resultierende Struktur:



(8.5) Modell-Framework: Implementierung der Modell-Komponenten

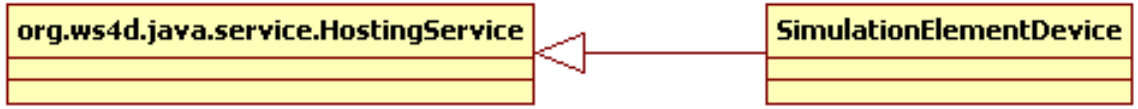
- Klassenstruktur der Komponenten:



- Stufe 0: „Basis-Struktur“ = z. B. Realisierung des Speicherns oder der Protokollierung; ferner Vorgabe von abstrakten Methoden
- Stufe 1: „Verbindungs-Struktur“ = Verwaltung der In- / OutConnectoren
- Stufe 2: konkrete Implementierungen der verschiedenen Komponenten inkl. des komponentenspezifischen Verhaltens

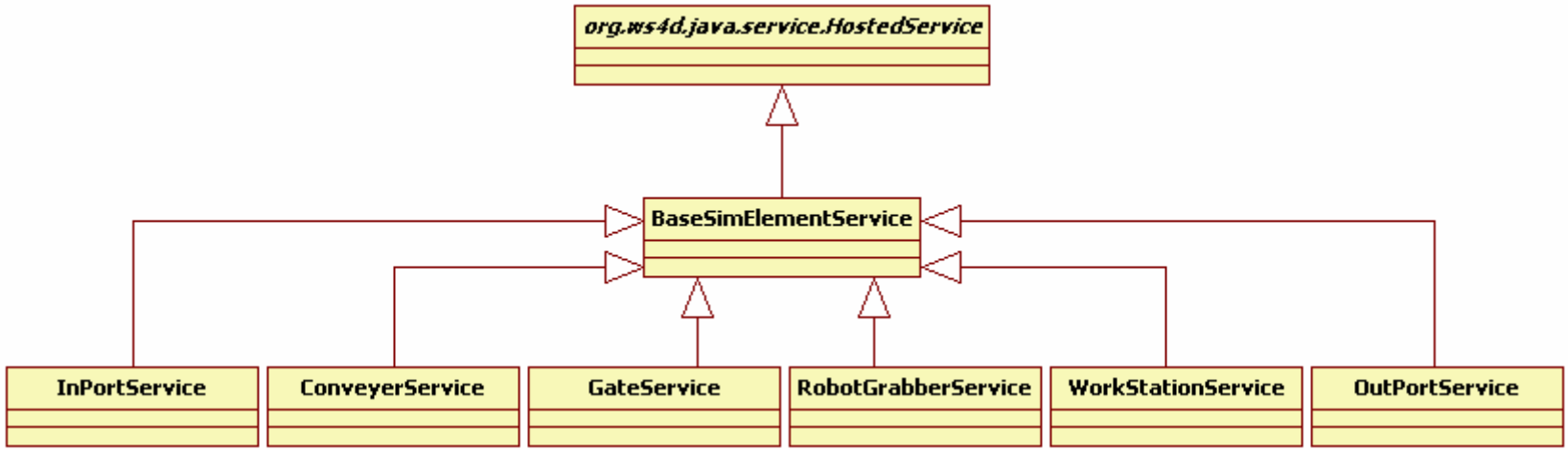
(8.6) Modell-Framework: Web Service der Modell-Komponenten

- „Universal-Device“ für alle Komponenten: das `SimulationElementDevice`



- Der dem jeweiligen Device hinzugefügte, komponentenspezifische Service definiert die Art bzw. die Funktionen der Device-Instanz

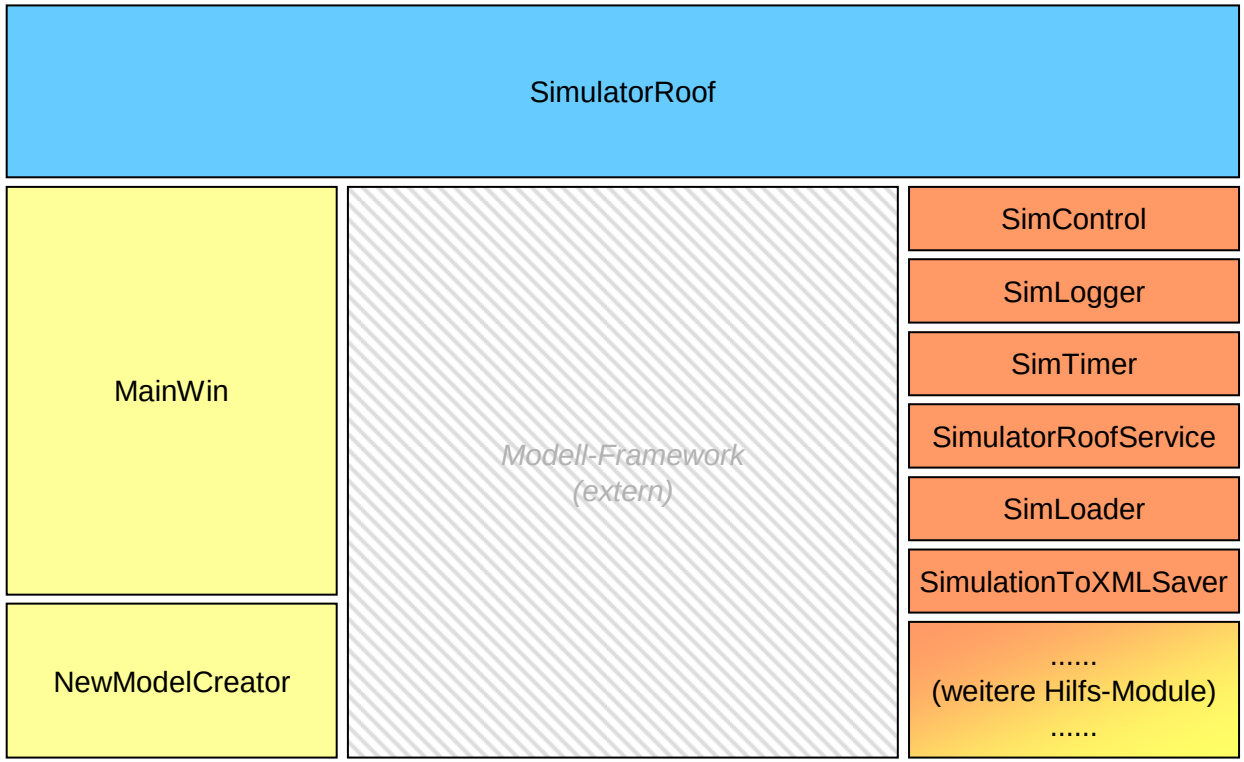
- Struktur der verschiedenen Services:



9. Simulations-Umgebung

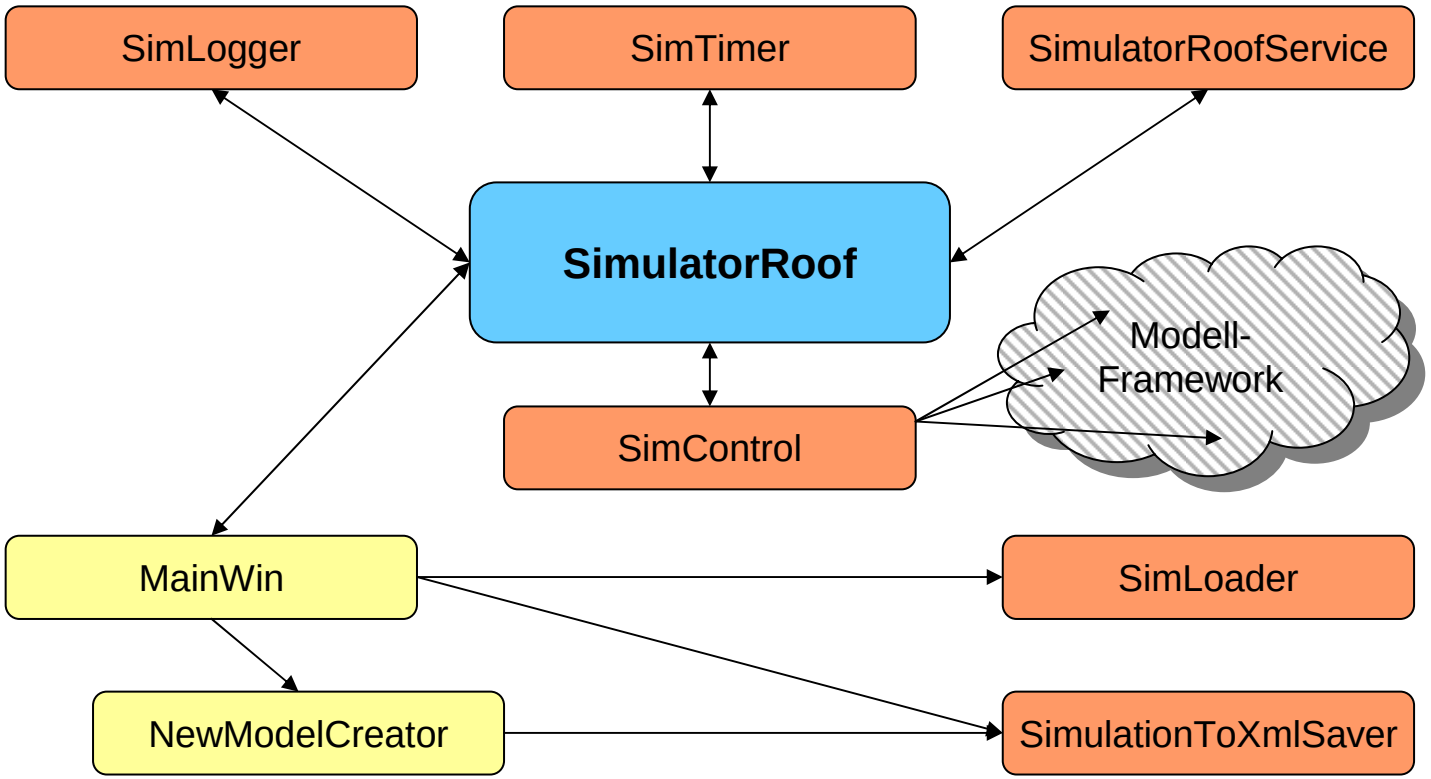
(9.1) Simulations-Umgebung: Module der Umgebung

- Ziel: möglichst modularer Aufbau der Simulations-Software zwecks einfachem Austausch bestimmter Komponenten
- Also: Identifizieren zusammengehöriger Funktionen und zusammenfassen dieser in Module
- Entstehende Module:



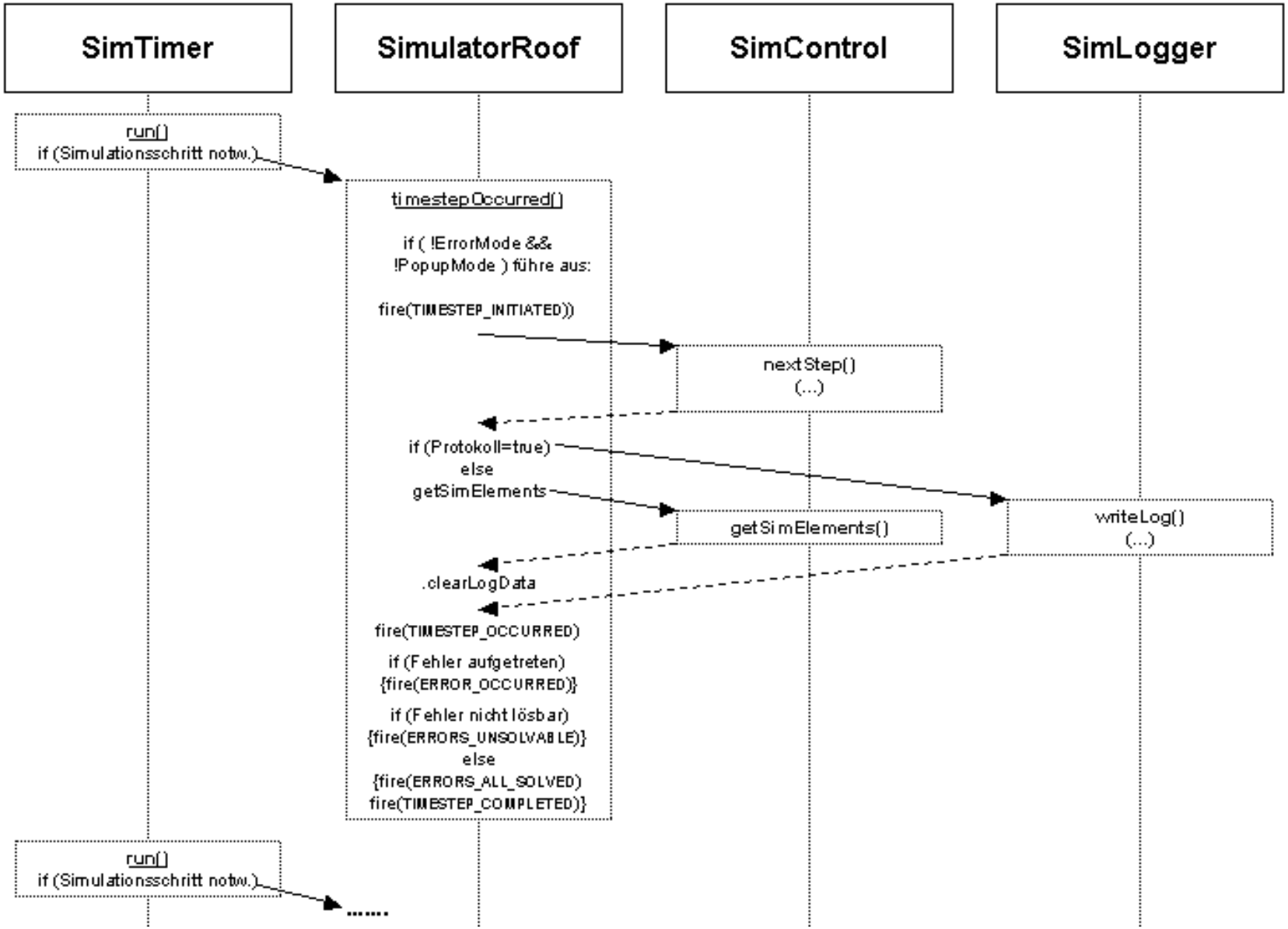
(9.2) Simulations-Umgebung: Interaktionsstruktur der Module

- Grobe Struktur der Modul-Interaktion:



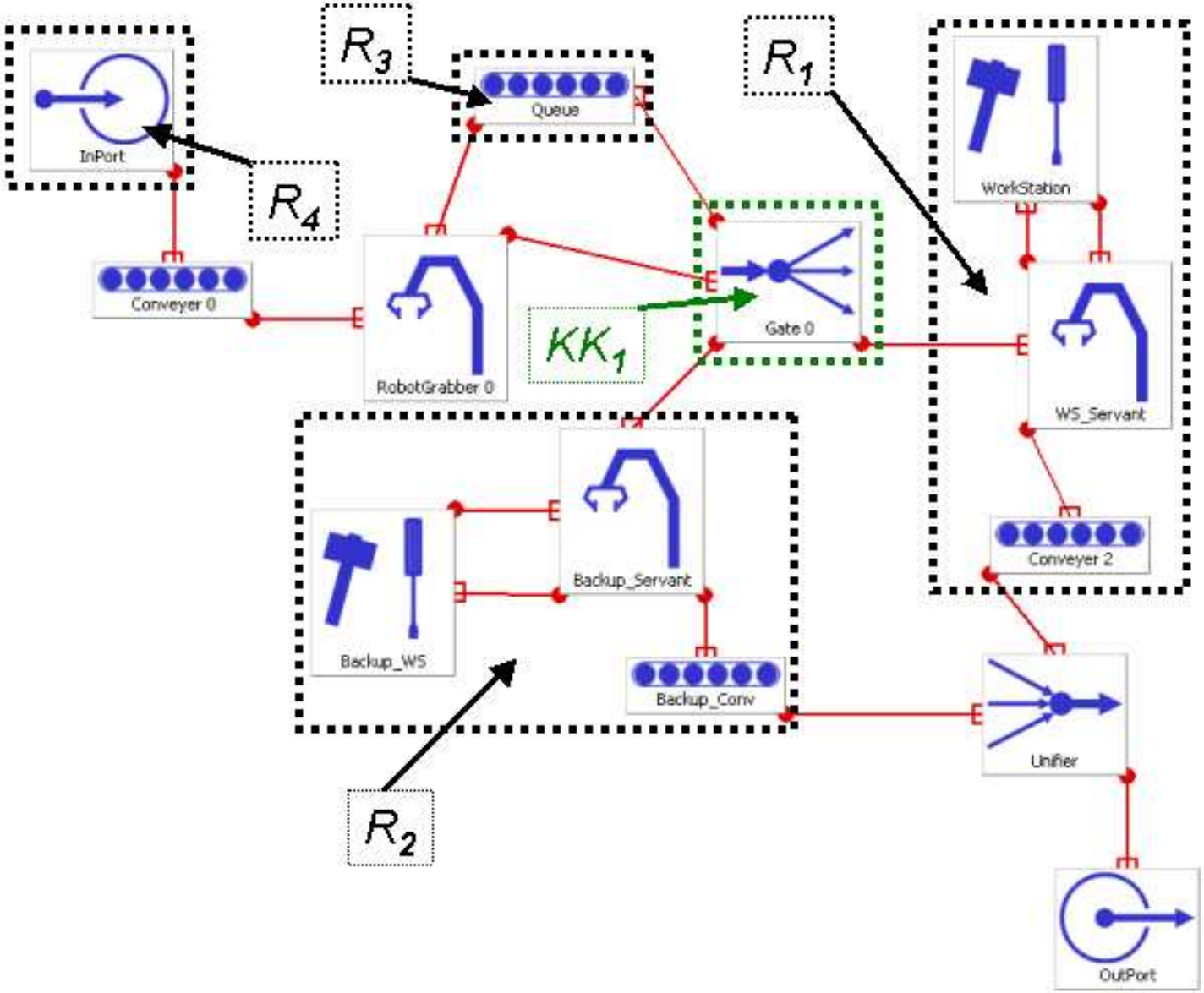
➔ zentrales Modul: SimulatorRoof

(9.3) Simulations-Umgebung: Beispielhafter Ablauf eines Simulations-Schrittes



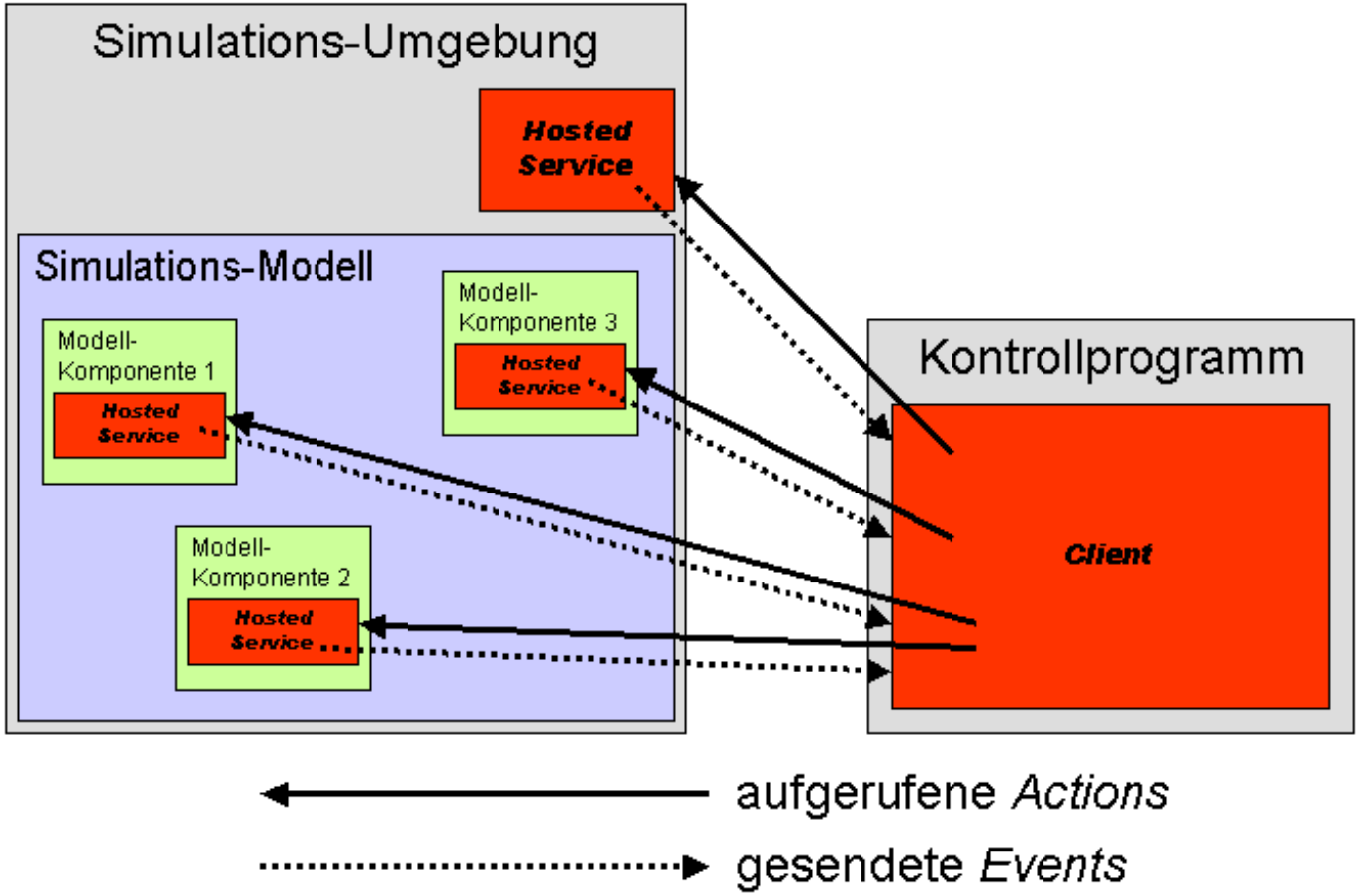
10. Anwendungsbeispiel

(10.1) Anwendungsbeispiel: verwendetes Modell



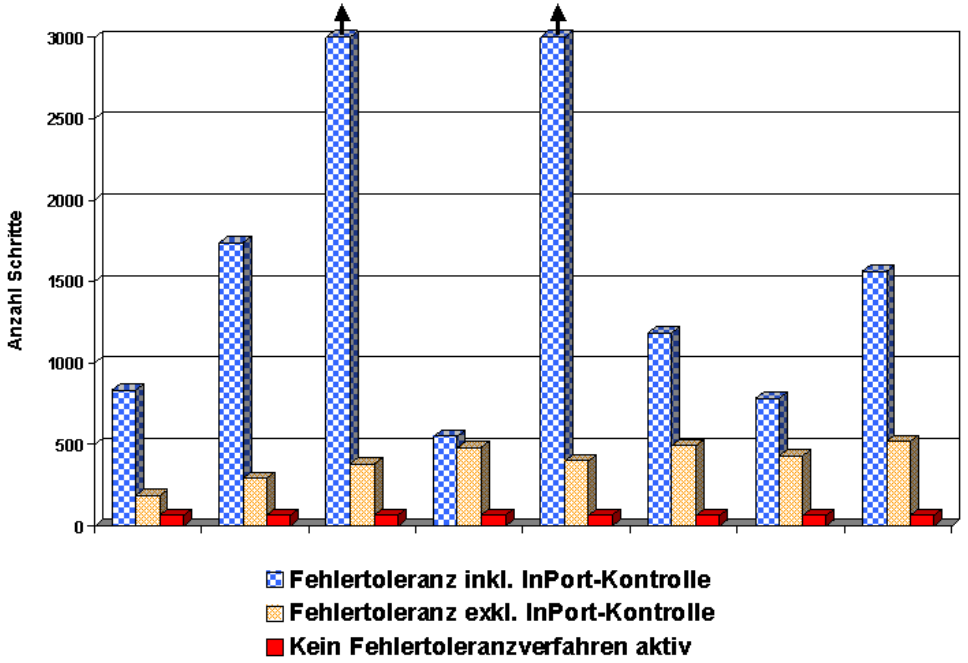
(10.2) Anwendungsbeispiel: Kommunikationsstruktur

- Kommunikationsstruktur Fehlertoleranz-Kontrollprogramm mit Simulation:



(10.3) Anwendungsbeispiel: Ergebnisse

- Durchgeführt wurden jew. 8 Simulationsläufe
  - (a) ohne jegliche Fehlertoleranz-Kontrolle
  - (b) mit Steuerung der Komponente KK_1, ohne Steuerung Payload-Rate
  - (c) mit Steuerung der Komponente KK_1 + Steuerung Payload-Rate
- Erfolgreiche Durchläufe (> 3000 Schritte ohne Abbruch der Sim.) nur bei Laufvariante c
- Durchschnittliche Schrittzahl bis zum Abbruch:
  - Variante a: konstant 65 Schritte
  - Variante b: 399 Schritte
  - Variante c: 1110 Schritte



Fazit...

... Fragen?