Folie: 1_1-Einführung

Software Qualität (2 Sichten)

- *externe Qualität ~> Benutzersicht*
  - ✓ Korrektheit
  - ✓ Zuverlässigkeit
  - ✓ Robustheit
  - ✓ Effizienz
  - ✓ Benutzerfreundlichkeit
  - ✓ (externe) Verständlichkeit
- *interne Qualität ~> Entwicklersicht*
  - ✓ **Wartbarkeit**
  - ✓ Wiederverwendbarkeit
  - ✓ Portierbarkeit
  - ✓ Verständlichkeit (intern)
  - ✓ Interoperabilität

Folie: 1_2-Konzepte

Problembereich – *Analyse und Entwurf* – Model – *Kodierung* – Programm

Mittel der Softwaretechnik

4 Schichten: Konzepte ~> Sprachen (Syntax, Semantik) ~> Methoden (Pragmatik) ~> Werkzeuge

Sprachtypen

- natürlich (z. B. Deutsch)
- formale (Semantik definiert), z. B. Petri-Netze
- informelle (z. B. Semantik nicht eindeutig definiert), z. B. Datenflussdiagramm
- textuell
- graphisch
- hybride (Mischung)

UML Diagrammsprachen

Diagramm	Kategorie
• Klassen- / Objektdiagramme	Strukturdiagramme
• Use Case-Diagramme • Statechart-Diagramme • Aktivitätendiagramme	Verhaltendiagramme
• Sequenzdiagramme • Kommunikationsdiagramme	Interaktionsdiagramme
• Komponentendiagramme • Verteilungsdiagramme	Implementierungsdiagramme
• Object Constraint Language (OCL)	Strukturelle Constraints

Das klassische Wasserfallmodell

1) Machbarkeitsstudie
2) Anforderungsdefinition: Pflichtenheft ~> Was soll Software leisten? — Kapitel II
3) Analyse: Analysedokument ~> Anforderungsanalyse — Kapitel III
4) Entwurf: Entwurfsdokument ~> Wie soll Funktion realisiert werden? — Kapitel IV
5) Implementierung & Test: Quelltext, Programmiersprachen (Alpha-Version) — Kapitel V
6) Auslieferung & Installation: fertiges System (Beta-Version)
7) Wartung: 60% der Softwarekosten

© by Harry Kran

Folie: 2_0 Pflichtenheft Überblick

Gliederung

1. Zielbestimmung: *Warum wird diese Software überhaupt entwickelt?*
2. Produkteinsatz: *Wie sieht das Problemfeld aus?* ***Ist-Zustand!***
   a. Beschreibung des Problembereichs
   b. Glossar
   c. Modell des Problembereichs — Klassendiagramm
   d. Geschäftsprozesse — Aktivitätendiagramm
3. Produktfunktion: *Was soll die Software leisten?* ***Soll-Zustand!*** — Use Case Diagramm
4. Produktcharakteristiken: *Wie soll die Software dies leisten?*

Folie: 2_1 Pflichtenheft Zielbestimmung und Co

1. Zielbestimmung

Inhalt: Gründe für Systementwicklung
Form: Text
Zielgruppe bestimmen

2. Produkteinsatz

a. Beschreibung des Problembereichs
Form: Text und Graphiken
Inhalt: Fachbegriffe erläutern

b. Glossar
Inhalt: Nachschlagewerk für Fachbegriffe
Form: Text mit max. 3 Sätzen

4. Produktcharakteristiken

Wichtig: nicht nur was, sondern auch wie soll es gemacht werden!

a. Systemumgebung
Entwicklungs- und Zielumgebung, Soft- und Hardware

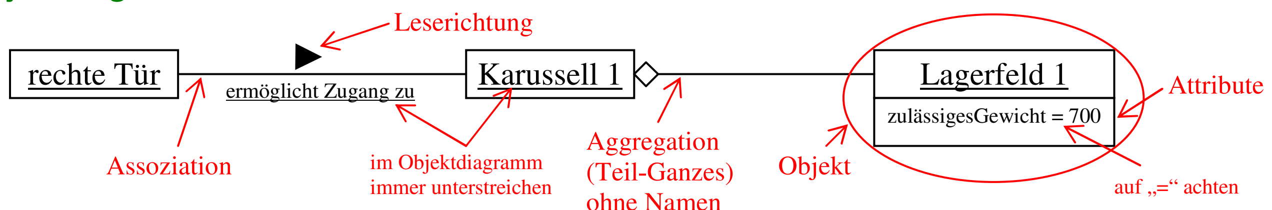
b. Nicht funktionale Anforderungen
In welcher Art soll die Software funktionieren?
Form: Tabelle
Typen:
- USE (Benutzerfreundlichkeit)
- EFFIZIENZ (Effizienz)
- PFLEGE (Wartbarkeit, Wiederverwendbarkeit, Portierbarkeit)
- SICHER (Zuverlässigkeit, Robustheit)
- LEGAL

Folie: 2_2 Modell des Problembereichs

2. Produkteinsatz

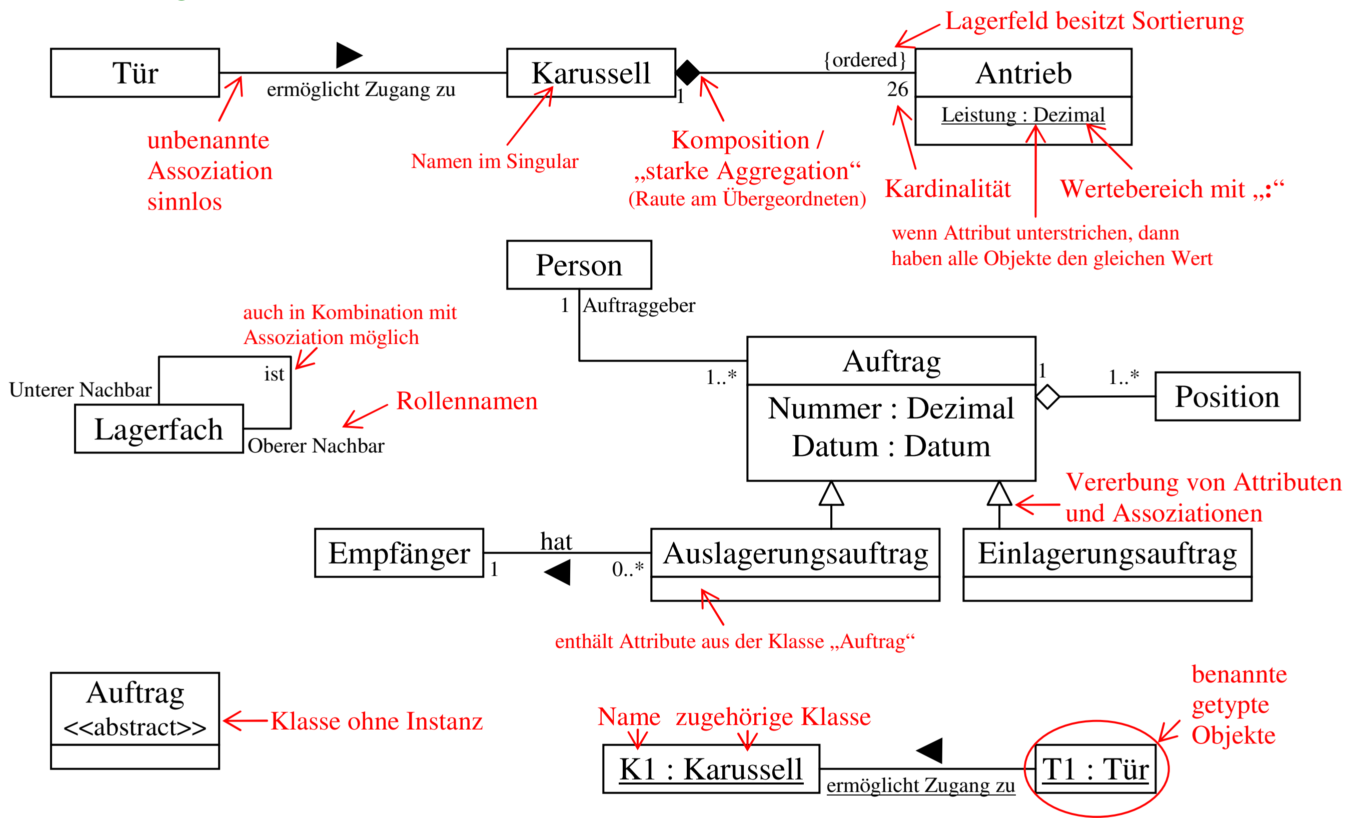
c. Modell des Problembereichs = Menge allgemein möglicher Situationen
Realität – *Abstraktion* – Modell
Beschreibt nur Strukturen: *Welche Begriffe gibt es in der Domäne? Wie hängen diese strukturell zusammen?*

Objektdiagramm



 © by Harry Kran

Klassendiagramm



Folie: 2_3 Geschäftsprozesse

2. Produkteinsatz

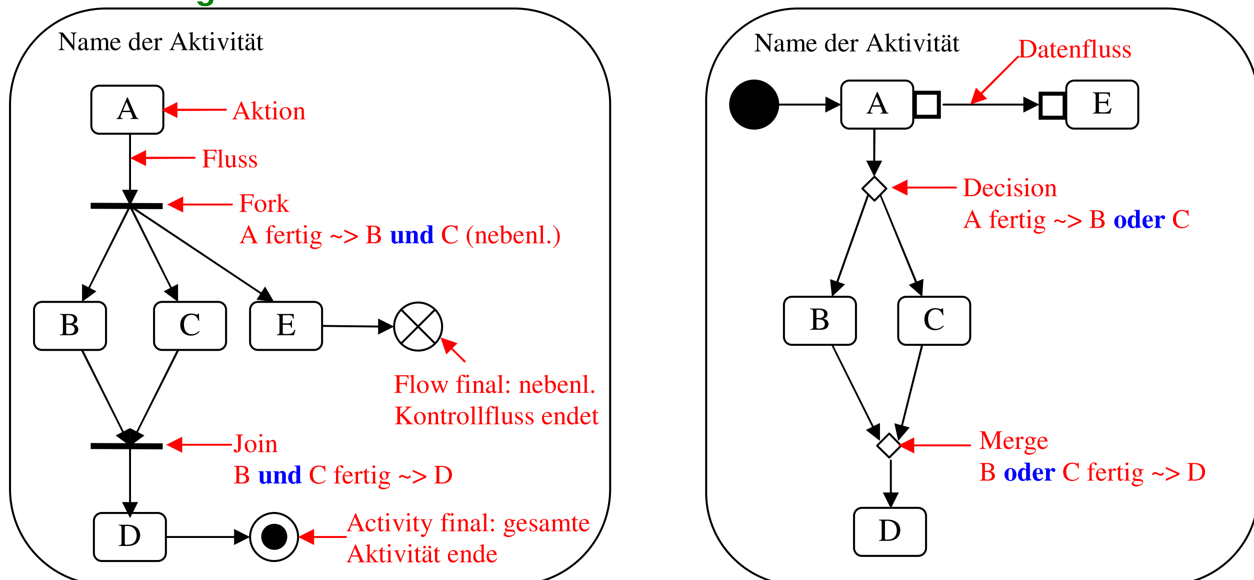
d. Geschäftsprozesse

Beschreibt das Verhalten

Was passiert in der Domäne eigentlich? Wer tut was in welchen Schritten?

Form: graphische Darstellung

Aktivitätendiagramm



Verfeinerung: Aktionen können selbst wieder Aktivitäten sein!

© by Harry Kran

Folie: 2_4 Produktfunktionen

3. Produktfunktionen

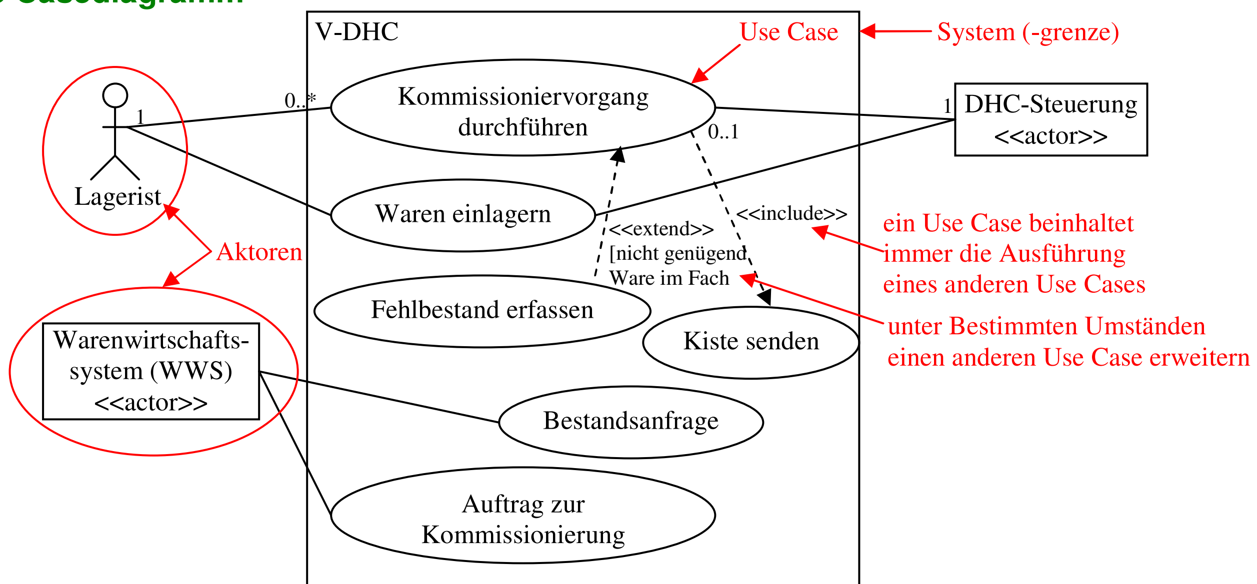
Was soll Software Leisten, um Situation zu verbessern?

- Unterstützung bisheriger Geschäftsprozesse
  - bisherige Arbeitsschritte ersetzten
  - Prozesse verändern
- Ermöglichung neuer Geschäftsprozesse

AG und AN müssen sich über den Funktionsumfang im klaren sein!

1. *Use Case-Diagramm:* Funktionalität des Systems und beteiligte Nutzer
2. *Charakterisierende Informationen:* Tabelle mit Details zu einem Use Case *(Vor- und Nachbedingung)*
3. *Szenarien:* Tabellen mit Erfolgs- und Alternativszenarien eines Use Cases *(Was kann schief gehen? – bel. Konkret)*
4. *GUI-Skizzen* von Szenarien
5. *Aktivitätendiagramm* eines Use Cases *(Stellen den allg. Ablauf dar!)*

Use Casediagramm



Folie: 2_5 Qualität Pflichtenheft

Nur 1/3 aller Projekte ist erfolgreich!

Gründe: Unvollständige Anforderungen, fehlende User-Integration, Spezifikationsänderungen …

Pflichtenheft ist das Fundament jedes IT-Projektes!!!

Anpassungen verursachen Kosten (Basis ~> Implementierung): Wartung 20x, Test 2-4x, Entwurf 0,5x …

Qualitäten

- Korrektheit (Syntax –> Grammatik, Semantik, Pragmatik, Review mit Domainexperten)
- Vollständigkeit (Review mit Anwender)
- Eindeutigkeit (Interpretierbarkeit der Aussagen im PH)
- Konsistenz
- Gewicht (nach Wichtigkeit)
- Überprüfbarkeit
- Änderbarkeit (Redundanzfrei)
- Nachvollziehbarkeit

Folie: 3_1 Architektur *(Analyse: Kapitel 3 im Wasserfallmodell)*

Pflichtenheft – **Architektur** – Quellcode *[leitet sich aus nicht-funktionalen Zielen ab]*

Analyse

1. Architektur festlegen -> Architektur
2. Anforderungen analysieren -> Analyse-Sequenzdiagramme
3. Ergebnisse synthetisieren -> Analyse-Klassendiagramm

 © by Harry Kran

1. ARCHITEKTUR

Architektursichten

- physikalisch (Grundriss, Aufriss, 3-D Ansicht) : Bauarchitektur
- logische (Netzwerkplan, Abwasserplan, Statik) : Softwarearchitektur
- Detaillierungsgrad

3-Sichten-Architektur

- Präsentationsschicht *(Oberfläche (GUI), Dialogkontrolle)*
- Anwendungs- / Logikschicht *(Businesregeln, Interpreter)*
- Datenhaltungsschicht *(Produktdaten, Kundendaten)*

} Betriebssystem Hardware

Softwarearchitektur

- Module mit Eigenschaften/Verhalten
- Beziehungen zwischen Modulen
- Beschreibungen der verbotenen/erlabten Interaktionen

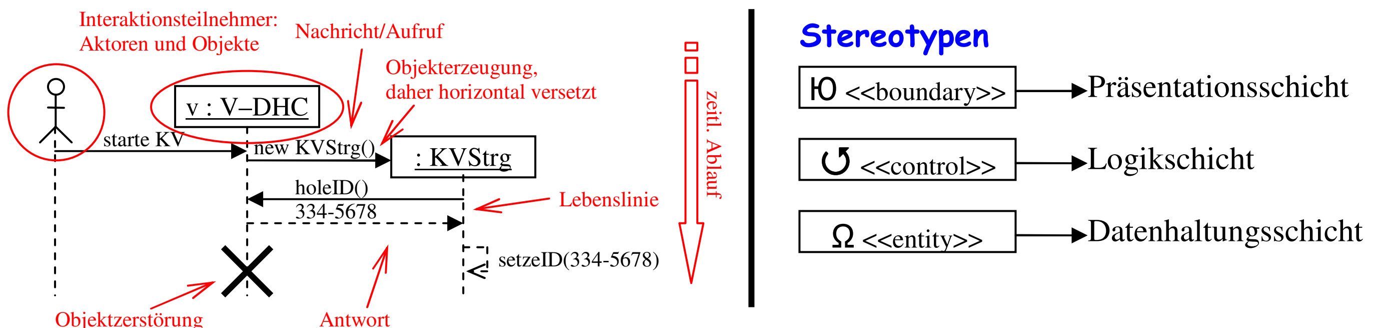
Software-Architekturstile

- Schichtenarchitektur
- Model – View – Controller (MVC) ~> für GUI's in Smalltalk, VT: verschiedene View derselben Daten
- Blackbord Architektur (z. B. Wikipedia): für Systeme der künstlichen Intelligenz
- Service Orientierte Architektur (Gelbe Seiten, Google)

Folie: 3_2 Feinanalyse

2. ANALYSE SEQUENZDIAGRAMME (SD)

Grundlage ist die 3-Schichten Architektur
Dekomposition: datenorientiert, funktional, ***objektorientiert***
Grundidee: Daten und Verhalten in Objekte kapseln, Objekte kommunizierten über Methodenaufrufe, Struktur- (Klassen) und Verhaltensdiagramme (Aktivitäten) kombinieren ~> Sequenzdiagramme



BOUNDARY-KLASSEN *(Übergangsklassen)*

- kapseln Interaktion des Systems von der Umwelt (GUI, Interface, Aktoren, …)
- min eine Klasse je Aktor
- Aktoren kommunizieren nur mit Boundary
- kein Aufruf auf Benutzer möglich

CLONTROLL-KLASSEN *(dürfen nicht auf Boundary-Klassen zeigen)*

- min eine Klasse je Use Case
- nehmen Input der Boundary auf und fordern diese zum Output auf
- kommunizieren mit anderen Control und Entity Klassen zur Aufgabenerfüllung

ENTITY-KLASSEN *(Gegenstandsklassen)*

- werden von Control-Klassen angesprochen
- Klassen des MdP sind *Kandidaten* für Entity Klassen (d. h. nicht alle)
- verkapseln Daten und Verhalten

 © by Harry Kran

Analyse-SD verfeinern Szenarien aus Produktfunktionen
Ausgangspunkt: Use Case Szenario ~> Black Box SD
Für jedes (relevante) Szenario wird eine Analyse (OO Dekomposition) durchgeführt
Ergebnis: Menge von SD, Info über benötigte Klassen

Folie: 3_3 Analyse Klassendiagramm

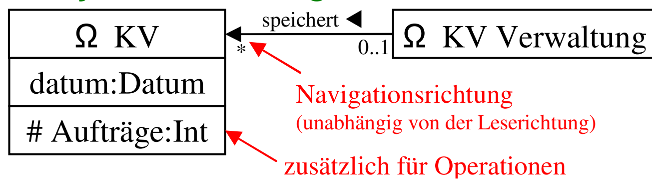
3. Analyse Klassendiagramm

Aus SD werden über Analyse Tabellen Klassendiagramme

Struktur der Tabellen

- Klassen: Objekttyp; ähnliche Namen ~> dasselbe? gleiche Namen ~> unterschiedliche Dinge?
- Aufgaben: ∀ Methoden, die an der Lebenslinie enden; Notation in der Tabelle: Methodenname + „:" + Wertebereich
- Attribute: Notation s. **Aufgaben**; treten bei fast allen Entity Klassen auf; Control Klassen haben „keine"; Boundary Klassen haben Zustandsattribute
- Kennt (dauerhaft?): Wann? – aktive Kommunikation / Objekterzeugung / Referenz; **Dauerhaft**: kennt Objekt über die Ausführung der Methode hinaus

Analyse Klassendiagramm



MdP vs. Analyse-Klassendiagramm

- Beschreibt das Problemgebiet / Beschreibt Lösungsansatz
- Enthält Begriffe des Problembereich / Übernimmt, verwirft, integriert mache Begriffe
- alle Klasse sind gleich / Klassen werden den verschiedenen Schichten zugeordnet
- Assoziationen ungerichtet / Assoziationen gerichtet

Analyse Klasse (entspricht einem Interface) ≠ Implementationsklasse

Folie: 4_1 Komponenten

Kapitel 4: Entwurf

Aufgaben

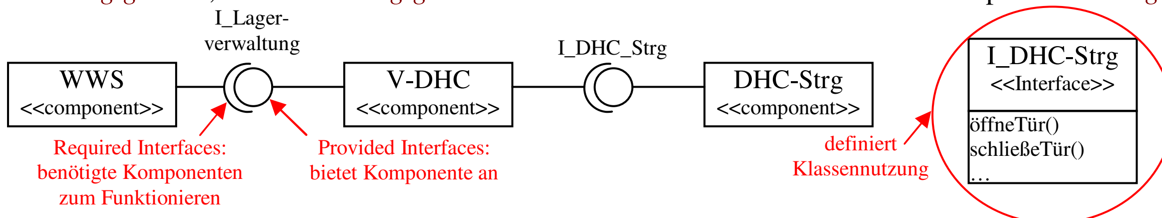
- Systementwurf legt die Realisierung der Softwarefunktionen fest
- Datenstrukturen an Plattform und Programmiersprache anpassen (in Analyse noch nicht)
- Architektur verfeinern
- Integration von Szenarien (Use Cases) zu vollständigen Verhaltensbeschreibungen für einzelne Klassen
- objektlokale Sicht auf Verhalten (Analyse: globale Sicht)

Gliederung

1. Komponentendiagramme: Festlegung der Systemumgebung / Verfeinern der Architektur
2. Statecharts: Verhaltensbeschreibung durch Protokolle
3. Klassendiagramme des Entwurfs: Präzisieren von Attributen, Assoziationen, Operationen
4. Qualität im Entwurf: Heuristiken, Pattern

1. Komponentendiagramme

Eine Softwarekomponente ist ein Baustein mit *vertraglich spezifizierten Schnittstellen* und *ausschließlich expliziten Kontextabhängigkeiten*, kann *unabhängig verwendet* werden und leicht mit anderen Komponenten *integriert* werden

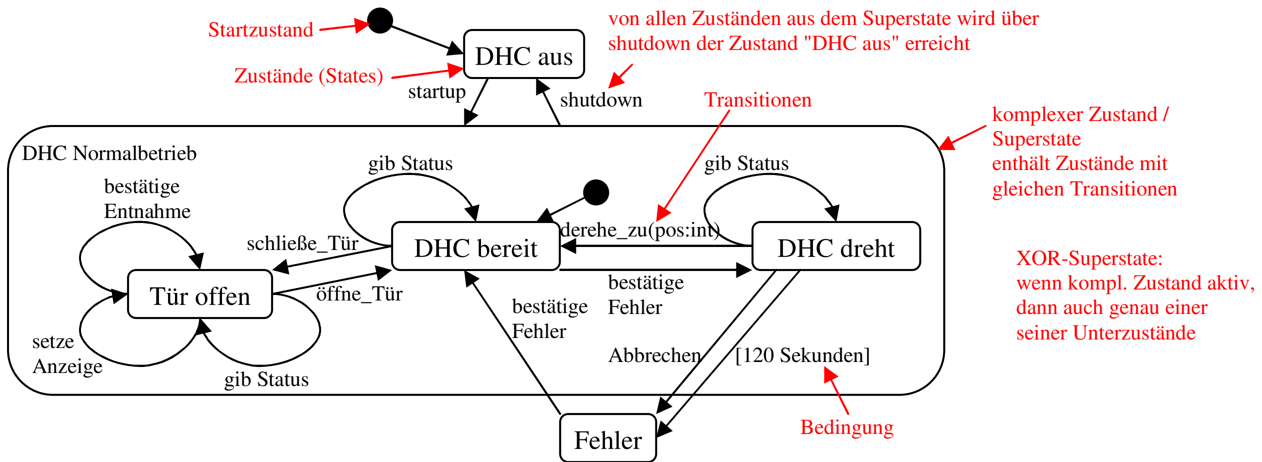


Folie: 4_2 Statecharts Teil1

2. Statecharts *(Zustandsautomaten)*

Protokoll: definierte Aufrufreihenfolge für Operationen eines Interfaces

 © by Harry Kran



Regeln für Zustandsdiagramme *(XOR-Zustände)*

1) Zustandsübergang von einem Zustand (innen oder außen) auf den Rand eines kompl. Zustands entspricht Zustandsübergang zum Anfangszustand des kompl. Zustands
2) Markierter Zustandsübergang vom Rand eines kompl. Zustands zu einem Zustand A (außen) entspricht Zustandsübergängen von jedem Zustand des kompl. Zustand zum Zustand A
3) Unmarkierter Übergang vom Rand eines kompl. Zustands wird bei Erreichen des Endzustands ausgelöst
4) History-State merkt sich den Teilzustand bei Verlassen des kompl. Zustand und aktiviert diesen wieder beim erneuten Betreten
5) Übergang vom Rand eines kompl. Zustands zu einem Zustand A (innen) entspricht Übergängen von jedem Zustand des kompl. Zustands zum Zustand A
6) Innere Transitionen haben eine höhere Priorität

Folie: 4_2 Statecharts Teil2

Regeln: Nebenläufigkeit *(AND-Zustände)*

Unabhängige Transitionen selektieren ~> AND-Superstate: Genau 1 Zustand aus jedem Bereich ist aktiv

1) siehe oben 1) gilt jedoch für alle Compartments (Bereiche)
2) siehe oben 3) es müssen jedoch zunächst aus allen Compartments die Endzustände erreicht worden sein

Bedingte Transition: Transition nur möglich, wenn Zustand aus einem anderen Compartment [in oder nicht in]
pro Klasse gibt es genau ein Statechart / Konsistenzregel: Statechart vs. Aktivitäten- / Analyse-Sequenzdiagramme

Folie: 4_3 Entwurfs Klassendiagramm

3. Klassendiagramme des Entwurfs (KD)

KD erhalten zusätzliche Sprachmittel: Präzisieren von Attributen, Assoziationen, Operationen
Attributsyntax: visibility (+ public / # protected / – private), name multiplicity: type-expression / property String
Property String: changeable / frozen / addOnly ~> Beispiel: # Vermerk[0..5]: AttributedString {addOnly}
Sichtbarkeitsangabe bei Rollennamen und Operationen

Folie: 4_4 Qualität im Entwurf

4. Qualität im Entwurf

Alle Funktionen berücksichtigt? Änderbar? Wartbar?
Gute Lösungen kopieren – Pattern (Lösungsmuster)
Pattern existieren auf Rollenebene und definieren Strukturen und Verhalten
Composite-Pattern: hierarchische Strukturen z. B. Bäume / Observer-Pattern / OO Klassen beste Lösung

Folie: 5_1 Implementierung

UML-Diagramme der Entwurfphase müssen in eine Programmiersprache (PS) übersetzt werden!
Probleme: keine Assoziationen in PS / Vererbung / unterschiedliche Sichtbarkeit
3 Arten der Assoziationsimplementierung: Gegenseitige Pointer (Problem: Referenzielle Integrität) / Einfache Pointer / Assoziationsklassen
Vererbungsimplementierung: Flags (Schalter) / Delegation (Operationen werde Auf- und Abwärts delegiert) / Interfaces + [Delegation] (gute Lösung bei Mehrfachvererbung)
Codegenerierung (lässt sich schwer standardisieren): Probleme: Verhaltensdiagramme und deren Semantik

 © by Harry Kran